Г. І. ГОГЕРЧАК

Інформаційні системи та бази даних

НАВЧАЛЬНИЙ ПОСІБНИК

Київ
Видавництво «Лікей»
2019

УДК [004.03 + 004.65] (075)

Г–58

Рецензенти:
Єгорова Марина Едуардівна, вчитель інформатики Криворізького навчально-виховного комплексу №129 «Гімназія-ліцей академічного спрямування» Криворізької міської ради Дніпропетровської області
Речич Наталія Василівна, вчитель інформатики гімназії №178 міста Києва
Рудик Олександр Борисович, доцент кафедри методики природничо-математичної освіти і технологій Інституту післядипломної педагогічної освіти Київського університету імені Бориса Грінченка, кандидат фізико-математичних наук, Заслужений вчитель України

Рекомендовано до друку
Вченою радою Інституту післядипломної педагогічної освіти
Київського університету імені Бориса Грінченка
протокол № 5 від 23 квітня 2019 року

Гогерчак Г.І.

Г–58 Інформаційні системи та бази даних: [Навч. посіб.] / Г.І. Гогерчак. — К.: Видавництво «Лікей», 2019. — 400 с.

ISBN 978-966-97946-0-4

Посібник призначено для учнів закладів загальної середньої освіти, студентів вищих навчальних закладів та тих, хто опановує бази даних з метою професійного вдосконалення. Видання відповідає блоку «Основи баз даних» схваленої МОН України програми «Офісні інформаційні технології» (автори: В.О. Потієнко, Г.І. Гогерчак). Теоретичний матеріал посібника передбачає повноцінний розгляд проектування баз даних, побудови запитів мовою SQL, а також використання в інформаційних системах. Передбачено використання програмних систем Microsoft Access та MySQL. Приклади та вправи посібника складено у відповідності до найкращих практик роботи з промисловими базами даних та у відповідності до типових завдань Всеукраїнських учнівських олімпіад з інформаційних технологій.

ISBN 978-966-97946-0-4

Зміст

Передмова

Вислів прем'єр-міністра Великої Британії часів Другої Світової Вінстона Черчилля «Хто володіє інформацією, той володіє світом» чи не в повній мірі характеризує той період історії, в якому ми з вами живемо. Комп'ютеризація не лише наукового та військового простору, а й повсякденного життя більшості людей на планеті призвела до того, що кожен може за лічені хвилини отримати доступ до великих обсягів даних, не докладаючи при цьому великих зусиль.

Ще буквально кілька років тому пошук інформації про людину навіть в добре організованому паперовому каталозі вимагав від кількох хвилин до годин часу, залежно від обсягів даних. Отримати ж масив даних з паперового каталогу справа взагалі не з легких, не кажучи вже й про подальшу обробку цих даних та їх певне узагальнення. Уявіть собі процес формування річного звіту великої сучасної корпорації на основі кожного окремого паперового касового чеку – навряд на таку обробку даних зараз би вистачило року...

Поява комп'ютера в широкому вжитку звела час виконання цих ресурсоємних операцій до доль секунди: в електронному вигляді можна зручно й ефективно зберігати набагато більшу кількість даних, аніж на папері, її легко модифікувати, видобувати, обробляти та узагальнювати. Паперові носії в цій конкуренції поступаються електронним за усіма показниками: швидкістю, вартістю, об'ємом носія тощо. Проте якщо робота з паперовими носіями не вимагає особливої підготовки спеціалістів, що працюють зі збереженою на них інформацією, робота з даними, збереженими в електронному вигляді, вже потребує спеціальних навичок. Саме тому найважливішим вмінням для роботи в будь-якій сфері сучасного суспільства (звісна річ, після основ роботи з комп'ютерними пристроями) є вміння працювати з даними.

Мета цієї книги — допомогти читачеві зорієнтуватися в сучасних технологіях роботи з даними та сформувати ґрунтовні знання в області роботи з базами даних різного типу та побудови інформаційних систем на їх основі.

В рамках книги будуть розглянуті:

- базові поняття теорії інформаційних систем та баз даних;
- математичні основи реляційного (табличного) підходу до побудови баз даних;
- принципи моделювання реляційних баз даних та їх фізичного втілення;
- здійснення вибірки даних та маніпуляції ними засобами мови структурованих запитів SQL;
- створення інформаційних систем в середовищі Microsoft Access;
- загальні принципи побудови промислових інформаційних систем з використанням баз даних на прикладі мови програмування Python.

Викладений тут матеріал побудовано таким чином, щоб бути корисним як тим, хто оволодіває базами даних в рамках навчальних курсів у школі чи вищому навчальному закладі, так і тим, хто здобуває відповідні знання чи поглиблює їх самостійно.

В цій книзі центральна увага приділяється роботі з системою Microsoft Access, оскільки на переконання автора вона є найкраще адаптованою саме для навчання роботі з базами даних, а також найбільш популярній системі керування базами даних професійного рівня — MySQL. На основі поданого матеріалу читач зможе легко перейти до роботи з іншими СКБД тих же типів. Після опанування тонкощів роботи з базами даних також оглядово розглядаються засоби роботи зі збереженими в них даними в популярній нині мові програмування Python.

Розглянуті тут практичні проблеми дозволять учням шкіл якісно підготуватися до олімпіад з інформаційних технологій різного рівня та познайомитися з тими аспектами технологій обробки даних, які будуть корисними у разі вибору ними спеціальності, пов'язаної з розробкою програмного забезпечення, аналітикою та комп'ютерними науками в цілому.

Оглядовий матеріал, що стосується побудови промислових інформаційних систем, дозволить читачеві сформувати розуміння поточного стану справ у сфері розробки програмного забезпечення (передусім, веборієнтованого), а також закласти фундамент для подальшого вивчення цього питання.

Всі зацікавлені читачі матимуть змогу не тільки опанувати теоретичний матеріал, а й закріпити його впродовж виконання запропонованих автором завдань та вправ — аж до побудови повноцінної інформаційної системи вкінці опанування цієї книги.

Умовні позначення та інтерактивні елементи

На сторінках цієї книги окрім власне нового матеріалу подано також вправи та виокремлені приклади.

Для зручності вправи до навчального матеріалу виділені в наступні блоки:

Вправа 71

Зміст вправи.

Для кожної вправи в кінці розділу подаються відповіді та вказівки до її розв'язання.

Більшість поданих в цій книзі вправ супроводжуються також спеціальними файлами. Такі супровідні файли розміщені на сайті електронної підтримки книги *http://isdb.itolymp.com/*. Вправи, до яких на цьому сайті подано додаткові файли, позначені в тексті книги спеціальним позначенням:

Вправа 51

Зміст вправи.

Впродовж опанування мови структурованих запитів SQL переважна більшість вправ на побудову запитів на вибірку даних передбачають можливість інтерактивного їх виконання та перевірки в середовищі системи *sql:itolymp*, що доступна за посиланням *https://sql.itolymp.com/*. Для виконання таких вправ слід увійти в спеціальну кімнату, доступну за кодом *isdbbook2018*. Такі вправи позначені в тексті книги наступним позначенням:

Вправа 43

Зміст вправи.

Приклади, що демонструють розв'язання типових завдань, виділені в книзі в наступні блоки та мають аналогічні позначення:

Приклад 9

Завдання прикладу.
Розв'язання прикладу.

Поради щодо використання посібника у підготовці до Всеукраїнської учнівської олімпіади з інформаційних технологій

Всеукраїнська учнівська олімпіада з інформаційних технологій проводиться відповідно до *Положення про Всеукраїнські учнівські олімпіади, турніри, конкурси з навчальних предметів, конкурси-захисти науково-дослідницьких робіт, олімпіади зі спеціальних дисциплін та конкурси фахової майстерності*, затвердженого наказом Міністерства освіти і науки, молоді та спорту України №1099 від 22.09.2011 та зареєстрованого в Міністерстві юстиції України 17.11.2011 за №1318/20056 у чотири етапи.

Кожен з етапів зазвичай передбачає виконання завдань у чотирьох додатках професійного офісного пакету: текстовому процесорі Word, редакторі презентацій PowerPoint, табличному процесорі Excel та системі управління базами даних Access.

В рамках цього посібника розглядається переважна більшість засобів, необхідних учаснику для успішного виконання завдань олімпіади різних рівнів, а також основні класи завдань, які зазвичай там пропонуються. Багато завдань різних етапів цієї олімпіади подаються як приклади або вправи з наведенням повного алгоритму їх розв'язання.

Успішна і ґрунтовна підготовка до подібних змагань в контексті баз даних на думку автора повинна супроводжуватися не тільки отриманням навичок роботи з інструментальними засобами офісного пакету, а й методичним вивченням базових принципів моделювання та реалізації баз даних, побудови запитів мовою SQL та інтерфейсів в середовищі Microsoft Access.

Даний посібник пропонує методичний виклад потрібної теоретичної інформації у поєднанні з практичним відпрацюванням набутих знань. Автор сподівається, що ця книга стане вірним путівником у світ баз даних та приведе читача до успіхів на всіх етапах змагання.

Вступ

Центральними в теорії інформаційних систем та баз даних виступають поняття даних та інформації.

Означення 1. *Інформація — це знання, які зменшують чи прибирають невизначеність щодо виникнення певної події серед заданої множини можливих подій.*

Означення 2. *Дані — це представлення інформації у формалізованому вигляді, придатному для передачі, інтерпретування та обробки.*

Попри те, що в широкому вжитку ми схильні ототожнювати ці два поняття, насправді вони мають різне смислове навантаження. Для цього достатньо розглянути простий приклад. Набір з тридцяти дійсних чисел, записаних через кому — це дані, їх легко зберегти (в тому числі й за допомогою комп'ютера), обробити (наприклад, обчислити середнє значення), але цей набір не несе жодної інформації, допоки ми не вкажемо, що ці дані — це середні температури повітря за кожну добу квітня 2019 року у місті Києві.

В сучасному світі ми маємо справу з усе більшою кількістю даних. Наше завдання — отримати з доступних нам даних певну інформацію і навпаки — зберігати інформацію у вигляді даних так, аби не втратити її суті. Але для отримання інформації спочатку дані потрібно якимось чином зберігати. Саме для цих цілей і використовують найчастіше бази даних.

Означення 3. *База даних — спільно використовувана колекція логічно пов'язаних даних та опис цих даних, організовані для задоволення інформаційних потреб певної організації.*

Означення 4. *База даних — незалежна від прикладних програм сукупність пов'язаних даних, організованих за певними правилами, які передбачають загальні принципи опису, зберігання і маніпулювання.*

За моделлю організації даних, тобто формальними правилами подання даних, маніпуляції ними та накладання обмежень цілісності, розрізняють ієрархічні, мережеві, реляційні та об'єктно-орієнтовані бази даних.

Бази даних з ієрархічною моделлю є одними з найстаріших. Така модель передбачає подання даних у вигляді деревовидної (ієрархічної) структури. В цій структурі кожен об'єкт може мати будь-яку кількість об'єктів-нащадків і не більш, ніж одного об'єкта-предка. Найбільш розповсюджений приклад такої бази даних – файлова система ОС Windows, що складається з кореневого каталогу (локального диску), в якому міститься ієрархія підкаталогів і файлів.

Мережеві бази даних є розширенням ієрархічних в тому аспекті, що об'єкт в такій моделі може мати більше, ніж одного об'єкта-предка. Саме такою моделлю послуговуються файлові системи Unix-подібних операційних систем, в яких підкаталоги та файли можуть міститися одночасно в кількох каталогах.

В об'єктно-орієнтованих базах даних дані моделюються у вигляді об'єктів, їх властивостей, методів та класів. Ця модель максимально наближена до принципів об'єктно-орієнтованого програмування і була створена для того, щоб подання даних на жорсткому диску та в програмному забезпеченні були максимально наближені одне до одного і не вимагали побудови додаткових інтерфейсів.

Бази даних з реляційною моделлю передбачають подання даних у вигляді відношень (таблиць). Один із наступних розділів ми присвятимо вивченню саме цього класу баз даних, оскільки системи керування базами даних саме з реляційним підходом є на сьогодні найбільш широковживаними в сфері розробки програмного забезпечення.

Означення 5. *Система управління базами даних — це програмно-апаратна система, що надає можливість визначення, створення, маніпулювання, контролю, управління та використання баз даних.*

Повномасштабні системи управління базами даних повинні виконувати наступні функції:

- збереження, видобування та оновлення даних;
- надання доступу до структури наявних баз даних та їх елементів;
- підтримка транзакцій — наборів дій, що виконуються користувачем чи

прикладною програмою для видобування чи зміни вмісту бази даних; при цьому система повинна гарантувати виконання всього набору дій та підтримувати можливість відкочування до початкового стану у разі виникнення на будь-якому етапі транзакції помилок виконання;
- керування паралельністю, тобто управління одночасним доступом до баз даних кількох користувачів;
- відновлення структур баз даних та їх вмісту;
- контроль доступу до даних та підтримка їх цілісності;
- підтримка обміну даними з прикладними програмними продуктами.

Незалежно від типу за моделлю організації даних, система керування базами даних повинна забезпечувати виконання усіх наведених вище функцій в тій чи іншій мірі. Остання функція з перелічених є важливою для того, щоб в будь-який момент розробник деякої прикладної програмної системи міг бути впевненим в коректності та несуперечності даних. Зазвичай цю властивість в теорії баз даних називають цілісністю даних. З цілісними даними, надійно збереженими в базі даних певного типу, можна з впевненістю працювати при побудові прикладних інформаційних систем.

Означення 6. *Інформаційна система — сукупність даних (або баз даних), систем керування базами даних, та прикладних програмних засобів, що функціонують як єдине ціле для забезпечення інформаційних потреб користувачів.*

Інформаційні портали засобів масової інформації, соціальні мережі, офіційні сайти урядових установ чи Вікіпедія – все це яскраві приклади інформаційних систем мережі Інтернет. Всі вони забезпечують в тій чи іншій формі можливість додавання, редагування, перегляду інформації, яка в свою чергу програмними засобами перетворюється в дані, що зберігаються на фізичних носіях.

За ступенем розподіленості виділяють настільні та розподілені інформаційні системи. Настільні системи передбачають, що всі дані та програмні засоби розміщені на одному комп'ютері. В розподілених системах компоненти розділені між різними комп'ютерами.

База даних — фундаментальний компонент інформаційної системи, а тому життєвий цикл інформаційної системи невід'ємно пов'язаний з життєвим циклом системи баз даних, що лежить в її основі. Життєвий цикл інформаційної системи зазвичай складається з етапів планування, збору й аналізу вимог, проектування, створення прототипу, реалізації, тестування та супроводу. Кожен з цих етапів стосується і безпосередньо розробки баз даних відповідної інформаційної системи.

Частина I

Бази даних

Розділ 1

Основні поняття теорії баз даних

Означення 7. *Предметна область — це сфера застосування конкретної бази даних.*

Найчастіше предметна область певної бази даних визначається прикладною програмою або тією її частиною, для якої вона будується. Це, наприклад, навчальний заклад, супермаркет, завод автозапчастин, театр тощо.

Предметна область обмежує набір класів (типів) об'єктів — сутностей, що зберігатимуться в базі даних. Природньо, в базі даних навчального закладу можуть зберігатися дані про вчителів, учнів, предмети, але, наприклад, дані про товари та небесні тіла навряд чи мають в ній зберігатися.

Означення 8. *Сутність — це множина однотипних об'єктів.*

В якості сутності можуть виступати особа, місце чи річ, поняття чи подія, які потрібно зберегти в базі даних. Вчитель, учень, предмет, товар, небесне тіло – все це приклади сутностей різних предметних областей.

Як ми зазначили раніше, набір сутностей бази даних обмежується її предметною областю. Зокрема в деяких предметних областях одна й та ж множина однотипних об'єктів може бути сутністю, а в інших, навіть пов'язаних за змістом, — ні. Наприклад, для предметної області «Навчальні заклади» сутності «Школа», «Клас», «Учень» є природніми. Проте, для предметної області «Школа», яка обмежує розгляд до однієї окремо взятої школи, сутність «Школа» вже не є властивою — тут навчальний заклад виступає в

якості всієї предметної області, а не її окремої частини.

Слід зауважити, що в текстовому описі предметної області сутності з граматичної точки зору зазвичай (але не завжди) представлені іменниками.

Вправа 1

Серед поданого нижче переліку визначте сутності предметної області «Факультет університету»:

- «факультет»;
- «аудиторія»;
- «назва предмету»;
- «кафедра»;
- «студент»;
- «викладає»;
- «предмет»;
- «розклад».

Для кожного об'єкта ми можемо визначити ті характеристики, які мають бути збережені для нього в базі даних. Ці характеристики прийнято називати атрибутами сутності.

Означення 9. *Атрибут — властивість, яка описує деяку характеристику описуваного об'єкта.*

Атрибутами учня можуть бути, наприклад, «прізвище», «ім'я» та «по батькові», «дата народження», «номер особової справи» тощо. Атрибути продовольчого товару — «штрих-код», «назва», «роздрібна ціна», «кількість на складі».

Об'єкти однієї сутності відповідно до визначення мають однаковий набір атрибутів. Наприклад, усі учні характеризуються прізвищем, іменем, по батькові, датою народження, номером свідоцтва, класом навчання тощо. Проте для вчителя атрибут «клас навчання» вже є незастосовним.

Атрибути можуть бути простими (складатися з одного неподільного компонента) та складеними (складатися з кількох компонент, наприклад атрибут «ПІБ» складається з прізвища, імені та по батькові). Також виділяють однозначні (приймають одне значення для одного об'єкта) та багатозначні (можуть приймати багато значень для одного об'єкта) атрибути. Прикладом багатозначного атрибута є «номер телефону»: однозначно визначити його за людиною неможливо, оскільки вона може мати кілька номерів телефону.

Вправа 2

Які сутності можуть мати предметні області «Супермаркети», «Житлові будинки»? Визначте атрибути цих сутностей.

В силу необхідності ідентифікації одного об'єкта поміж інших вводиться поняття ключа сутності.

Означення 10. *Ключ (або потенційний ключ) — мінімальний набір атрибутів, значення якого однозначно визначає об'єкт серед усіх об'єктів заданої сутності.*

Наприклад, ключем сутності «Товар» може бути визначений атрибут «штрих-код», який є унікальним для кожного товару. В цій ситуації маючи значення штрих-коду ми можемо однозначно ідентифікувати конкретний товар, а отже й решту його властивостей (атрибутів).

Ключ може складатися також з кількох полів. Наприклад, ключем сутності «Вчитель» можна визначити атрибути «серія» та «номер паспорта», адже їх комбінація є унікальною (принаймні в межах однієї держави). Тут за серією та номером паспорта ми можемо однозначно визначити вчителя, якому цей паспорт належить, а отже й значення усіх його атрибутів, збережених в базі даних. Такий ключ часто називають складеним. У випадку складеного ключа значення окремих атрибутів, що його утворюють, можуть повторюватись, проте їх комбінація завжди є унікальною.

З іншого боку, набір полів «серія», «номер паспорта» та «дата народження» вже не є ключем. Хоча значення цього набору однозначно визначає об'єкт сутності вчитель, проте ця властивість зберігається навіть якщо прибрати атрибут «дата народження», а отже набір не є мінімальним.

З іншого боку, сутність може мати декілька різних ключів. Наприклад, для сутності «Вчитель» окрім набору атрибутів «серія» та «номер паспорта» ключем є також атрибут «ідентифікаційний номер платника податків». Слід зауважити, що комбінація атрибутів «ідентифікаційний номер платника податків», «серія» та «номер паспорта» ключем сутності «Вчитель» не буде з тих же міркувань.

Означення 11. *Первинний ключ — це ключ, обраний для ідентифікації об'єкта сутності.*

Вправа 3

1. Знайдіть ключі сутності «Працівник», якій властиві атрибути:
   - «прізвище»;
   - «ім'я»;
   - «по батькові»;
   - «країна»;
   - «місто»;
   - «паспорт».
2. Знайдіть ключі сутності «Користувач», якій властиві атрибути:
   - «номер в списку»;
   - «електронна пошта»;
   - «логін»;
   - «прізвище»;
   - «ім'я»;
   - «номер телефону».
3. Знайдіть ключі сутності «Учень», якій властиві атрибути:
   - «номер свідоцтва про народження»;
   - «прізвище»;
   - «ім'я»;
   - «по батькові»;
   - «ідентифікаційний код платника податків»;
   - «номер особової справи».

В ході розв'язання завдань вищенаведеної Вправи 3 можна помітити, що ключ сутності залежить від того, за яких умов працюватиме та чи інша база даних. Наприклад, сутність «Президент» матиме ключ «назва держави», якщо в її зміст ми вкладатимемо діючих президентів держав світу, складений ключ «назва держави» і «дата вступу на посаду» – якщо сутність представлятиме всіх діючих і екс-президентів держав світу, ключ «дата вступу на посаду» – якщо об'єктами сутності будуть президенти України за час її незалежності.

Вправа 4

1. Які ключі та за яких умов матиме сутність «Клас школи»?
2. За яких умов ключем сутності «Книга» в предметній області «Бібліотека» буде комбінація атрибутів «ПІБ автора» та «назва»?

Можливі випадки, коли жодна комбінація атрибутів не володіє властивостями, необхідними для формування ключа. В такому разі до сутності

додається додатковий унікальний атрибут – так званий штучний ключ.

Якщо розглянути сутність «Товар» предметної області «Супермаркети», то атрибут «штрих-код» насправді є штучним ключем цієї сутності, який колись було додано в перелік характеристик товарів задля їх ідентифікації.

Вправа 5

1. Визначте ключі сутності «Авіарейс», якій властиві атрибути:
   - «аеропорт відправлення»;
   - «аеропорт посадки»;
   - «дата відправлення».
2. Визначте ключі сутності «Громадянин», якій властиві атрибути:
   - «прізвище»;
   - «ім'я»;
   - «дата народження».

Природньо, що об'єкти можуть бути одне з одним пов'язані. Вчитель *Іванова Марія Іванівна* навчає учнів *української мові*, учень *Бабанський Олександр* навчається в *11-Б* класі, покупець *Петренко Раїса Андріївна* придбала один буханець *бородинського хліба* тощо. Якщо уважно прослідкувати за цими висловлюваннями, їх можна дещо узагальнити і замість зв'язків між об'єктами сформулювати відповідні їм зв'язки між сутностями: «вчитель навчає предмету», «учень навчається в класі», «покупець придбав товар».

Означення 12. *Зв'язок — це те, що поєднує декілька сутностей.*

Зв'язок також може мати свої атрибути. Наприклад, у зв'язку «студент складає іспит» його атрибутом є результат складання — «оцінка».

Слід зауважити, що в текстовому описі предметної області зв'язки з граматичної точки зору зазвичай (але не завжди) представлені дієсловами.

Вправа 6

1. Для обраних сутностей предметної області «Факультет університету» (Вправа 2) визначте зв'язки між ними.
2. Для обраних сутностей предметних областей «Супермаркети»,

«Житлові будинки» (Вправа 2) визначте зв'язки між ними.

В деяких випадках, коли атрибутів сутності недостатньо для формування її ключа, можна запозичити ключ іншої сутності, пов'язаної з цією зв'язком, для формування унікального набору атрибутів. Наприклад, для ідентифікації навчального класу серед інших класів шкіл міста ми можемо запозичити ключ пов'язаної сутності «Школа» — «номер цієї школи». Тоді комбінація атрибутів «номер школи» та «назва класу» буде ключем сутності «клас».

Означення 13. *Зовнішній ключ — це сукупність атрибутів сутності, які складають ключ іншої, пов'язаної з нею, сутності.*

Вправа 7

Визначте атрибути, що можуть бути зовнішніми ключами, для сутності «Авіарейс», якій властиві атрибути:

- «аеропорт відправлення»;
- «аеропорт посадки»;
- «дата відправлення».

Вкажіть сутності, з яких відповідні зовнішні ключі могли бути запозичені.

Узагальнюючи вищесказане, можна дати й таки визначення сутності:

Означення 14. *Сутність — це множина об'єктів, що мають однаковий набір атрибутів та однотипні зв'язки з іншими об'єктами.*

1.1 Відповіді та вказівки до розв'язання вправ

№1. «Аудиторія», «Кафедра», «Студент», «Предмет» — сутності цієї предметної області, оскільки мають свої спільні характеристики.

«Факультет» не є сутністю в предметній області «Факультет університету», адже він власне є всією предметною областю. «Назва предмету» — це характеристика об'єкту, а не сам об'єкт. «Викладає» та «розклад» скоріше поєднують сутності між собою, аніж є ними.

№2. Предметна область «Супермаркети» може мати сутності:

- «Товар» з атрибутами «штрих-код», «назва», «ціна»;
- «Виробник» з атрибутами «назва», «адреса»;
- «Склад» з атрибутами «назва», «адреса»;
- «Працівник» з атрибутами «номер паспорта», «прізвище», «ім'я», «по батькові»;
- «Супермаркет» з атрибутами «назва», «адреса».

Предметна область «Житлові будинки» може мати сутності:

- «Будинок» з атрибутами «вулиця», «номер будинку», «кількість поверхів»;
- «Під'їзд» з атрибутами «номер», «адреса»;
- «Квартира» з атрибутами «номер», «площа»;
- «Житель» з атрибутами «номер паспорта», «прізвище», «ім'я», «по батькові».

№3.

1. Ключем сутності «Працівник» є атрибут «паспорт»; комбінація атрибутів «прізвище», «ім'я», «по батькові» ключем сутності не буде, оскільки співпадіння значень цих атрибутів у двох різних людей цілком можливе.
2. Ключами сутності «Користувач» є атрибути «номер в списку», «електронна пошта», «логін», «номер телефону» (кожен окремо); атрибут «номер телефону» може бути ключем, оскільки один номер телефону теж однозначно визначає його власника.
3. Ключами сутності «Учень» є атрибути «номер свідоцтва про народження», «ідентифікаційний код платника податків», «номер особової справи» (кожен окремо); комбінація атрибутів «прізвище», «ім'я», «по батькові» ключем сутності не буде, оскільки співпадіння значень цих атрибутів у двох різних людей цілком можливе.

№4.

1. Ключем сутності «Клас школи» може бути атрибут «назва», якщо розглядаються класи в межах однієї школи, комбінація атрибутів «назва» та «номер школи» — якщо класи в межах міста, комбінація атрибутів «назва», «номер школи» та «місто» — якщо класи в межах країни.
2. Комбінація атрибутів «ПІБ автора» та «назва» буде ключем сутності «Книга», якщо ця сутність передбачає книги як мистецькі твори. З іншого боку, якщо передбачати збереження даних про кожну книгу як матеріальний об'єкт, ключем можна вважати лише певний унікальний «код книги», адже тираж книжок зазвичай більший за одиницю.

№5.

1. Ключем сутності «Авіарейс» не може бути жодне із зазначених полів, оскільки за одним напрямком в один і той самий день може вилітати два і більше літаків. До цього переліку можна додати атрибут «час відправлення», тоді ключем сутності буде комбінація атрибутів «аеропорт відправлення», «аеропорт посадки», «дата відправлення», «час відправлення». Також можна ввести унікальний «код рейсу», який теж може виступати в ролі ключа цієї сутності.
2. Ключем сутності «Громадянин» не може бути жодне із зазначених полів, оскільки значення всіх полів можуть співпадати у різних людей. Для цього вводиться унікальний «номер паспорта».

№6.

1. Прикладами зв'язків для предметної області «Факультет університету» є «аудиторія належить кафедрі», «кафедра відповідає за викладання предмету», «студент навчається на кафедрі».
2. Прикладами зв'язків для предметної області «Супермаркети» є «товар виготовляється виробником», «товар зберігається на складі», «працівник керує складом». Прикладами зв'язків для предметної області «Житлові будинки» є «під'їзд належить до будинку», «квартира належить до під'їзду», «житель є власником квартири».

№7. Раніше ми визначили ключ сутності «Авіарейс» — комбінацію атрибутів «аеропорт відправлення», «аеропорт посадки», «дата відправлення», «час відправлення». Атрибути «аеропорт відправлення» та «аеропорт посадки» тут є зовнішніми ключами і запозичені з сутності «Аеропорти» для формування ключа сутності «Авіарейс».

Розділ 2

Проектування баз даних

Означення 15. *Проектування баз даних – це процес створення проекту бази даних.*

Основна мета процесу проектування бази даних – ефективне з точки зору логіки, часу виконання та займаної пам'яті представлення даних (сутностей разом з відповідними їм атрибутами) та зв'язків між ними, що необхідні для всіх областей використання майбутньої інформаційної системи та всіх груп майбутніх її користувачів.

Вищевказана мета для великих баз даних може досягатися за допомогою моделювання даних, кінцевим продуктом якого є модель, якій властиві наступні характеристики:

- простота (легкість розуміння моделі професіоналами та непрофесіоналами);
- виразність (можливість відрізнити різні сутності, зв'язки між ними і обмеження, що накладаються, одне від одного);
- ненадлишковість (модель передбачає збереження кожного факту рівно один раз);
- розширюваність (можливість оновлення моделі у зв'язку з додаванням нового функціоналу інформаційної системи);
- можливість представлення у вигляді зрозумілих діаграмних позначень.

Проектування баз даних здійснюється в три фази:

1. Концептуальне проектування: створення концептуальної моделі даних, незалежної від вибору моделі організації даних та конкретної системи

керування базами даних.
2. Логічне проектування: створення на основі концептуальної моделі логічної моделі даних відповідно до обраної моделі організації даних, але незалежно від конкретної системи керування базами даних.
3. Фізичне проектування: реалізація логічної моделі засобами конкретної системи керування базами даних.

Такий підхід до проектування називається низхідним, оскільки спочатку ідентифікуються предметна область, її сутності, потім – атрибути та зв'язки між сутностями. Цей підхід є найбільш розповсюджуваним та зручним для великих баз даних. Найчастіше в його рамках для побудови реальних баз даних використовується технологія, що базується на моделі «сутність-зв'язок», яку буде розглянуто далі в цьому розділі.

На його противагу, переважно для невеликих за розмірами баз даних використовується висхідний підхід, який передбачає спочатку ідентифікацію всіх необхідних атрибутів, а потім, на основі отриманої сукупності, — вибудовування сутностей та зв'язків між ними. Найчастіше цей процес здійснюється за допомогою нормалізації даних.

Зупинимося детальніше на суті обох підходів.

2.1 Концептуальне проектування бази даних

2.1.1 Модель «сутність-зв'язок»

1976 року Пітером Пін-Шен Ченом, американським професором комп'ютерних наук в університеті штату Луїзіана, була запропонована модель «сутність-зв'язок» (англійською entity-relationship model, часто – ER-модель або ER-діаграма), яка значно спрощує процес концептуального моделювання баз даних.

На ER-моделі графічно зображуються сутності предметної області, її атрибути, а також зв'язки між сутностями та атрибути цих зв'язків. Як ми побачимо згодом, деякі характеристики цих елементів також позначаються на такій діаграмі, що покращує прозорість моделі та її доступність як професіоналам сфери, так і іншим зацікавленим в роботі розроблюваної інформаційної системи особам.

На даний момент існує кілька варіантів графічного зображення вищевказаних елементів на ER-діаграмах (Рисунок 2.1). Ми користуватимемось тією нотацією, якою послуговується безкоштовний онлайн-сервіс ERDplus (*https://erdplus.com/*), аби в читача була можливість зручним чином відтворювати запропоновані далі діаграми та виконувати вправи до цього розділу.

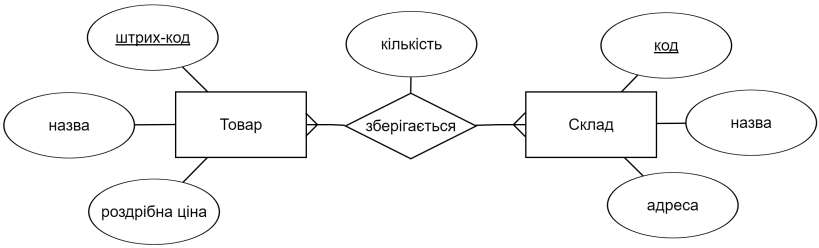


Рис. 2.1: ER-діаграма зв'язку «товар зберігається на складі» з атрибутом «кількість» та атрибутами сутностей

В цій нотації, як власне й у переважній більшості, сутність позначається прямокутником, всередині якого вказано її назву.

Атрибут сутності позначається овалом, всередині якого подається назва атрибуту. Овал з'єднується прямою лінією з прямокутником тієї сутності, якій відповідає зображуваний ним атрибут.

Зазвичай на ER-діаграмах позначають лише ключ, який було обрано в якості первинного. Всі атрибути, які належать до первинного ключа, позначаються підкресленням їх назви на відповідній діаграмі.

Зв'язок між сутностями ми позначатимемо ромбом, в якому вказуватимемо зміст цього зв'язку. Ромб з'єднується з тими сутностями, які залучені до відповідного зв'язку.

Атрибут зв'язку позначається на ER-діаграмі аналогічно до атрибута сутності овалом з написом, проте з'єднується він прямою лінією з ромбом відповідного йому зв'язку.

2.1.2 Класифікація зв'язків за множинністю

Існує декілька критеріїв класифікації зв'язків між сутностями в теорії баз даних. В рамках цього розділу ми розглянемо класифікацію за двома такими критеріями, які є важливими в процесі проектування баз даних та подальшій фізичній реалізації моделі даних в конкретній системі керування базами даних.

Перший з них — це множинність зв'язку.

Означення 16. *Множинність зв'язку (або показник кардинальності) описує кількість можливих зв'язків між об'єктами сутностей-учасниць цього зв'язку.*

Розглянемо зв'язок «виробник виготовляє товар». Природньо, що товар одного ґатунку виготовляється рівно одним виробником. З іншого боку, один виробник може виготовляти багато різних товарів (в теорії баз даних багато — це принаймні більше одного). В такому разі кажуть, що зв'язок має множинність «один до багатьох».

Позначення множинностей здійснюється на ER-моделі наступним чином. З боку сутності, що має множинність «один» (один товар виготовляється одним виробником), на лінії, що з'єднує ромб зв'язку з прямокутником сутності, ми ставимо вертикальний штрих. З боку сутності, що має множинність «багато» (один виробник виготовляє багато товарів), на лінії, що з'єднує ромб зв'язку з прямокутником сутності, ми ставимо «кролячу лапку» (Рисунок 2.2).

Рис. 2.2: Зв'язок «виробник виготовляє товар» з множинністю «один до багатьох»

Розглянемо зв'язок «товар зберігається на складі». Природньо, що товар одного ґатунку може зберігатися одночасно на кількох складах. З іншого боку, на одному й тому ж складі можуть зберігатися різні товари. В такому разі кажуть, що зв'язок має множинність «багато до багатьох». Її позначають «кролячими лапками» з обох боків зв'язку (Рисунок 2.3).

Рис. 2.3: Зв'язок «товар зберігається на складі» з множинністю «багато до багатьох»

Розглянемо зв'язок «працівник керує складом». Зазвичай працівник може керувати не більш, ніж одним складом. З іншого боку, одним складом керує рівно один працівник. В такому разі кажуть, що зв'язок має множинність «один до одного». Її позначають вертикальними штрихами з обох боків зв'язку (Рисунок 2.4).

У вище поданих прикладах для визначення множинностей відповідних зв'язків ми для кожного з них ставили два питання:

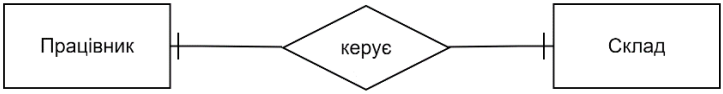


Рис. 2.4: Зв'язок «працівник керує складом» з множинністю «один до одного»

- з одним об'єктом першої сутності скільки об'єктів другої сутності може бути пов'язано в рамках цього зв'язку?
- з одним об'єктом другої сутності скільки об'єктів першої сутності може бути пов'язано в рамках цього зв'язку?

Відповідь на перше питання впливає на позначку поряд з прямокутником другої сутності на ER-діаграмі, а відповідь на друге питання — на позначку поряд з прямокутником першої сутності.

Вправа 8

1. Визначте множинності наступних зв'язків:
   - «вчитель навчає клас»;
   - «чоловік одружений з жінкою»;
   - «студент отримав результат оцінювання з предмету»;
   - «мати народила дитину»;
   - «місто належить до країни»;
   - «гвіздок забито в дошку».
2. Замалюйте розглянуті вище зв'язки на ER-моделі.

Вправа 9

За допомогою ресурсів мережі Інтернет дослідіть формат проведення пісенного конкурсу «Євробачення» у 2018 році. Визначте наявні сутності предметної області та принаймні по одному зв'язку кожної множинності. Замалюйте кожен зі зв'язків на ER-моделі з відповідними позначеннями.

2.1.3 Класифікація зв'язків за ступенем

Зв'язки також класифікують за їх ступенем.

Означення 17. *Ступінь зв'язку — це кількість сутностей, що до нього залучені.*

Досі ми розглядали лише зв'язки ступеня 2, їх ще називають бінарними. Приклад такого зв'язку подано вище (Рисунок 2.4).

Розглянемо в предметній області «Супермаркети» наступну ситуацію. Для продажу товарів в певному супермаркеті необхідно привезти їх з відповідних складів. Як ми вже раніше зазначали, один і той самий товар може зберігатися на кількох складах одночасно. В такому разі, при збереженні факту з якого складу в який супермаркет який товар привезено:

- якщо поєднати тільки товар та супермаркет – не вистачає інформації про те, з якого складу товар привезено;
- якщо поєднати тільки товар та склад – не вистачає інформації про те, в який супермаркет товар привезено;
- якщо поєднати тільки супермаркет та склад – не вистачає інформації про те, який товар привезено.

Таким чином бінарного зв'язку в цій ситуації недостатньо. Насправді у зв'язку товар привезено зі складу в супермаркет залучено одразу три сутності. Такий зв'язок називають тернарним (Рисунок 2.5).

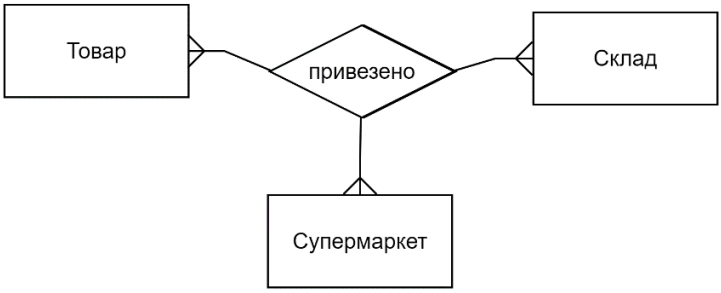


Рис. 2.5: Тернарний зв'язок «товар привезено зі складу в супермаркет»

Слід звернути увагу на те, що тернарні зв'язки, як власне і зв'язки будь-якого іншого ступеню, теж мають множинність. В цьому випадку процес її визначення майже ідентичний випадку бінарних зв'язків, проте кількість питань відповідає кількості залучених сутностей. У випадку зв'язку ступеня

три їх три:

- з кожною парою об'єктів першої та другої сутностей скільки об'єктів третьої сутності може бути пов'язано в рамках цього зв'язку?
- з кожною парою об'єктів першої та третьої сутностей скільки об'єктів другої сутності може бути пов'язано в рамках цього зв'язку?
- з кожною парою об'єктів другої та третьої сутностей скільки об'єктів першої сутності може бути пов'язано в рамках цього зв'язку?

Для вищерозглянутого зв'язку «товар привезено зі складу в супермаркет» питання та відповіді на них матимуть наступні формулювання:

- один і той самий товар з одного й того ж складу у скільки супермаркетів може бути завезено? – у багато;
- один і той самий товар у один й той же супермаркет зі скількох складів може бути завезено? – з багатьох;
- з одного й того ж складу в один і той же супермаркет скільки різних товарів може бути завезено? – багато.

Таким чином множинність зв'язку «багато до багатьох до багатьох» на ER-діаграмі (Рисунок 2.5) позначено правильно. Слід зазначити, що зв'язки інших множинностей зі ступенем більше двох зустрічаються вкрай рідко.

Аналогічним чином визначаються й зв'язки більшого ступеню, наприклад кватернарні.

Існує ще одне особливе значення ступеню зв'язку – один. Такий зв'язок називають унарним або рекурсивним, він поєднує сутність саму з собою. В якості його прикладу розглянемо збереження факту про те, що працівник є підлеглим іншого працівника. В цьому випадку з одного боку сутність працівник виступає в якості підлеглого, з іншого — в якості начальника (Рисунок 2.6).

Рис. 2.6: Унарний зв'язок «працівник керує працівником»

Оскільки в такому вигляді ER-діаграма не надто інформативна щодо

суті зв'язку, для зображення унарних зв'язків часто використовують рольові імена, які вказують призначення кожної сутності-учасниці відповідного зв'язку. В цьому випадку рольовими іменами можна визначити слова «підлеглий» та «начальник» (Рисунок 2.7).

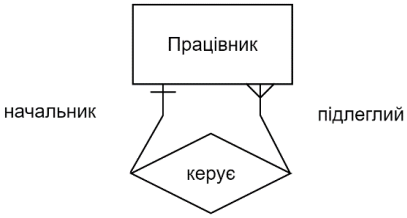


Рис. 2.7: Унарний зв'язок «працівник керує працівником» з рольовими іменами

Множинність розглянутого зв'язку визначається аналогічно до множинності бінарного зв'язку. В даному випадку підлеглий може мати не більше одного безпосереднього начальника, а начальник — багатьох підлеглих. Отже, зв'язок має множинність «один до багатьох».

Вправа 10

І тур І етапу І Всеукраїнської учнівської Інтернет-олімпіади з інформаційних технологій
Зобразіть на ER-діаграмі зв'язок між двома футбольними командами, що грають матч Ліги Чемпіонів, із зазначенням результату матчу. Зазначте множинність зв'язку.

Вправа 11

За допомогою ресурсів мережі Інтернет дослідіть формат проведення пісенного конкурсу «Євробачення» у 2018 році. Визначте наявні сутності предметної області та принаймні один тернарний зв'язок з атрибутом. Доповніть ER-модель предметної області, створену у Вправі 9.

2.1.4 Зв'язок типу «загальний вид — різновид»

Нехай в межах предметної області «Мережа супермаркетів» нам необхідно зберігати інформацію про всі приміщення, якими володіє наша мережа.

Звісна річ, що до таких приміщень належать як самі супермаркети, так і склади (вважатимемо, що наша мережа не працює зі сторонніми складами, а зберігає всі свої товари у власних складських приміщеннях).

Зазвичай у кожного приміщення є завідувач. Звісно, можна створити два окремі зв'язки: між завідувачем та супермаркетом, а також між завідувачем та складом, — які зберігатимуть відповідну інформацію. Але що в такому разі вдіяти з приміщеннями, які не є ані супермаркетами, ані складами? Наприклад, головний офіс компанії тощо. Створення сутності «Інше приміщення» та її зв'язування з сутністю «Завідувач» призведе до надмірної кількості повторень однакових за змістом атрибутів та зв'язків. Пізніше ми переконаємося в тому, що це значно ускладнює подальшу роботу з даними, зокрема їх видобування та підтримку актуальності.

Виникає питання, як доповнити модель можливістю представлення всіх приміщень таким чином, щоб зберегти її логічність та повноту? Відповідь доволі проста: ввести нову сутність «Приміщення».

Очевидно, що факт «у приміщення є завідувач» подаватиметься відповідним зв'язком між сутностями «Приміщення» та «Завідувач» множинністю «один до багатьох» (адже ми не знаємо напевне, що завідувач керує рівно одним приміщенням).

Проте, ми подали в моделі ще не всі факти, пов'язані зі створеною сутністю «Приміщення». Те, що супермаркет та склад є приміщеннями, можна подати за допомогою двох окремих зв'язків, зображених на Рисунку 2.8.

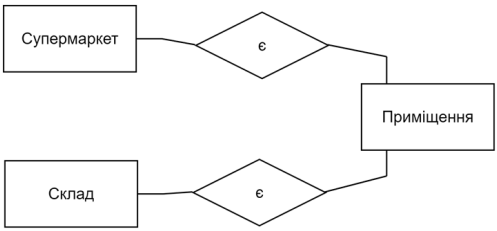


Рис. 2.8: Зв'язки «супермаркет є приміщенням» та «склад є приміщенням»

Зв'язки «супермаркет є приміщенням» та «склад є приміщенням» є прикладами зв'язків типу «загальний вид — різновид», в яких фіксується належність об'єкту більш загальної сутності її підмножині — сукупності об'єктів

більш конкретної сутності.

Вправа 12

Чому вищевказані зв'язки мають саме такі назви, а не, наприклад, «приміщення є супермаркетом» та «приміщення є складом»?

Тепер визначимо множинність розглянутих вище зв'язків. Для зв'язку «приміщення є складом» маємо:

- скількома приміщеннями є один склад? — рівно одним;
- скількома складами є одне приміщення? — не більш, ніж одним.

Відповіді на аналогічні питання для зв'язку «приміщення є супермаркетом» ідентичні. Множинності обох зв'язків — «один до одного», а тому ER-діаграма набуває вигляду, поданого на Рисунку 2.9.

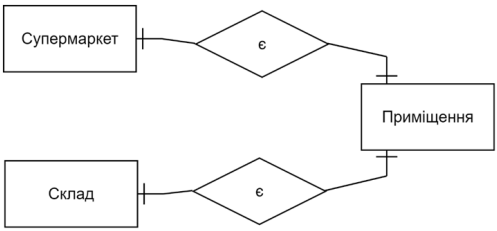


Рис. 2.9: ER-модель зв'язків «супермаркет є приміщенням» та «склад є приміщенням»

Насправді, має місце наступне твердження:

Теорема 1. Множинність будь-якого зв'язку типу «загальний вид — різновид» — «один до одного».

Вправа 13

Доведіть цю теорему, використовуючи запитання для визначення множинності зв'язку.

Вправа 14

II етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас

В базі даних інформаційної системи «Головоломки» зберігається інформація про головоломки. Про кожну головоломку відомі її ідентифікатор та назва. Серед головоломок виділяють:

- друковані, для яких відомі також завдання та відповідь у вигляді окремих файлів та тип (кросворд, ребус, судоку, японський кросворд або доміно-пасьянс);
- усні, для яких відомі також завдання та відповідь в текстовому вигляді;
- механічні, для яких відомі також текстовий опис та зображення головоломки;
- з предметами, для яких відомі також текстовий опис, завдання та відповідь у вигляді окремих файлів та предмет (сірники, монети або гральні карти).

Побудуйте ER-модель цієї предметної області.

Вправа 15

IV етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас

В базі даних зберігаються натуральні числа та їх текстове представлення українською мовою. Серед них виділяють рекурсивно визначені числа та числа, визначені за допомогою арифметичних функцій. Також існують числа, що не належать до жодного з цих класів.

Про рекурсивно визначені числа відомий текстовий опис їх властивостей. Серед рекурсивно визначених чисел виділяють числа Фібоначчі (наприклад: 1, 2, 3, 5, 8, 13) та числа Люка (наприклад: 1, 2, 3, 4, 7, 11). Існують рекурсивно визначені числа, що не є ані числами Фібоначчі, ані числами Люка.

Для чисел Фібоначчі та чисел Люка зберігається інформація про порядковий номер числа у відповідній послідовності.

Про арифметичні функції відома також назва класу формуючої функції.

Побудуйте ER-модель цієї предметної області.

2.1.5 Побудова концептуальної моделі предметної області

Моделювання предметної області за допомогою ER-діаграм дозволяє абстрагуватися під час цього процесу як від конкретної системи управління базами даних, так і від класу баз даних взагалі.

Проектування на даному рівні передбачає виконання наступних етапів:

1. **Визначення меж предметної області, що розглядається.** Перед виділенням концептів предметної області важливо з'ясувати ту частину інформаційного простору, який в майбутньому буде зберігатися в базі даних. Так, предметна область «Мережа супермаркетів» може зберігати інформацію лише про товари, супермаркети та те, які товари де продаються. З іншого боку, ця ж предметна область може фіксувати також і дані про склади, окремі поставки продукції від постачальників, працівників мережі та її відділів тощо. Що більшим є розмір області знань, що розглядається, то більшою буде концептуальна модель відповідної предметної області та більш громіздким буде процес її проектування.
2. **Ідентифікація сутностей.** В рамках окресленої області знань на цьому етапі здійснюється виділення збережуваних об'єктів та їх узагальнення до сутностей — множин однотипних об'єктів. Як ми вже зазначили раніше, сутності зазвичай представлені іменниками в текстовому описі предметної області й уособлюють класи осіб, предметів, подій тощо, дані про які повинні бути збережені у відповідній базі. Один з підходів до ідентифікації сутностей полягає у формуванні переліку параметрів, які повинна зберігати база даних, та поступового об'єднання пов'язаних характеристик в сутності. Проте у предметних областях невеликого масштабу цей процес не потребує такої формалізації і досить часто з текстового опису одразу очевидні класи тих об'єктів, які після проектування бази даних будуть в ній збережені.
3. **Ідентифікація зв'язків між сутностями.** Після формування повного переліку сутностей здійснюється аналіз зв'язків, тобто того, які об'єкти цих сутностей можуть бути одне з одним пов'язані. Як вже згадувалось раніше, в текстовому описі предметної області відношення такого роду зазвичай представляються дієсловами, що значно полегшує процес їх визначення. Наприклад, з речення «Товари постачаються від постачальників.» ми можемо вилучити бінарний зв'язок «постачаються» між сутностями «товар» та «постачальник», а з речення «Товари супермаркету зберігаються на складі.» — тернарний зв'язок «зберігаються» між сутностями «товар», «супермаркет» та «склад».

На цьому етапі важливо коректно визначити зв'язки відповідно до їх ступенів, аби побудована в результаті проектування база даних могла в повній мірі представляти інформацію про предметну область, зокрема замість тернарних зв'язків до моделі не потрапив набір бінарних.

4. **Визначення множинностей зв'язків.**
5. **Ідентифікація атрибутів сутностей та зв'язків.** Деякі початки формування переліку характеристик, що в кінцевому рахунку будуть збережені в проектованій базі даних, можуть здійснюватися ще на етапі ідентифікації сутностей, де згідно з формальним підходом сукупність властивостей розбивається на класи пов'язаних даних — сутності. На цьому етапі належність певних атрибутів певним сутностям фіксується остаточно. Зокрема, визначаються також і ті властивості, які виступають параметрами зв'язків між об'єктами і які важко пов'язати з об'єктами однієї конкретної сутності, — атрибути зв'язків.
6. **Визначення первинного ключа кожної сутності.** Після фіксування переліку атрибутів кожної сутності ми переходимо до процесу визначення того набору характеристик, який визначатиме об'єкт сутності однозначно — її ключа. Відповідно до наведених вище викладок, для цього нам потрібно визначити усі потенційні ключі сутності та обрати один з них в якості первинного. Якщо ж потенційні ключі відсутні, в якості первинного ключа ми обираємо штучний ключ — унікальний ідентифікатор об'єкта відповідної сутності.
7. **Визначення домену кожного з атрибутів.** На цьому етапі концептуального проектування для кожної властивості як сутностей, так і зв'язків визначається її домен — множина значень, які ця характеристика може приймати. Наприклад, атрибут «стать» сутності «працівник» може набувати значення з множини $\{male, female\}$. Слід зазначити, що на цьому рівні проектування домени фіксуються досить умовно, в деяких випадках — неформально, описово, а конкретизуються вже далі, на логічному чи навіть фізичному рівні. Наприклад, для атрибуту «прізвище» тієї ж сутності «працівник» домен може бути визначений як текстовий, хоча формально множина текстів є нескінченною і математично її описати доволі складно.
8. **Побудова моделі «сутність-зв'язок».** Цей етап передбачає формування вихідного документу концептуального рівня проектування — ER-діаграми, яка фіксує дані про предметну область, отримані на зазначених вище етапах. Отримана в результаті застосування розглянутих в цьому підрозділі технік ER-модель є відправною точкою для проектування на наступному рівні.

На деяких з вищезазначених етапів може з'ясуватися, що перелік сутно-

стей та/або зв'язків між ними було визначено не в повній мірі, або некоретно. В такому разі слід повернутися на відповідний етап проектування, виправити помилку та повторити слідуючі за ним етапи відповідно до нових вихідних даних.

Приклад 1

І етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас

В інформаційній системі «Енергетика» користувачам надається інформація про українські атомні та гідроелектростанції. Про кожну електростанцію відомі її ідентифікатор, назва, розташування та потужність. Про кожну атомну електростанцію відомо те, діюча вона чи неактивна. Про гідроелектростанції відомий їх тип (руслова, пригреблева, деривaційна, гідроакумулювальна або припливна). Атомні електростанції складаються з енергоблоків, про які відомі номер в межах станції, тип реактора та потужність. Гідроелектростанції можуть розташовуватися на одній чи кількох річках, про які відома їх довжина. Побудуйте концептуальну модель відповідної бази даних та зобразіть її на ER-діаграмі.

Оскільки межі предметної області цілком описано в її текстовому поданні, перший етап концептуального проектування можна опустити. З опису предметної області, поданого вище, можна легко визначити сутності, що її складають. Загалом в описі подано чотири типи об'єктів: атомні електростанції, гідроелектростанції, річки та енергоблоки. Слід зауважити, що дві перші категорії не представляють одну єдину сутність, адже за другим визначенням сутності (див. Означення 14), це множина об'єктів, що мають однаковий набір атрибутів та однотипні зв'язки з іншими об'єктами. Не всі електростанції складаються з енергоблоків і не всі електростанції розташовані на річках, а отже атомні та гідроелектростанції мають різнотипні зв'язки та різні атрибути, звідки є різними сутностями в предметній області, що розглядається. Отже, маємо чотири сутності: «атомна електростанція», «гідроелектростанція», «річка» та «енергоблок».

З опису між заданими сутностями можна виділити два зв'язки: «енергоблок належить до атомної електростанції» та «гідроелектростанція розташована на річці». Визначимо їх множинності за поданим раніше алгоритмом.

Розглянемо зв'язок «енергоблок належить до атомної електростанції». Поставимо два питання та дамо на них відповіді:

- один енергоблок до скількох атомних електростанцій може належати? — не більш, ніж до однієї;
- одна атомна електростанція скільки енергоблоків може містити? — один або більше, тобто багато.

Отже, зв'язок має множинність «один до багатьох» (Рисунок 2.10).

Рис. 2.10: Зв'язок «енергоблок належить до атомної електростанції»

Розглянемо зв'язок «гідроелектростанція розташована на річці». Поставимо два питання та дамо на них відповіді:

- одна гідроелектростанція на скількох річках може розташовуватися? — на одній або більше, тобто на багатьох;
- одна річка може бути місцем розташування скількох гідроелектростанцій? — однієї або більше, тобто багатьох.

Отже, зв'язок має множинність «багато до багатьох» (Рисунок 2.11).

Рис. 2.11: Зв'язок гідроелектростанція розташована на річці

В предметній області «Енергетика», яку ми наразі проектуємо, атрибути сутностей та їх домени (в дужках) однозначно визначаються з її текстового опису:

- для сутності «атомна електростанція»: «ідентифікатор» (число), «назва» (текст), «розташування» (текст), «потужність» (число) та «діючість» ($\{yes, no\}$);
- для сутності «гідроелектростанція»: «ідентифікатор» (число), «назва» (текст), «розташування» (текст), «потужність» (число) та «тип» (текст);
- для сутності «енергоблок»: «номер» (число), «тип реактора» (текст) та «потужність» (число);

- для сутності «річка»: неявний атрибут «назва» (текст), що випливає з контексту текстового опису, та «довжина» (число).

У вище визначених зв'язків атрибути відсутні.
Відповідно до визначення первинного ключа можна легко визначити ключі сутностей:

- для сутності «атомна електростанція»: «ідентифікатор»;
- для сутності «гідроелектростанція»: «ідентифікатор»;
- для сутності «енергоблок»: «ідентифікатор атомної електростанції», «номер»;
- для сутності «річка»: «назва».

Зверніть увагу, що для сутності «енергоблок» її власних атрибутів для визначення ключа було недостатньо, тому для побудови первинного ключа було запозичено ключ пов'язаної сутності «атомна електростанція».
На основі отриманих даних побудуємо ER-діаграму предметної області (Рисунок 2.12).

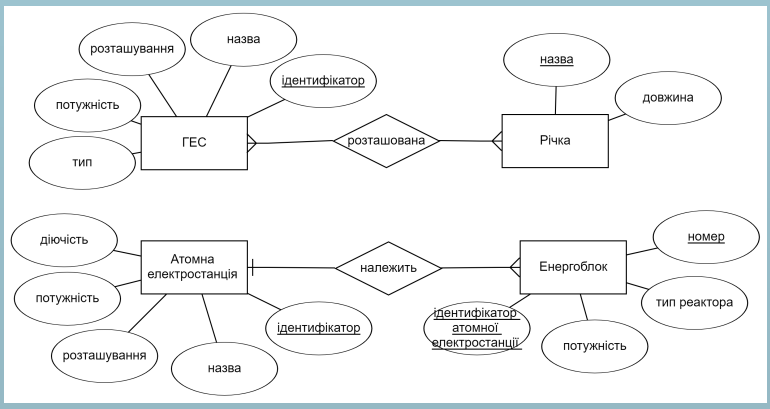


Рис. 2.12: ER-модель інформаційної системи «Енергетика»

Вище в предметній області «Енергетика» ми виділили сутності «атомна електростанція», «гідроелектростанція», «річка» та «енергоблок». Переліки атрибутів сутностей «атомна електростанція» та «гідроелектростанція» доволі схожі, адже більшість характеристик має будь-яка електростанція, а не тільки конкретні розглянуті. Ґрунтована на цьому переліку сутностей схема є цілком коректною, якщо ми будуємо інформаційну систему лише для атомних та гідроелектростанцій.

Але якщо до цих електростанцій додати решту їх категорій, схема в повній мірі предметну область вже не представлятиме.
Тому поставимо перед собою завдання побудувати модель, яка дозволятиме без втрати змісту попередньої схеми доповнити її так, щоб вона могла представляти усі електростанції, а зв'язки забезпечувати лише для двох їх типів, які ми розглядали раніше.
Один із варіантів рішення даної проблеми — виділити сутність «інша електростанція», яка представлятиме решту електростанцій. Він дозволить вирішити цю проблему в повній мірі, проте не дозволить надалі розширяти ER-модель, оскільки створити зв'язок для всіх електростанцій одночасно з певною іншою сутністю буде доволі складно.
Більш коректним і з точки зору розширюваності, і з точки зору логіки є шлях виділення окремої сутності «електростанція», яка представлятиме станції усіх типів. Тоді сутності «атомна електростанція», «гідроелектростанція» представлятимуть різновиди основної сутності.
Таким чином між сутностями електростанція та атомна електростанція, а також електростанція та гідроелектростанція слід побудувати зв'язки «є»: «атомна електростанція є електростанцією», «гідроелектростанція є електростанцією». Тоді сутність «електростанція» вбере в себе спільні для всіх електростанцій атрибути («ідентифікатор» (число), «назва» (текст), «розташування» (текст)та «потужність» (число)), а в сутностях «атомна електростанція», «гідроелектростанція» залишаться тільки ті атрибути, які для них є унікальними («діючість» ($\{yes, no\}$) та «тип» (текст) відповідно), а також ключ сутності-виду — «ідентифікатор».

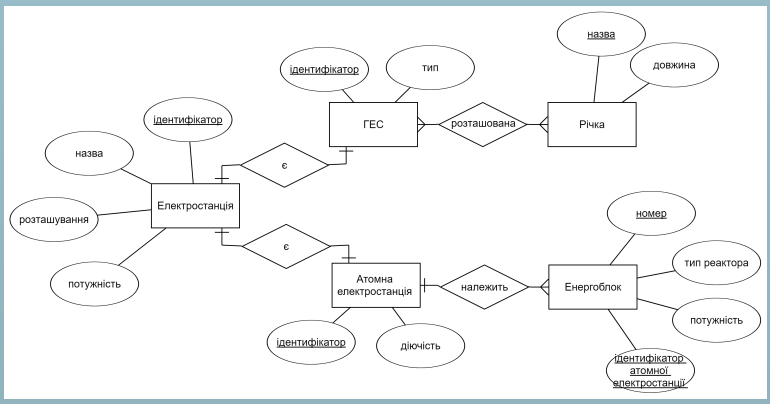


Рис. 2.13: ER-модель інформаційної системи «Енергетика»

Множинність довільного зв'язку «загальний вид-різновид» — «один до одного», тому ER-модель набуває вигляду, поданого на Рисунку 2.13.
Зверніть увагу, що на певному етапі ми визначили, що обраний нами перелік сутностей є неповним. З цього моменту процес проектування повертається на етап *2. Ідентифікація сутностей.* Тому слід також відповідним чином модифікувати перелік сутностей та зв'язків предметної області.

Вправа 16

Чи буде коректною для предметної області «Енергетика» з прикладу ER-діаграма, подана на Рисунку 2.14, якщо не потрібно буде зберігати інформацію про тип ГЕС, а діючість стане атрибутом усіх електростанцій? Чому?

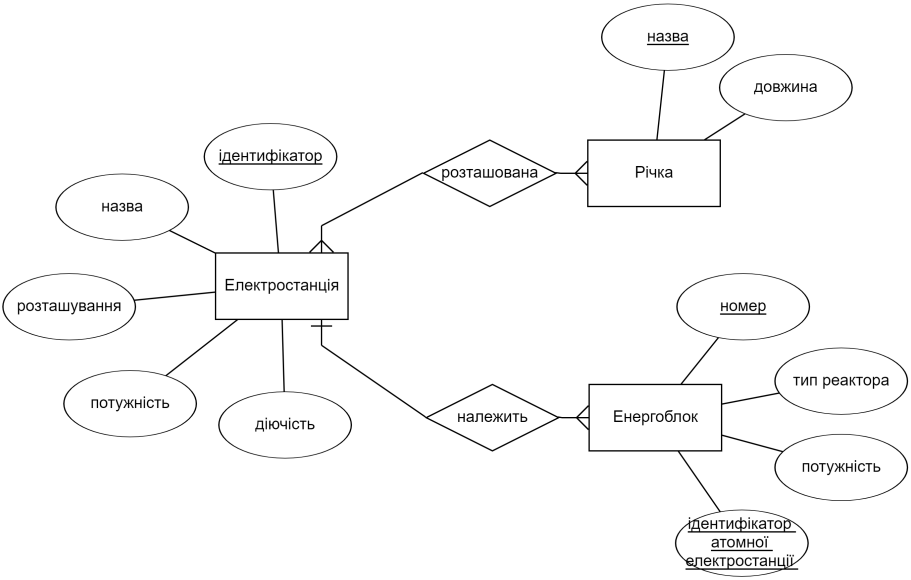


Рис. 2.14: ER-модель до Вправи 16

Вправа 17

І Всеукраїнська учнівська Інтернет-олімпіада з інформаційних технологій, тренувальний тур І етапу

Інформаційна система «Школа» зберігає інформацію про навчальний процес в окремому закладі освіти, а саме про класи (назва та класна кімната), учнів (номер особової справи, прізвище, ім'я, по батькові), предмети (назва та галузь знань), вчителів (номер паспорта, прізвище, ім'я, по батькові, категорія) та класних керівників серед вчителів. Для класних керівників додатково зберігається їх особистий робочий кабінет. База даних інформаційної системи повинна зберігати дані про те, який учень в якому класі навчається, а також про розклад класів: який вчитель в якому класі на якому уроці в який день який предмет в якому кабінеті викладає. Також потрібно зберігати який класний керівник яким класом керує.

1. Визначте сутності предметної області, зв'язки між ними та їх множинності. Визначте атрибути обраних сутностей та їх ключі.
2. Зобразіть загальну ER-модель предметної області.
3. Вкажіть на діаграмі атрибути для тих доданих зв'язків, для яких вони задані.

Вправа 18

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2016», 10-11 клас

В інформаційній системі «Файлова система» зберігається інформація про каталоги та файли Unix-подібної операційної системи.

Кожен каталог характеризується ідентифікатором (унікальним номером каталогу) та назвою (текст). В каталозі можуть міститися інші каталоги, а також файли. Каталог зокрема може бути і порожнім.

Вважається, що в системі завжди існує єдиний кореневий каталог з назвою root. Усі інші каталоги та файли містяться в ньому безпосередньо або вкладені в інші каталоги, що в ньому містяться.

Кожен файл характеризується назвою. Кожен об'єкт повинен міститися в рівно одному каталозі (в тому числі, можливо, в кореневому).

Побудуйте ER-модель цієї предметної області. Врахуйте, що в одному каталозі не може бути каталогів з одним іменем, так само як однаково названих файлів. Проте, наприклад каталог та файл itolymp можуть міститися в одному каталозі одночасно.

Вправа 19

IV етап I Всеукраїнської учнівської олімпіади з інформаційних технологій

Побудуйте ER-модель предметної області «Водні ресурси», опис якої подано нижче.

Кожна ріка характеризується назвою, довжиною і площею басейну, а море — площею водної поверхні. Потрібно зберігати відомості про те, територією яких держав протікає ріка, а також у яку водойму (море чи іншу ріку) вона впадає. Штучні моря (водосховища) також характеризуються площею водної поверхні і в них також можуть впадати ріки, однак кожне водосховище, на відміну від природного моря, розташоване на певній ріці.

Рис. 2.15: До Вправи 20

Вправа 20

IV етап III Всеукраїнської учнівської олімпіади з інформаційних технологій

Побудуйте ER-модель предметної області «Географічна карта», опис якої подано нижче.

Карта поділена на області, які можуть бути державами та морями. Як держава, так і море має площу, але держава характеризується ще чисельністю населення. Потрібно зберігати відомості про те, які держави та моря з якими державами та морями межують. Крім того, є міста, що характеризуються чисельністю населення та належать певним державам.

Вправа 21

IV етап IV Всеукраїнської учнівської олімпіади з інформаційних технологій

Побудуйте ER-модель бази даних для зберігання відомостей про лабіринт у квадраті 10x10. Лабіринт є набором тунелів, спрямованих вертикально чи горизонтально. Потрібно заборонити можливість зберігання даних про тунелі, спрямовані по діагоналі.

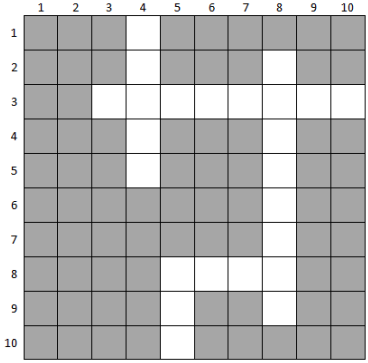


Рис. 2.16: До Вправи 21

2.2 Логічне проектування бази даних

В рамках даного підрозділу ми познайомимося з наступним рівнем проектування баз даних — логічним. На цьому етапі проектування створюється модель, чутлива до моделі організації даних. Надалі ми акцентуватимемо увагу на базах даних з реляційною моделлю даних, що є предметом вивчення даної книги.

2.2.1 Математичні основи реляційного підходу до організації баз даних

Реляційний підхід до організації баз даних ґрунтується на математичному апараті теорії множин та відношень.

Поняття множини відносять до аксіоматичних понять, які не мають точного визначення. В рамках цієї книги для розгляду математичних основ теорії реляційних баз даних нам буде достатньо наступного інтуїтивного визначення цього поняття.

Означення 18. *Множина — невпорядкована сукупність об'єктів.*

Сучасна теорія множин будується на системі аксіом, з яких виводяться всі теореми і твердження теорії множин. Оскільки теорія реляційних баз даних пов'язана лише зі скінченими множинами, виклад наступного матеріалу не залежить від того, якої із цих систем дотримуватися, тому ми опустимо розгляд аксіом в рамках даного посібника та розглянемо основні властивості в дещо спрощеній формі.

Означення 19. *Елементами множини називаються її об'єкти.*

Твердження «x є елементом множини A» записують як $x \in A$. Якщо x не є елементом множини A, пишуть $x \notin A$.

До прикладів множин можна віднести:

- сукупність з п'яти літер латиниці: $A = \{a, b, c, d, e\}$;
- сукупність плодів: $C = \{\text{яблуко}, \text{груша}, \text{виноград}, \text{апельсин}\}$;
- множина натуральних чисел N;
- множина цілих чисел Z;
- множина дійсних чисел R.

Означення 20. *Множина, що не містить жодного елементу, називається порожньою і позначається* $\varnothing$.

Традиційно множини задають одним із трьох способів:

1. **За допомогою текстового опису.** Наприклад, множина M складається з місяців, що починаються на літеру Л.
2. **За допомогою нотації Ростера.** Та ж множина M у цій нотації матиме вигляд $M = \{\text{лютий}, \text{липень}, \text{листопад}\}$. У фігурних дужках тут вказуються через кому всі елементи відповідної множини.
3. **За допомогою нотації побудови множини.** Та ж множина M у цій нотації матиме вигляд $M = \{x | x$ — місяць і x починається з літери Л$\}$. У фігурних дужках тут до вертикальної риски вказується вигляд об'єктів множини, а після неї — умова, у разі виконання якої відповідний об'єкт належить множині.

Означення 21. *Множина A є підмножиною множини B (позначається як $A \subseteq B$), якщо кожен елемент множини A є також елементом множини B.*

Визначимо операції над множинами, які нам стануть в нагоді під час розгляду реляційної моделі.

Означення 22. *Перетином множин A та B називається множина $A \cap B$, що складається з усіх об'єктів, що є елементами множин A і B одночасно.*

Означення 23. *Об'єднанням множин A та B називається множина $A \cup B$, що складається з усіх об'єктів, що є елементами або множини A, або множини B, або обох множин одночасно.*

Означення 24. *Різницею множин A та B називається множина $A \setminus B$, що складається з усіх об'єктів, що є елементами множини A, але не є елементами множини B.*

Вправа 22

1. Для заданих множин A та B обчисліть їх перетин, об'єднання та різницю:
   - $A = \{5, 9\}, B = \{1, 2, 9, 12\}$;
   - $A = \{\frac{p}{q} | p \in N, q \in N\}, B = Z$;

- $A = \{x^2 | x - \text{парне}\}, B = \{x | x - \text{парне}\}$;
- $A = \{6x | x \in N\}, B = \{6x + 3 | x \in N\}$.

2. Використовуючи визначення підмножини та відповідних операцій доведіть тотожності:

- $A \subseteq A \cup B$;
- $A \cap B \subseteq A$;
- якщо $A \subseteq B$, то $A \setminus B = \varnothing$.

Означення 25. *Потужністю $|A|$ скінченної множини A називається кількість її елементів.*

Наприклад:

- $|\{a, b, c, d, e\}| = 5$;
- $|\varnothing| = 0$;
- $|\{\varnothing\}| = 1$.

Вправа 23

1. Серед математиків кожен сьомий — філософ, а серед філософів кожен дев'ятий – математик. Кого більше і в скільки разів: філософів чи математиків?
2. В саду Івана та Ганни росли 2018 червоних кущів. Кожен з них полив по половині всіх кущів. При цьому виявилось, що рівно три кущі — найкрасивіші — були политі як Іваном, так і Ганною. Скільки кущів залишились неполитими?
3. Лісник рахував сосни в лісі. Він обійшов 5 кіл, як показано на Рисунку 2.17, і всередині кожного кола він нарахував рівно по 3 сосни. Чи можливо, що лісник жодного разу не помилився?

Означення 26. *Впорядкованою парою елементів x та y називається множина $(x, y) = \{x, \{x, y\}\}$.*

Простими словами впорядкована пара — така пара елементів x та y, для якої, на відміну від двоелементної множини, задається черговість (порядок) цих елементів, тобто в загальному випадку $(x, y) \neq (y, x)$.

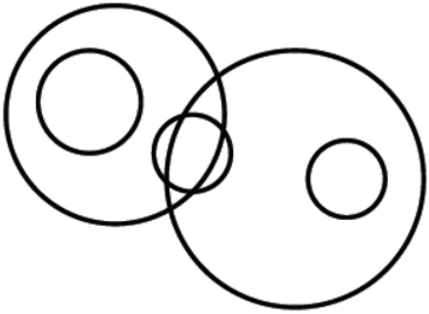


Рис. 2.17: До Вправи 23

Означення 27. *Декартовим добутком множин A та B називається множина $A \times B$, що складається з усіх можливих упорядкованих пар об'єктів, в яких перший об'єкт є елементом множини A, а другий — множини B.*

$$A \times B = \{(x, y) | x \in A, y \in B\}$$

Простими словами, декартів добуток множин — це множина усіх можливих комбінацій елементів з цих множин.

Нехай $A = \{1, 2, 3\}, B = \{a, b, c, d\}$. Тоді декартовий добуток цих множин матиме вигляд:

$$\begin{aligned} A \times B = \{&(1, a), (1, b), (1, c), (1, d), \\ &(2, a), (2, b), (2, c), (2, d), \\ &(3, a), (3, b), (3, c), 3, d)\} \end{aligned} \quad (2.1)$$

Теорема 2. Якщо потужності скінченних множин A та B — m та n відповідно, то потужність множини $A \times B$ — $m * n$.

Вправа 24

Доведіть вищенаведене твердження.

Вправа 25

1. Для заданих множин A та B обчисліть їх декартів добуток:
   - $A = \{1, 2\}, B = \{1, 3\}$;
   - $A = \{a, b\}, B = \varnothing$;
   - $A = \{\text{яблуко}, \text{груша}, \text{кокос}\}$, $B = \{\text{яблуня}, \text{груша}, \text{пальма}\}$;
   - $A = R, B = R$.
2. За допомогою операції декартового добутку випишіть усі можливі двоцифрові числа, що складаються тільки з цифр 1, 2, 3.

Означення 28. *Бінарним відношенням R на множинах A та B називається довільна підмножина їх декартового добутку $A \times B$.*

$$R \subseteq A \times B$$

Розглянемо декартовий добуток множин $A = \{\text{яблуко}, \text{груша}, \text{кокос}\}$, $B = \{\text{яблуня}, \text{груша}, \text{пальма}\}$:

$$\begin{aligned} A \times B = \{&(\text{яблуко, яблуня}), (\text{яблуко, груша}), (\text{яблуко, пальма}),\\ &(\text{груша, яблуня}), (\text{груша, груша}), (\text{груша, пальма}),\\ &(\text{кокос, яблуня}), (\text{кокос, груша}), (\text{кокос, пальма})\} \end{aligned} \quad (2.2)$$

За означенням 28, будь-яка підмножина отриманої нами множини є відношенням. В тому числі відношенням є і наступна множина:

$$R_{\text{плід на дереві}} = \{(\text{яблуко, яблуня}), (\text{груша, груша}), (\text{кокос, пальма})\} \quad (2.3)$$

Це відношення за своїм сенсом означає «плід росте на дереві».

За аналогією, відношення «літера належить слову» між множинами $\{a, b, c\}$та$\{aa, bb, bc, abc, aabbbbc\}$ матиме вигляд:

$$\begin{aligned} R_{\text{літера в слові}} = \{&(a, aa), (a, abc), (a, aabbbbc),\\ &(b, bb), (b, bc), (b, abc), (b, aabbbbc),\\ &(c, bc), (c, abc), (c, aabbbbc)\} \end{aligned} \quad (2.4)$$

Вправа 26

1. Побудуйте відношення «друге число є квадратом першого» між множинами $\{1, 2, 3\}$ та N.
2. Побудуйте відношення «перше та друге числа рівні між собою» між множинами Z та N.
3. Побудуйте відношення «перше та друге числа мають однакову парність» між множинами $\{1, 2, 3\}$ та N.

Поняття впорядкованої пари можна розширити. Так, впорядкованою трійкою елементів x, y та z можна вважати пару вигляду $(x, y, z) = (x, (y, z))$, а впорядкованою n-кою елементів $a_1, a_2, ..., a_n$, з тих самих міркувань, — пару $(a_1, a_2, ..., a_n) = (a_1, (a_2, ..., (a_{n-1}, a_n)...))$.

В такому разі, природньо, що декартовий добуток n множин $A_1 \times A_2 \times \cdots \times A_n$ — множина усіх можливих n-ок $(a_1, a_2, ..., a_n)$, де $a_1 \in A_1, a_2 \in A_2, ..., a_n \in A_n$. Подібним чином поняття відношення можна розширити з упорядкованих пар на упорядковані n-ки.

Означення 29. *n-арним відношенням R на множинах $A_1, A_2, \ldots, A_n$ називається довільна підмножина їх декартового добутку $A_1 \times A_2 \times \cdots \times A_n$.*

$$R \subseteq A_1 \times A_2 \times \cdots \times A_n$$

Розглянемо три множини: множину товарів T = {хліб, молоко, ковбаса}, множину складів S = {Склад №1, Склад №2} та множину супермаркетів M = {Відділення №1, Відділення №2}. Тоді тернарне відношення на цих множинах «товар привезено зі складу в супермаркет» матиме вигляд:

$$R_{\text{товар привезено зі складу в суперм.}} = \{(\text{хліб, Склад №1, Відділення №2}), (\text{хліб, Склад №2, Відділення №1}), (\text{молоко, Склад №2, Відділення №1}), (\text{ковбаса, Склад №2, Відділення №1})\} \quad (2.5)$$

Вищерозглянуте тернарне відношення відповідає тернарному зв'язку «товар привезено зі складу в супермаркет», який ми розглянули раніше впродовж концептуального моделювання. Тут елементами множин, на яких побудоване відношення, виступають цілі об'єкти.

З іншого боку, за допомогою відношень ми маємо змогу описувати й самі об'єкти. Розглянемо множину прізвищ F = {Іванов, Петренко, Сидоров}, імен L = {Андрій, Василь} та натуральних чисел N. Тоді відношення R = {(Петренко, Андрій, 18), (Сидоров, Василь, 17), (Петренко, Василь, 17)}

представляє множину учнів шкільного класу, про кожного з яких відомі їх прізвище, ім'я та вік. Таке відношення легко подати у вигляді Таблиці 2.1.

Петренко	Андрій	18
Сидоров	Василь	17
Петренко	Василь	17

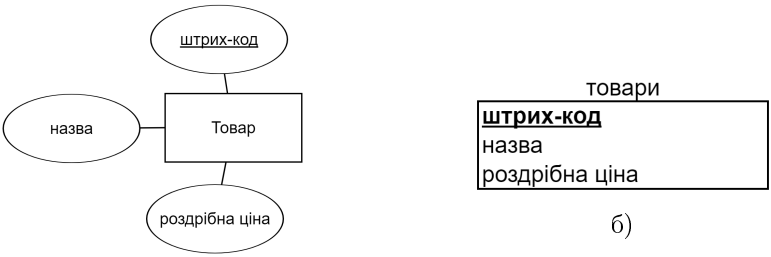
Табл. 2.1: Тернарне відношення «учень»

Далі n-арні відношення ми називатимемо просто — таблицями.

2.2.2 Реляційна модель

Продуктом логічного моделювання, так само як і концептуального, є модель бази даних, а вхідними даними цього процесу — модель «сутність-зв'язок». Вкотре нагадаємо, що на цьому етапі ми отримуємо модель, чутливу до форми організації даних. В нашому випадку — реляційну (табличну) модель, побудовану на основі таблиць та зв'язків між ними.

Кожна сутність концептуальної моделі відтворюється в реляційній моделі у вигляді окремої таблиці, стовпці якої відповідають її атрибутам.



а) б)

Рис. 2.18: (а) ER-діаграма та (б) реляційна модель сутності «товар»

Наприклад, реляційну модель сутності «товар» з атрибутами «штрих-код», «назва» та «роздрібна ціна», ER-модель якої подана на Рисунку 2.18а, можна зобразити у вигляді, зображеному на Рисунку 2.18б.

Означення 30. *Стовпці таблиці називаються полями, рядки таблиці — записами.*

Таблицю на реляційній моделі подають у вигляді прямокутника, згори зовні якого розташовують назву цієї таблиці, а всередині — перелік її полів (стовпців).

Означення 31. *Ключ таблиці — мінімальний набір її полів, значення якого однозначно визначає запис серед усіх записів цієї таблиці.*

Ключ таблиці, що представляє сутність предметної області, відповідає ключу цієї сутності в концептуальній моделі. Поля таблиці, що належать до її ключа, на реляційній моделі підкреслюються або виділяються іншим чином.

Загальний вигляд реляційної моделі зв'язку поданий на Рисунку 2.19. Як ви можете помітити, зв'язок тут позначається лінією, що з'єднує не просто таблиці, а конкретні їх поля. Чому саме так, ви дізнаєтеся під час детального розгляду побудови реляційної моделі.

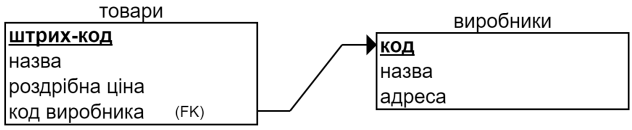


Рис. 2.19: Реляційна модель зв'язку «виробник виготовляє товар»

Якщо відтворення сутностей концептуальної моделі проводиться легко, то відтворення на реляційній моделі зв'язків між сутностями — процес більш складний і нелінійний. В залежності від множинності та ступеня зв'язку шляхи перетворення ER-моделі на реляційну відрізняються. Розглянемо їх більш детально.

2.2.3 Побудова реляційної моделі бінарних зв'язків

Раніше ми успішно побудували концептуальні моделі для трьох зв'язків — по одному для кожної з множинностей можливих для бінарних зв'язків. Далі ми зосередимось на тому, як отримані ER-діаграми перетворити на реляційні.

Зв'язки «один до багатьох»

Розглянемо зв'язок «виробник виготовляє товар» множинністю «один до багатьох», ER-діаграму якого подано на Рисунку 2.20.

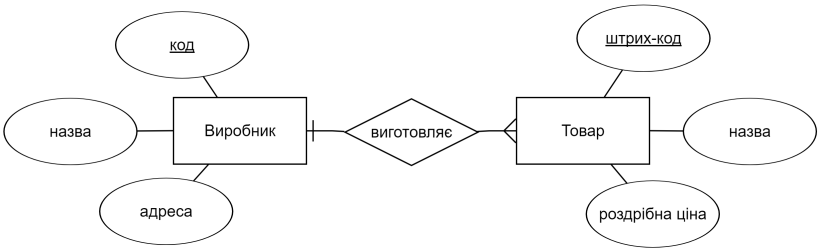


Рис. 2.20: ER-модель зв'язку «виробник виготовляє товар»

Як вже було сказано раніше, кожна сутність ER-моделі представляється в реляційній моделі як окрема таблиця. Отже, до реляційної моделі зв'язку потраплять щонайменше дві таблиці: таблиця «виробники» та таблиця «товари».

Уявімо, що ці таблиці подані в електронному вигляді в деякому текстовому чи табличному процесорі (дивіться Рисунок 2.21). Яким чином ми можемо вказати для певного товару який саме виробник його виготовив? Оскільки певний виробник однозначно визначається його кодом (код — ключ сутності «Виробник»), то насправді достатньо в таблицю «товари» внести додаткове поле «код виробника», в якому зазначити код того виробника, який виготовив товар. Після цього товар та виробник вважаються зв'язаними, адже за допомогою нескладних алгоритмічних маніпуляцій можна легко визначити, наприклад, назву виробника того чи іншого товару.

товари

штрих-код	назва	роздрібна ціна	код виробника
8215562868	Хліб білий	12,80 ₴	1
9896514456	Хліб житній	10,30 ₴	1
6546468784	Молоко українське	28,30 ₴	2
6546872698	Сир голландський	231,10 ₴	2
8635434366	Сирок плавлений	13,20 ₴	2
2455585436	Батон білий	14,30 ₴	3

виробники

код	назва	адреса
1	Київхліб	м. Київ, вул. Межигірська, 83
2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 2.21: Приклад табличного подання зв'язку «виробник виготовляє товар»

Вправа 27

1. Запропонуйте словесний опис алгоритму, за яким для кожного товару можна визначити його назву та назву його виробника, використовуючи вищенаведену модель організації даних.
2. Запрограмуйте цей алгоритм довільною мовою програмування, подаючи відповідні таблиці у вигляді текстових файлів, або ж реалізуйте його за допомогою формул в середовищі текстового процесора, подаючи відповідні таблиці на різних його аркушах.

Відповідну реляційну модель зв'язку «виробник виготовляє товар» подано на Рисунку 2.22. Кажуть, що товари пов'язані з виробниками за допомогою полів «код виробника» та «код» цих таблиць. Ці поля на схемі з'єднують лінією.

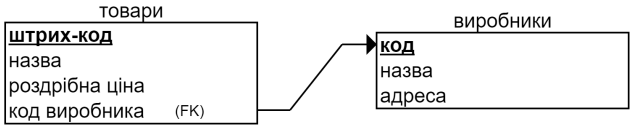


Рис. 2.22: Реляційна модель зв'язку «виробник виготовляє товар»

Узагальнимо вищесказане на усі бінарні зв'язки множинністю «один до багатьох». Отже, *для відтворення зв'язку множинністю «один до багатьох» на реляційній моделі слід до полів таблиці, якій відповідає позначка «багато» («кроляча лапка» на ER-моделі), додати ключ таблиці, якій відповідає позначка «один». Кажуть, що таблиці пов'язані між собою за полями ключа і відповідного йому зовнішнього ключа. Відповідні поля пов'язаних таблиць на схемі з'єднують лінією.* В такому випадку, за допомогою збереженого зовнішнього ключа легко визначити значення всіх полів пов'язаного запису іншої таблиці.

Означення 32. *Зовнішній ключ таблиці — це сукупність її полів, які складають ключ іншої, пов'язаної з нею, таблиці.*

Нагадаємо, що раніше ми вже зустрічалися з поняттям зовнішнього ключа, коли розглядали концептуальне моделювання. Тоді ми запозичували ключ іншої сутності, коли не вистачало атрибутів для формування ключа. Звертаємо увагу на те, що на реляційних моделях, побудованих за допомогою ERDplus, поля, що входять до складу зовнішнього ключа, позначаються

поміткою *(FK)* — від англійського словосполучення foreign key — зовнішній ключ. Стрілка на лініях зв'язків вказує від зовнішнього ключа до основного.

Основні та зв'язані таблиці

Реалізація зв'язків «один до одного» багато в чому залежить від того, яку з двох таблиць обрано зв'язаною, а яку — основною. Далі буде подано кілька неформальних рекомендацій, які значно спрощують перехід від концептуальної до реляційної моделі.

Означення 33. *Зв'язаною у реляційному зв'язку двох таблиць обирається та таблиця, кожен запис якої бере участь в цьому зв'язку не більш, ніж один раз. Якщо таких таблиць декілька, то зв'язаною обирається та таблиця, в якій більша частка записів бере участь у зв'язку. Якщо ця частка є однаковою або неможливо її оцінити, то довільна з двох таблиць визначається як зв'язана.*

Наприклад, у зв'язку «працівник керує складом» кожен склад обов'язково бере участь, адже кожен склад має керівника, причому рівно одного. Тому таблиця «склади» тут є зв'язаною. З іншого боку, не обов'язково кожен працівник повинен керувати складом, а тому таблиця «працівники» зв'язаною не є.

У зв'язку «виробник виготовляє товар» кожен товар виготовляється певним одним виробником, проте виробник може виготовляти більш, ніж один товар, або ж не виготовляти їх взагалі. Тому таблиця «товари» тут є зв'язаною, а таблиця «виробники» — ні.

Означення 34. *Основною у реляційному зв'язку двох таблиць обирається таблиця, що не є зв'язаною.*

Насправді, має місце наступне твердження:

Теорема 3. У зв'язку множинністю «один до багатьох» таблиця, якій відповідає помітка «багато», завжди є зв'язаною, а таблиця, якій відповідає помітка «один» — основною.

Вправа 28

Доведіть вищенаведене твердження.

Вправа 29

Визначте основну та зв'язану таблиці для наступних зв'язків:

- чоловік одружений з жінкою;
- планета є небесним тілом;
- мати народила дитину;
- вчитель є класним керівником певного класу;
- місто належить до країни.

Теорема 4. У зв'язку типу «загальний вид — різновид» таблиця, що відповідає за загальний вид, завжди є основною, а таблиця, що відповідає за різновид — зв'язаною.

Вправа 30

Доведіть вищенаведене твердження.

Вправа 31

Чи можуть у бінарному зв'язку обидві таблиці бути основними? Поясніть свою думку.

Зв'язки «один до одного»

Освіжимо в пам'яті зв'язок «працівник керує складом» множинністю «один до одного», ER-діаграму якого подано на Рисунку 2.23.

Знов-таки, реляційна модель цього зв'язку міститиме принаймні дві таблиці: таблицю «працівники» та таблицю «склади», — кожна з яких представляє сутності, залучені в цьому зв'язку.

Очевидно, що відтворити зв'язок між цими таблицями на реляційній моделі можна аналогічно тому, як це було зроблено раніше для зв'язків множинністю «один до багатьох»: за допомогою додавання зовнішнього ключа до таблиці. Проте в даному випадку не є таким очевидним, до якої з таблиць такий набір полів слід додавати, а тому ґрунтовно розглянемо кожен з двох можливих варіантів.

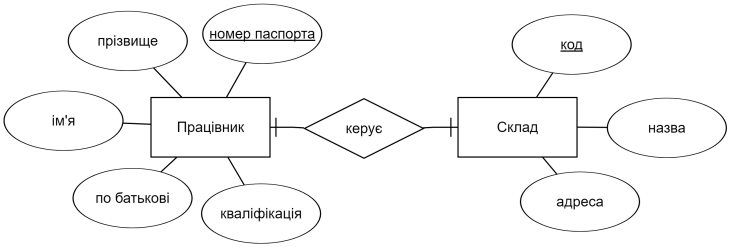


Рис. 2.23: ER-модель зв'язку «працівник керує складом»

Нехай ми додаємо ключ таблиці «склади» (а він складається з єдиного поля «код») до таблиці «працівники» і реляційна схема досягає вигляду, поданого на Рисунку 2.24.

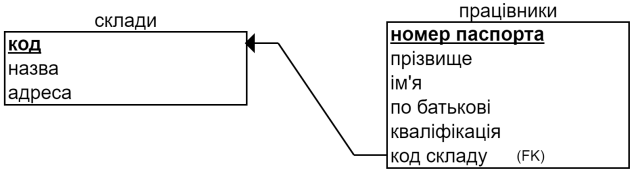


Рис. 2.24: Реляційна модель зв'язку «працівник керує складом» (варіант 1)

працівники

номер паспорта	прізвище	ім'я	по батькові	кваліфікація	код склад
АА123456	Іванов	Андрій	Сергійович	вища	1
АН564646	Андрієнко	Роман	Григорович	вища	2
ОН654643	Петренко	Віктор	Борисович	вища	2
НН564127	Бобров	Семен	Іванович	1 кат.	3
СР486823	Титаренко	Віктор	Леонідович	3 кат.	3
ВР556465	Боголюбов	Борис	Арсенович	вища	3

склади

код	назва	адреса
1	Васильківський	м. Київ, вул. Васильківська, 102
2	Голосіївський	м. Київ, вул. Ломоносова, 53
3	Ужгородський	м. Ужгород, вул. Володимирська, 92

Рис. 2.25: Приклад табличного подання зв'язку «працівник керує складом» (варіант 1)

Якщо уважно поглянути на реляційну модель та на приклад табличного подання цього зв'язку (дивіться Рисунок 2.25), можна помітити, що в такому випадку для кожного працівника відомий код складу, яким він керує, а отже кожен працівник керує якимось складом. Чи відповідає це дійсності? Ні, оскільки велика кількість працівників, як-от касири, адміністратори торгових залів, продавці, не керують жодним складом. З іншого боку, ця схема

зовсім не гарантує, що кожен склад має свого керівника. Отже, ми дійшли висновку, що такий варіант моделі не є коректним.

Вправа 32

Чому, на вашу думку, не слід розглядати як коректний варіант, коли для працівників, що не є керівниками складів, поле «код складу» просто залишається порожнім? У відповіді на це питання вам може допомогти досвід програмування.

Якщо ж додати ключ таблиці «працівники» (тобто «номер паспорта») до таблиці «склади», реляційна модель набуде вигляду, поданого на Рисунку 2.26.

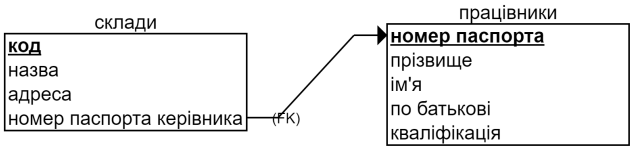


Рис. 2.26: Реляційна модель зв'язку «працівник керує складом» (варіант 2)

працівники

номер паспорта	прізвище	ім'я	по батькові	кваліфікація
АА123456	Іванов	Андрій	Сергійович	вища
АН564646	Андрієнко	Роман	Григорович	вища
ОН654643	Петренко	Віктор	Борисович	вища
НН564127	Бобров	Семен	Іванович	1 кат.
СР486823	Титаренко	Віктор	Леонідович	3 кат.
ВР556465	Боголюбов	Борис	Арсенович	вища

склади

номер паспорта керівника	код	назва	адреса
АН564646	1	Васильківський	м. Київ, вул. Вас...
СР486823	2	Голосіївський	м. Київ, вул. Ло...
АА123456	3	Ужгородський	м. Ужгород, вул...

Рис. 2.27: Приклад табличного подання зв'язку «працівник керує складом» (варіант 2)

Аналізуючи цю реляційну схему та приклад табличного подання до неї (дивіться Рисунок 2.27) можна зауважити, що тут для кожного складу відомий номер паспорта його керівника, а отже кожен склад має керівника, що цілком відповідає предметній області нашої майбутньої бази даних. З іншого боку, зовсім не обов'язково працівник має бути керівником складу. Для цього достатньо жодного разу не вказати його номер паспорта у

відповідному полі таблиці «склади».

Отже, коректним є другий варіант реляційної моделі. Кажуть, що склади пов'язані з працівниками за допомогою полів «номер паспорта керівника» та «номер паспорта» цих таблиць. Ці поля на схемі так само з'єднують лінією.

Оскільки логічна модель повинна розширювати знання, подані на концептуальній моделі, з неї зокрема повинна очевидно випливати множинність зв'язку. Тому тут і далі ми позначатимемо зовнішній ключ, який залучено в зв'язку множинністю «один до одного», поміткою *(U)* — від англійського слова unique — унікальний (дивіться Рисунок 2.28). Дійсно, сукупність значень зовнішнього ключа, що бере участь в такому зв'язку, не містить двох однакових елементів.

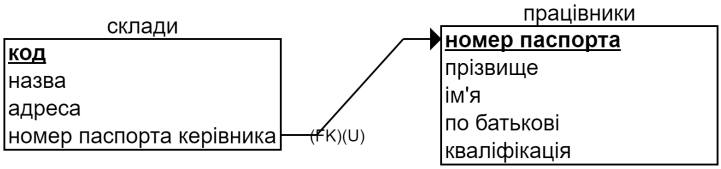


Рис. 2.28: Реляційна модель зв'язку «працівник керує складом»

Вправа 33

Доведіть вищенаведене твердження.

Сформулюємо загальні рекомендації до побудови реляційної моделі зв'язків множинності «один до одного». Отже, *для відтворення зв'язку множинності «один до одного» на реляційній моделі слід до полів зв'язаної таблиці додати ключ основної таблиці. Кажуть, що таблиці пов'язані між собою за полями ключа основної таблиці і відповідного йому зовнішнього ключа зв'язаної таблиці. Відповідні поля пов'язаних таблиць на схемі з'єднують лінією.* Як і для випадку з множинністю «один до багатьох», за допомогою збереженого зовнішнього ключа легко визначити значення всіх полів пов'язаного запису основної таблиці.

Вправа 34

II етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас
Побудуйте реляційну модель предметної області «Головоломки», використовуючи ER-модель, побудовану у Вправі 14.

Вправа 35

IV етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас
Побудуйте реляційну модель предметної області «Натуральні числа», використовуючи ER-модель, побудовану у Вправі 15.

Зв'язки «багато до багатьох»

Раніше ми вже розглядали зв'язок «товар зберігається на складі» множинністю «багато до багатьох», ER-діаграму якого подано на Рисунку 2.29.

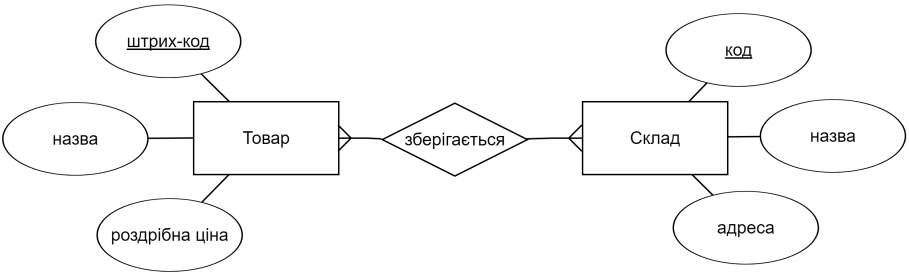


Рис. 2.29: ER-модель зв'язку «товар зберігається на складі»

Аналогічно до двох попередніх випадків, реляційна модель цього зв'язку міститиме принаймні дві таблиці: таблицю «товари» та таблицю «склади», — кожна з яких представляє сутності, залучені в цьому зв'язку. Сам зв'язок відтворити на реляційній моделі за допомогою додавання полів до однієї з таблиць, на жаль, не вдасться. Дійсно, оскільки будь-якому екземплярові будь-якої з цих двох сутностей може відповідати багато екземплярів іншої сутності, до кожного об'єкта нам довелося б додати цілий перелік ключів пов'язаних з ним об'єктів.

Насправді, зв'язок легко реалізувати в реляційній парадигмі за допомогою додаткової таблиці. Дійсно, той факт, що певний товар зберігається на певному складі, можна подати парою об'єктів — відповідних товару та складу, а отже й парою ключів відповідних сутностей. Загалом зв'язок являтиме собою множину таких пар, яку легко подати за допомогою таблиці (дивіться Рисунок 2.30).

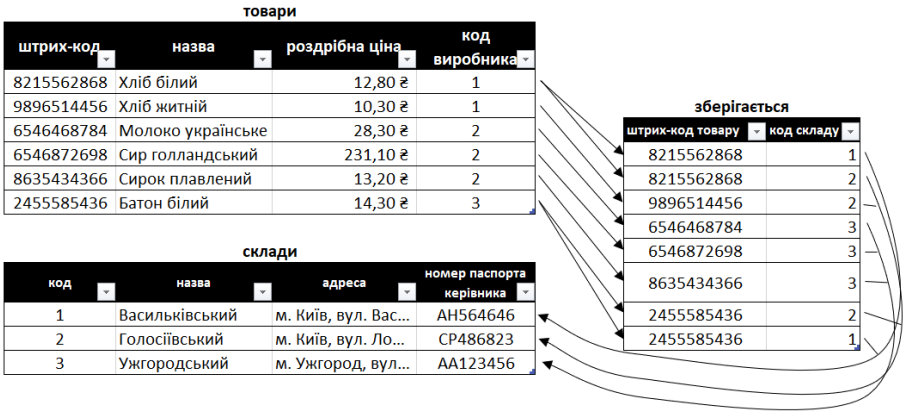


Рис. 2.30: Приклад табличного подання зв'язку «товар зберігається на складі»

Реляційну модель цього зв'язку подано на Рисунку 2.31.

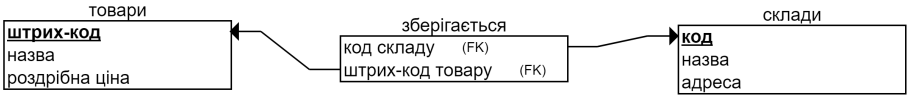


Рис. 2.31: Реляційна модель зв'язку «товар зберігається на складі»

Внаслідок додавання таблиці «зберігається» за двома зовнішніми ключами таблиць «товари» та «склади» ми можемо легко отримати інформацію з решти полів.

Якщо звернути увагу на побудовану додаткову таблицю з Рисунку 2.31, можна зауважити, що в ній факт наявності одного й того ж товару на одному й тому ж складі може зберігатися двічі, адже жодних додаткових обмежень на поля цієї таблиці не накладено. Насправді, аби уникнути повторювання інформації, слід зовнішні ключі додаткової таблиці об'єднувати

в ключ, який забезпечуватиме унікальність їх комбінації. Тому цілком коректна реляційна модель матиме вигляд, поданий на Рисунку 2.32.

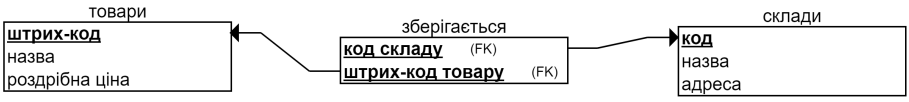


Рис. 2.32: Реляційна модель зв'язку «товар зберігається на складі»

Неважко помітити, що зв'язку множинності «багато до багатьох» на ER-діаграмі відповідає додаткова таблиця та два зв'язки множинності «один до багатьох» на реляційній моделі. Кажуть, що *додаткова таблиця розбиває зв'язок «багато до багатьох» на два зв'язки «один до багатьох»*. В даному конкретному випадку можна казати, що товари пов'язані зі складами за допомогою полів «штрих-код» та «код» цих таблиць та відповідних їм зовнішніх ключів додаткової таблиці «зберігається». Ці поля на схемі, так само як і раніше, з'єднують лініями.

Узагальнимо рекомендації до побудови реляційної моделі зв'язків множинності «багато до багатьох». Отже, *для відтворення зв'язку множинності «багато до багатьох» на реляційній моделі слід додати на неї додаткову таблицю, до полів якої додати ключі пов'язуваних таблиць. Кажуть, що таблиці пов'язані між собою через додаткову таблицю за полями ключів цих таблиць і відповідних їм зовнішніх ключів цієї додаткової таблиці. Відповідні поля пов'язаних таблиць на схемі з'єднують лінією.* Як і для випадку з іншими множинностями, за допомогою збереженого зовнішнього ключа легко визначити значення всіх полів пов'язаного запису основної таблиці.

Слід зазначити, що якщо записи зв'язаних таблиць доволі рідко беруть участь в зв'язку, для економії пам'яті та зменшення кількості порожніх полів таблиць, зв'язок множинності «один до одного» або «один до багатьох» слід також реалізовувати за допомогою допоміжної таблиці. Подібний приклад подано трохи нижче під час розгляду відтворення унарних зв'язків.

Вправа 36

Відобразіть на реляційній моделі наступні зв'язки:

- вчитель навчає клас;

- чоловік одружений з жінкою;
- мати народила дитину;
- місто належить до країни;
- гвіздок забито в дошку.

Скористайтесь ER-діаграмами, побудованими раніше у Вправі 8, попередньо визначивши ключі відповідних сутностей.

Вправа 37

Використовуючи ER-діаграму, побудовану для пісенного конкурсу «Євробачення» у 2018 році у Вправі 9, замалюйте реляційну модель відповідної бази даних.

Необов'язкові зв'язки

Переважна більшість зв'язків множинністю «один до багатьох» або «один до одного» є обов'язковими, тобто екземпляри однієї з сутностей, залучених до таких зв'язків, обов'язково принаймні один раз беруть участь у зв'язку. Проте, зовсім не для усіх таких зв'язків ця властивість є справедливою.

Розглянемо зв'язок «працівник працює в кабінеті» (Рисунок 2.33). Зазвичай доволі невелика частка працівників мережі супермаркетів працює в кабінетах — це переважно вище керівництво. Один працівник не може працювати більш, ніж в одному кабінеті одночасно, проте в одному кабінеті цілком може працювати більш, ніж один працівник — це вказує нам на те, що множинність цього зв'язку «один до багатьох».

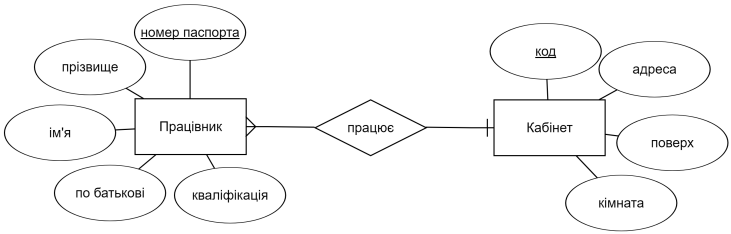


Рис. 2.33: ER-модель зв'язку «працівник працює в кабінеті»

Якщо скористатися вищенаведеними рекомендаціями для зв'язків «один до багатьох», реляційна модель матиме вигляд, поданий на Рисунку 2.34.

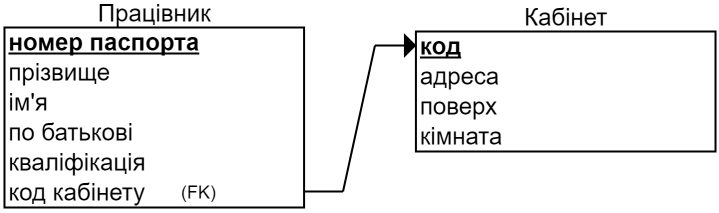


Рис. 2.34: Реляційна модель зв'язку «працівник працює в кабінеті»

Як ми вже зазначили, переважна більшість працівників мережі супермаркетів не працює в кабінетах, а отже в основному значення в стовпці «код кабінету» будуть порожніми (що означатиме, що працівник кабінету не має). Такий варіант реляційної моделі може мати право на життя, якщо необхідно часто дізнаватися, чи працює працівник в кабінеті. Проте з точки зору займаної пам'яті ця модель є неоптимальною, оскільки пам'ять під код кабінету виділяється для кожного працівника. Тому в цьому випадку більш оптимальним буде варіант реалізації, який передбачає використання допоміжної таблиці так, як це було розглянуто для зв'язків множинністю «багато до багатьох» (Рисунок 2.35).

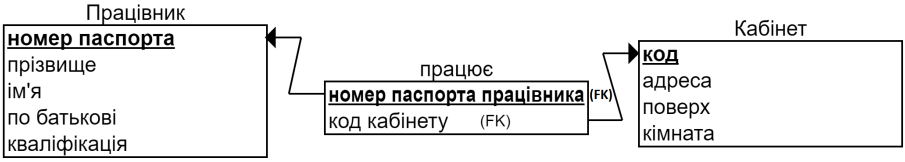


Рис. 2.35: Реляційна модель зв'язку «працівник працює в кабінеті»

Слід зауважити, що в допоміжній таблиці ключ обирається таким чином, аби забезпечувати коректну множинність зв'язку, що реалізується. В даному випадку ключем обрано саме поле «номер паспорта працівника», оскільки працівник може працювати не більш, ніж в одному кабінеті.

Аналогічною є ситуація й для зв'язків множинністю «один до одного», в яких об'єкти залучені рідко. Проте, в цьому випадку обидва зовнішні ключі допоміжної таблиці повинні позначатися як унікальні.

В сучасній сфері розробки програмного забезпечення оптимізація за пам'яттю відійшла на далекий план у зв'язку зі здешевшанням носіїв інформацію. Проте, на думку автора, другий варіант все ж варто використовувати якщо об'єкти рідко беруть участь у зв'язку. Це дозволить зробити реляційну модель більш наочною та уникати роздуття її розмірів там, де воно не критично необхідне для покращення інших показників ефективності роботи.

З іншого боку, якщо не всі, але більшість об'єктів певної сутності залучені до зв'язку, то більш оптимальним буде використання раніше розглянутої стратегії без допоміжної таблиці, так як розмір зайвої пам'яті буде невеликим, а менша кількість таблиць дасть відчутний приріст швидкості запитів до бази даних в майбутньому.

Представлення атрибутів зв'язків

Як ми вже встигли переконатися в ході розгляду концептуального проектування, атрибутами можуть володіти не тільки сутності, а й зв'язки між ними.

Найчастіше окремі атрибути мають зв'язки множинністю «багато до багатьох», і це не дивно. Оскільки решта зв'язків залучають принаймні одну сутність, що асоціюється з позначкою «один», атрибути таких зв'язків можуть розглядатися як атрибути цієї сутності. Так, атрибут «класна кімната» зв'язку «вчитель керує класом», множинність якого «один до одного», може розглядатися також як атрибут класу. Тому зупинимося детально на відтворенні атрибутів саме зв'язків «багато до багатьох».

Вище ми вже зупинялися на розгляді зв'язку «товар зберігається на складі», проте навмисно в спрощеному його вигляді. Насправді, часто цінною буде інформація не тільки про те, які саме товари де зберігаються, але й в якій кількості. Разом з цим атрибутом зв'язку ER-модель набуде вигляду, поданого на Рисунку 2.36.

Згадуючи, що зв'язок множинністю «багато до багатьох» реалізується на реляційній моделі за допомогою додаткової таблиці, можна помітити, що кожен запис цієї додаткової таблиці відповідає за пов'язування певних товару та складу. А тому, якщо додати до цієї таблиці поле «кількість», ми зможемо зберігати також інформацію й про те скільки саме кожного товару на кожному складі. Відповідна реляційна модель зображена на Рисунку 2.37.

Отже, *для подання на реляційній моделі атрибутів зв'язку «багато до багатьох» слід до додаткової таблиці, за допомогою якої цей зв'язок відтворюється, додати поля, що відповідають цим атрибутам.*

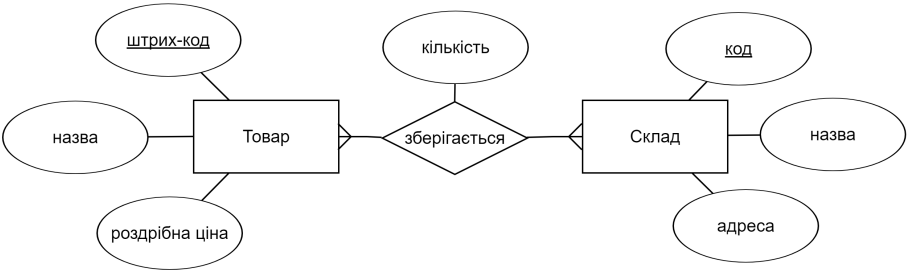


Рис. 2.36: ER-модель зв'язку «товар зберігається на складі» з атрибутом «кількість»

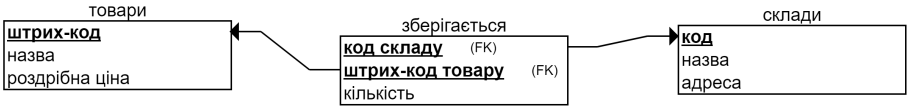


Рис. 2.37: Реляційна модель зв'язку «товар зберігається на складі» з атрибутом «кількість»

Вправа 38

Побудуйте ER- та реляційну моделі наступних зв'язків:

- чоловік одружений з жінкою в певний день;
- студент отримав результат оцінювання з предмету;
- мати народила дитину в певний день.

Скористайтесь ER-діаграмами, побудованими раніше у Вправі 8, попередньо визначивши ключі відповідних сутностей.

2.2.4 Побудова реляційної моделі тернарних зв'язків

Особливість тернарних зв'язків полягає в тому, що в таких зв'язках залучені не дві, а три сутності предметної області, а отже вищенаведені інструкції до них не можуть бути застосованими. Проте, ці інструкції нескладно адаптувати до зв'язків більших ступенів.

Найчастіше серед зв'язків зі ступенем три трапляються ті, що мають

множинність «багато до багатьох до багатьох», тому ми детальніше зупинимося саме на цьому випадку.

Раніше ми вже розглядали тернарний зв'язок «товар привезено зі складу в супермаркет», ER-модель якого подано на Рисунку 2.38.

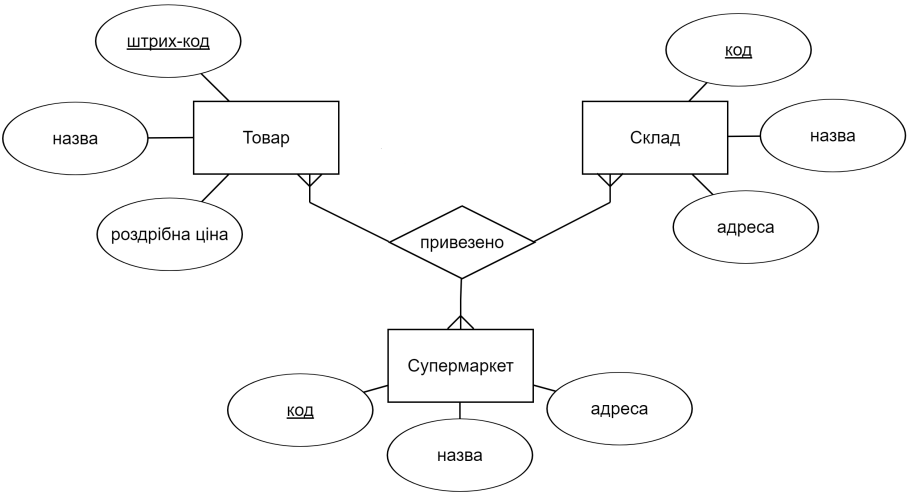


Рис. 2.38: ER-модель зв'язку «товар привезено зі складу в супермаркет»

Насправді, процес відтворення цього зв'язку на реляційній моделі доволі схожий на процес відтворення на ній бінарного зв'язку множинністю «багато до багатьох». Дійсно, обійтися трьома таблицями, які відповідають залученим у зв'язок сутностям, тут не вийде, а тому слід залучати додаткову таблицю, в яку додавати зовнішні ключі відповідних сутностей, як це зображено на Рисунку 2.39.

Узагальнимо рекомендації до побудови реляційної моделі зв'язків множинності «багато до багатьох до багатьох». Отже, *для відтворення зв'язку множинності «багато до багатьох до багатьох» на реляційній моделі слід додати на неї додаткову таблицю, до полів якої додати ключі трьох пов'язуваних таблиць. Кажуть, що таблиці пов'язані між собою через додаткову таблицю за полями ключів цих таблиць і відповідних їм зовнішніх ключів цієї додаткової таблиці. Відповідні поля пов'язаних таблиць на схемі з'єднують лінією.* Як і для випадку з бінарними зв'язками, для додавання атрибутів зв'язку достатньо додати поля у допоміжну таблицю.

Рис. 2.39: Реляційна модель зв'язку «товар привезено зі складу в супермаркет»

Вправа 39

Використовуючи ER-діаграму, побудовану для пісенного конкурсу «Євробачення» у 2018 році у Вправі 11, доповніть реляційну модель, побудовану у Вправі 37.

2.2.5 Побудова реляційної моделі унарних зв'язків

В ході розгляду концептуального проектування ми розглядали унарний зв'язок «працівник керує працівником», ER-модель якого подано на Рисунку 2.40.

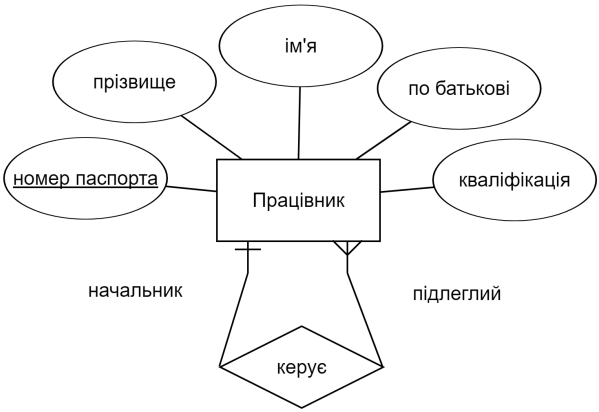


Рис. 2.40: ER-модель зв'язку «працівник керує працівником»

Оскільки зв'язок має множинність «один до багатьох», скористаємось рекомендаціями по втіленню бінарних зв'язків цієї множинності. Нагадаємо, що для відтворення зв'язку множинністю «один до багатьох» на реляційній моделі слід до полів таблиці, якій відповідає позначка «багато» («кроляча лапка» на ER-моделі), додати ключ таблиці, якій відповідає позначка «один». Отже, нам слід додати до таблиці «працівники» поле «начальник», в якому для кожного запису зберігатиметься код працівника, який є начальником відповідного йому працівника.

Коректна реляційна модель цього зв'язку зображена на Рисунку 2.41.

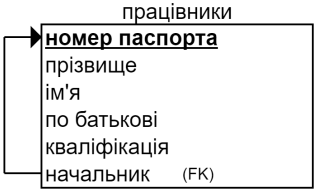


Рис. 2.41: Реляційна модель зв'язку «працівник керує працівником»

Розглянемо інший унарний зв'язок, що представляє інформацію про те, які працівники одружені один на одному. Його ER-модель зображено на Рисунку 2.42 і множинність, очевидно, «один до одного».

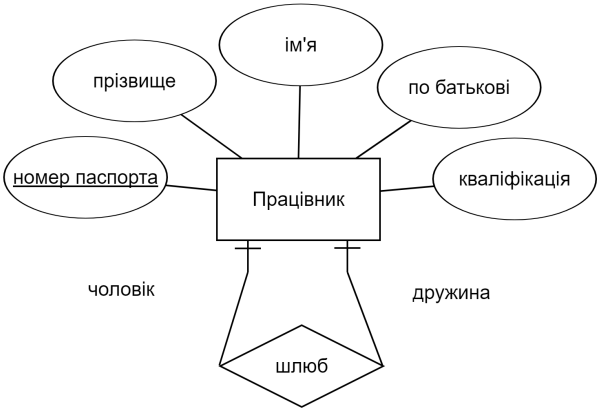


Рис. 2.42: ER-модель зв'язку «шлюб»

Оскільки зв'язок має множинність «один до одного», скористаємось ре-

комендаціями по втіленню бінарних зв'язків цієї множинності. Нагадаємо, що для відтворення зв'язку множинності «один до одного» на реляційній моделі слід до полів зв'язаної таблиці додати ключ основної таблиці.

Одним із варіантів виходу з ситуації, що склалася, є залишення поля «подружня пара» в таблиці порожнім, якщо відповідний працівник неодружений (дивіться Рисунок 2.43). Проте у Вправі 32 ми вже переконалися, що це проблематично з точки зору зарезервованої комп'ютером пам'яті, оскільки шлюби між працівниками трапляються доволі рідко.

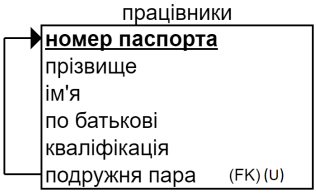


Рис. 2.43: Реляційна модель зв'язку «шлюб» з додатковим полем

Вправа 40

Які ще проблеми можуть виникнути при такій організації цього зв'язку?

Отже, ми дійшли висновку, що додавання додаткового поля не вирішить проблему в повній мірі, тому нам доведеться скористатися допоміжною таблицею. Аналогічно до того, як ми це робили у випадку зв'язку з множинністю «багато до багатьох», в допоміжну таблицю ми додаємо ключі поєднуваних таблиць (в даному випадку двічі ключ однієї й тієї ж таблиці), проте додатково вказуємо що ці поля повинні бути унікальними, аби забезпечити коректну множинність отриманої схеми. Відповідну модель подано на Рисунку 2.44.

Вправа 41

Яким чином слід змінити концептуальну та логічну моделі цього зв'язку аби забезпечити умову, що в шлюб в Україні можуть вступити тільки особи різної статі?

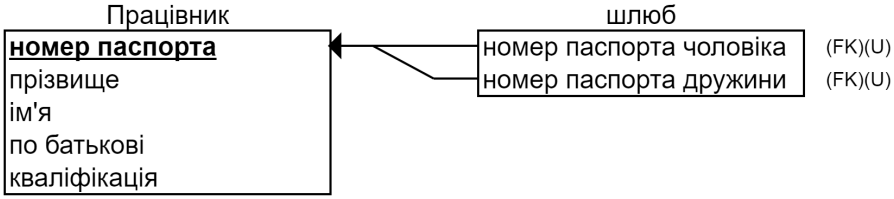


Рис. 2.44: Реляційна модель зв'язку «шлюб» з додатковою таблицею

Унарні зв'язки множинністю «багато до багатьох» відтворюються на реляційній моделі так само, як і бінарні зв'язки тієї ж множинності.

Вправа 42

Побудуйте реляційну модель зв'язку між двома футбольними командами, що грають матч Ліги Чемпіонів, із зазначенням результату матчу, на основі його ER-моделі, побудованої у Вправі 10. Вважайте, що назва команди є унікальною.

Вправа 43

Побудуйте ER- та реляційну моделі зв'язку «працівник товаришує з працівником».

2.2.6 Складений зовнішній ключ

Оскільки зовнішній ключ є відображенням ключа певної пов'язаної таблиці, далеко не завжди він складається лише з одного поля. Так, якщо в таблиці «працівники» поле «номер паспорта» розділити на два: «серія паспорта» та власне шестицифровий «номер паспорта», — зовнішній ключ в таблиці «склади» також доведеться видозмінити.

Якщо зовнішній ключ складається з двох або більше полів, його називають *складеним*.

У випадку складених зовнішніх ключів на реляційній моделі з'єднується їх поєднання та відповідний ключ іншої таблиці (Рисунок 2.45). Кажуть,

що зв'язок «працівник керує складом» зв'язує відповідні таблиці за двома полями одночасно.

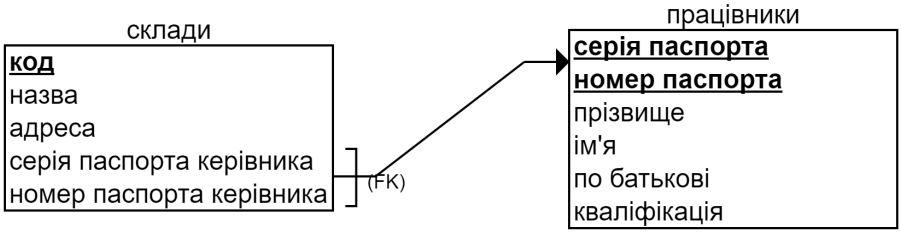


Рис. 2.45: Реляційна модель зв'язку «працівник керує складом» зі складеним зовнішнім ключем

2.2.7 Побудова логічної моделі предметної області

Оскільки ми ґрунтовно розглянули засоби логічного проектування реляційних баз даних, ми можемо сформулювати покроковий план, за яким побудова такої моделі повинна відбуватися.

Отож, проектування на даному рівні передбачає виконання наступних етапів:

1. **Відтворення сутностей.** Як ми вже кілька разів зазначали, кожній сутності предметної області на логічному рівні проектування відповідає окрема таблиця, причому атрибути сутності відображаються в поля цієї таблиці. Тому найпершим етапом у створенні повноцінної реляційної схеми є саме перетворення сутностей у відповідні їм таблиці. Ключі таких таблиць будуть співпадати з ключами сутностей предметної області на концептуальній моделі.
2. **Відтворення зв'язків.** В рамках цього кроку логічного проектування здійснюється перетворення зв'язків з ER-діаграми на реляційні зв'язки між таблицями. Для бінарних та унарних зв'язків множинністю «один до одного» та «один до багатьох» цей процес найчастіше супроводжується додаванням зовнішнього ключа в зв'язану таблицю, для бінарних та унарних зв'язків множинністю «багато до багатьох», а також для тернарних зв'язків множинністю «багато до багатьох до багатьох» на цьому етапі буде створено допоміжні таблиці із зовнішніми ключами пов'язуваних таблиць. *Нагадуємо, що якщо записи пов'язуваних таблиць доволі рідко беруть участь в зв'язку, для*

економії пам'яті та зменшення кількості порожніх полів таблиць, зв'язок множинності «один до одного» або «один до багатьох» слід також реалізовувати за допомогою допоміжної таблиці.

3. **Побудова реляційної моделі.** Цей етап передбачає формування вихідного документу концептуального рівня проектування — реляційної схеми, яка фіксує дані про набір таблиць реляційної бази даних та зв'язків між ними, отриманий на зазначених вище етапах. Отримана в результаті застосування розглянутих в цьому підрозділі технік модель є відправною точкою для проектування на наступному — фізичному рівні. На цьому етапі слід переконатися, що відповідна схема в повній мірі відображає інформацію про таблиці, їх поля, ключі, зв'язки між таблицями. Також важливою є можливість визначення зі схеми множинностей усіх наявних зв'язків.

Логічне проектування — це більш формалізований процес, аніж концептуальне, оскільки за ER-моделлю в переважній більшості випадків можна однозначно визначити відповідну їй реляційну схему.

Повернемося до нашого прикладу аби опанувати процес логічного моделювання більш ґрунтовно.

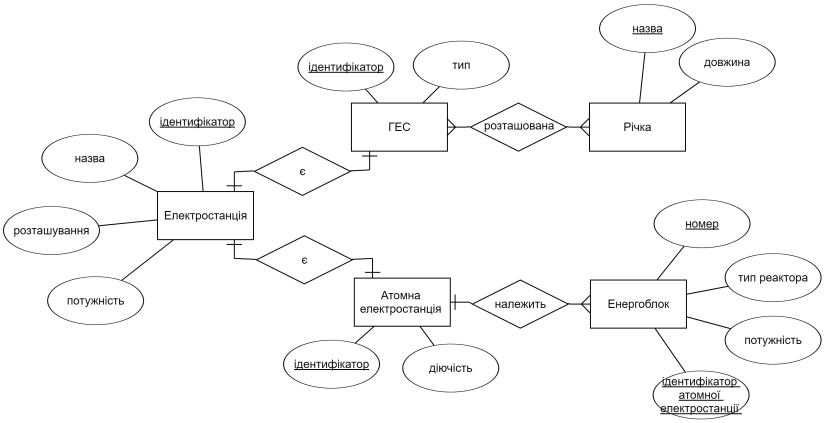


Рис. 2.46: ER-модель інформаційної системи «Енергетика»

Приклад 2

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2017», 10-11 клас

В інформаційній системі «Енергетика» користувачам надається інформація про українські атомні та гідроелектростанції. Про кожну електростанцію відомі її ідентифікатор, назва, розташування та потужність. Про кожну атомну електростанцію відомо те, діюча вона чи неактивна. Про гідроелектростанції відомий їх тип (руслова, пригреблева, деривaційна, гідроакумулювальна або припливна). Атомні електростанції складаються з енергоблоків, про які відомі номер в межах станції, тип реактора та потужність. Гідроелектростанції можуть розташовуватися на одній чи кількох річках, про які відома їх довжина. Побудуйте логічну модель відповідної бази даних та зобразіть її на реляційній схемі.

По завершенню попереднього прикладу ми отримали концептуальну модель вищеописаної предметної області «Енергетика», зображено на Рисунку 2.46. Продовжимо наш рух у напрямку створення бази даних цієї системи.

На першому етапі нам слід відтворити на реляційній моделі сутності предметної області та їх атрибути. Нагадаємо, кожна сутність на реляційній схемі представляється рівно однією таблицею, поля якої відповідають атрибутам цієї сутності. На цьому етапі схема матиме вигляд, зображений на Рисунку 2.47.

Рис. 2.47: Реляційна схема на першому кроці

Впродовж другого кроку ми повинні перетворити зв'язки між сутностями на відповідні їм зв'язки між таблицями. Розглянемо кожен зв'язок окремо.

Зв'язок «енергоблок належить до атомної електростанції» має множинність «один до багатьох». Відповідно до розглянутого раніше ма-

теріалу, в такому випадку слід до полів таблиці, якій відповідає позначка «багато», додати ключ таблиці, якій відповідає позначка «один». Оскільки зовнішній ключ таблиці «атомні електростанції» вже було додано раніше для формування ключа сутності «Енергоблок», ми використаємо вже наявне поле «ідентифікатор АЕС» для побудови зв'язку. В результаті отримуємо реляційну схему зв'язку, подану на Рисунку 2.48.

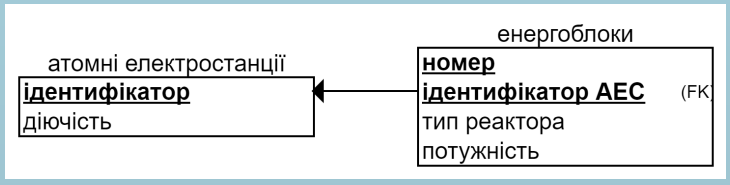


Рис. 2.48: Реляційна схема зв'язку «енергоблок належить до атомної електростанції»

Зв'язок «гідроелектростанція розташована на річці» має множинність «багато до багатьох». Відповідно до розглянутого раніше матеріалу, в такому випадку слід додати на реляційну схему допоміжну таблицю, до полів якої додати ключі пов'язуваних таблиць. Отже, слід створити нову таблицю «розташована», полями якої будуть зовнішні ключі, взяті з таблиць «ГЕС» («ідентифікатор ГЕС») та «річки» («назва річки»). Між зовнішніми та відповідними їм основними ключами будуємо зв'язки, як це зображено на Рисунку 2.49.

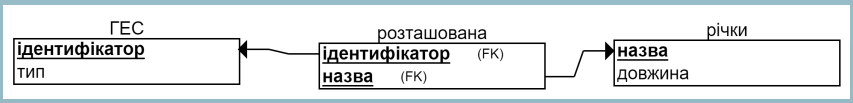


Рис. 2.49: Реляційна схема зв'язку «гідроелектростанція розташована на річці»

Зв'язок «гідроелектростанція є електростанцією» має множинність «один до одного» (дивіться Теорему 1). Відповідно до розглянутого раніше матеріалу, в такому випадку слід до полів зв'язаної таблиці додати ключ основної таблиці. Пригадаємо, відповідно до Теореми 4, у зв'язку типу «загальний вид-різновид» таблиця, що відповідає за загальний вид, завжди є основною, а таблиця, що відповідає за різновид — зв'язаною. Отже, основною є таблиця «електростанції», а зв'язаною

— «ГЕС». Оскільки зовнішній ключ таблиці «електростанції» вже було додано раніше для формування ключа сутності «ГЕС», ми використаємо вже наявне поле «ідентифікатор» для побудови зв'язку. В результаті отримуємо реляційну схему зв'язку, подану на Рисунку 2.50.

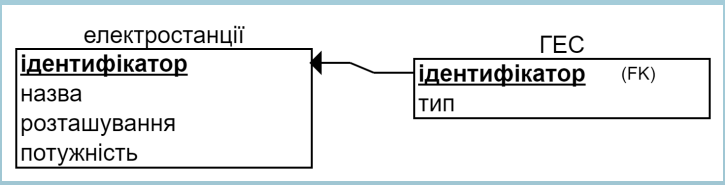


Рис. 2.50: Реляційна схема зв'язку «гідроелектростанція є електростанцією»

Зв'язок «атомна електростанція є електростанцією» має схожий зміст до зв'язку «гідроелектростанція є електростанцією». Побудувати його модель пропонується читачеві самостійно.
Після того, як всі сутності та зв'язки було втілено в реляційній парадигмі, слід скомпонувати єдину реляційну схему з усіх таблиць та зв'язків між ними. На основі вищесказаного, ця схема для предметної області «Енергетика» матиме вигляд, поданий на Рисунку 2.51.

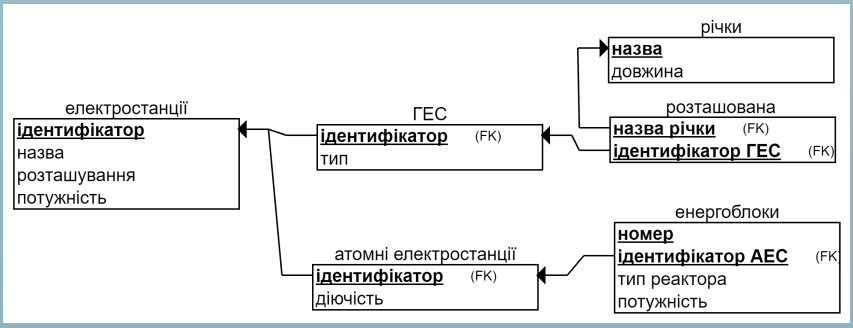


Рис. 2.51: Реляційна схема предметної області «Енергетика»

Вправа 44

I Всеукраїнська учнівська Інтернет-олімпіада з інформаційних технологій, тренувальний тур I етапу

Використовуючи ER-діаграму, побудовану для інформаційної системи «Школа» у Вправі 17, побудуйте реляційну модель бази даних.

Вправа 45

І етап Інтернет-олімпіади з інформаційних технологій «IOIT-2016», 10-11 клас
Використовуючи ER-діаграму, побудовану для інформаційної системи «Файлова система» у Вправі 18, побудуйте реляційну модель відповідної бази даних.

Вправа 46

IV етап І Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи ER-діаграму, побудовану для предметної області «Водні ресурси» у Вправі 19, побудуйте реляційну модель відповідної бази даних.

Вправа 47

IV етап ІІІ Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи ER-діаграму, побудовану для предметної області «Географічна карта» у Вправі 20, побудуйте реляційну модель бази даних.

Вправа 48

IV етап IV Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи ER-діаграму, побудовану для предметної області «Лабіринти» у Вправі 21, побудуйте реляційну модель відповідної бази даних.

2.3 Фізичне проектування бази даних

На цьому етапі проектування здійснюється адаптація реляційної моделі під можливості конкретної системи управління базами даних та власне фізичне втілення самої бази даних. В рамках цієї книги ми розглянемо особливості роботи в двох системах управління базами даних: Microsoft Access як однієї з найкращих для навчання, а також MySQL як найбільш популярної серед розробників програмного забезпечення.

2.3.1 Реалізація реляційної моделі в СУБД Microsoft Access

Microsoft Access — система управління базами даних, що входить до складу пакету офісних програм Microsoft Office Professional та є досить популярною в навчальних закладах в усьому світі, де використовується здебільшого для навчання студентів основам баз даних.

Для опанування цього підрозділу книги вам знадобиться встановлений пакет Microsoft Office Professional 2016. Цей офісний пакет є пропрієтарним програмним забезпеченням, тобто для його використання необхідно придбати ліцензію. Ви можете використовувати й інші версії цього програмного продукту, проте деякі інструменти в них можуть відрізнятися від тих, які розглядаються в даній книзі.

Огляд середовища

Microsoft Access для пристроїв під управлінням операційної системи Windows являє собою настільний застосунок, що надає можливість маніпуляції базами даних, а також створення інформаційних систем на їх базі.

Система управління базами даних підтримує роботу з п'ятьма видами об'єктів: таблиці, запити, форми, звіти та макроси. В рамках цього підрозділу книги ми докладно розглянемо принципи реалізації реляційних моделей в цьому середовищі — тобто власне роботу з таблицями та зв'язками між цими таблицями. Пізніше в цій частині книги ми розглянемо принципи побудови запитів, а детальний розгляд форм, звітів та макросів як інструментів створення інтерфейсу вміщений в наступній частині книги, що присвячена інформаційним системам.

Створення баз даних

Під час запуску програмного продукту користувачеві пропонується обрати одну з останніх відкритих на комп'ютері баз даних, створити нову по-

рожню базу даних, або ж створити базу даних з одного з багатьох макетів вбудованої бібліотеки.

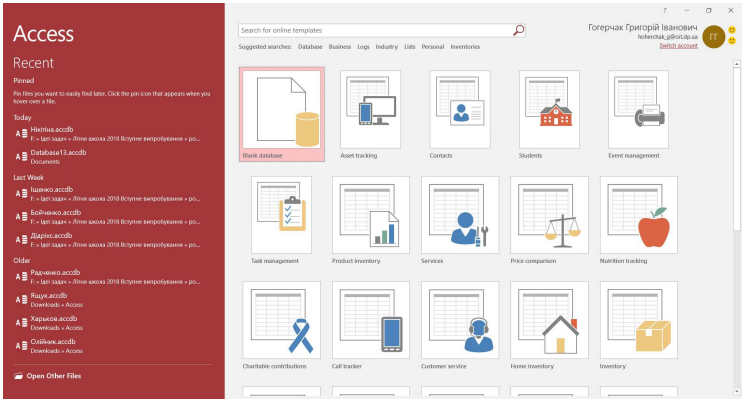


Рис. 2.52: Стартове меню Microsoft Access

В ході розгляду даної книги ми створюватимемо бази даних «з нуля», а тому в стартовому вікні (дивіться Рисунок 2.52) практично завжди обиратимемо пункт *Blank database («Порожня база даних»)*.

Після натиснення на цей пункт меню система відобразить вікно, в якому користувачеві буде надана можливість обрати розташування майбутньої бази даних (Рисунок 2.53). Натискання кнопки *Create (Створити)* створює порожню базу даних у тому розташуванні, яке було обрано користувачем. За промовчанням це бібліотека Документи поточного користувача системи.

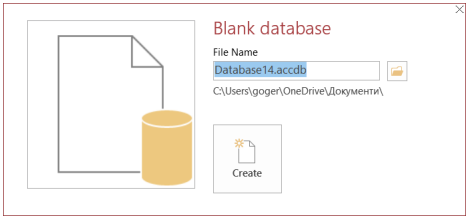


Рис. 2.53: Вікно створення нової бази даних

За виглядом вікна, що відкривається по цьому (Рисунок 2.54), можна подумати, що база даних насправді не порожня, адже тут вже відображається таблиця Table1. Ця таблиця насправді ще не збережена в базі даних.

Система просто пропонує вам почати наповнення бази даних створенням першої таблиці.

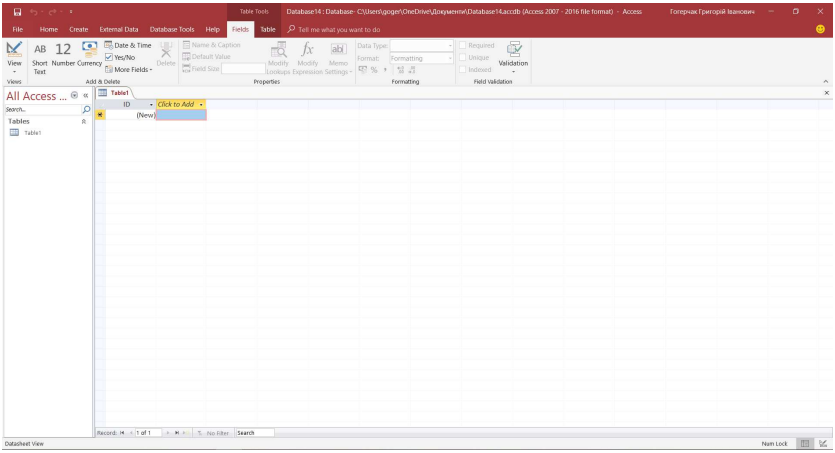


Рис. 2.54: Таблиця Table1 в порожній базі даних

Робота з таблицями

Створити нову таблицю можна також в будь-який момент за допомогою інструментів групи *Tables (Таблиці)* вкладки *Create (Створити)* головного меню програми (Рисунок 2.55).

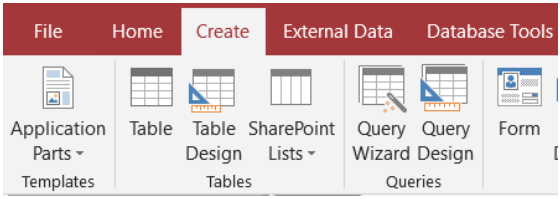


Рис. 2.55: Інструменти для створення таблиць

Загалом є дві форми подання таблиць: *Datasheet View (Подання таблиці)* та *Design View (Подання конструктора)*. Перемикатися між ними можна за допомогою інструментів в правій нижній частині вікна програми. Перше подання є стандартним поданням таблиць: в ньому відображаються як поля (підписи стовпців), так і дані таблиці (рядки). Його використовують

найчастіше для наповнення таблиць інформацією. Друге подання дозволяє зосередитися на структурі таблиці та відображає тільки поля з їх характеристиками.

Вигляд подання конструктора подано на Рисунку 2.56. В верхній частині подання подано таблицю з трьома стовпцями. До першого стовпця вносяться назви полів таблиці, в другому — обирається їх тип. Заповнення третього стовпця є необов'язковим, але сюди можна вводити примітки до полів таблиці.

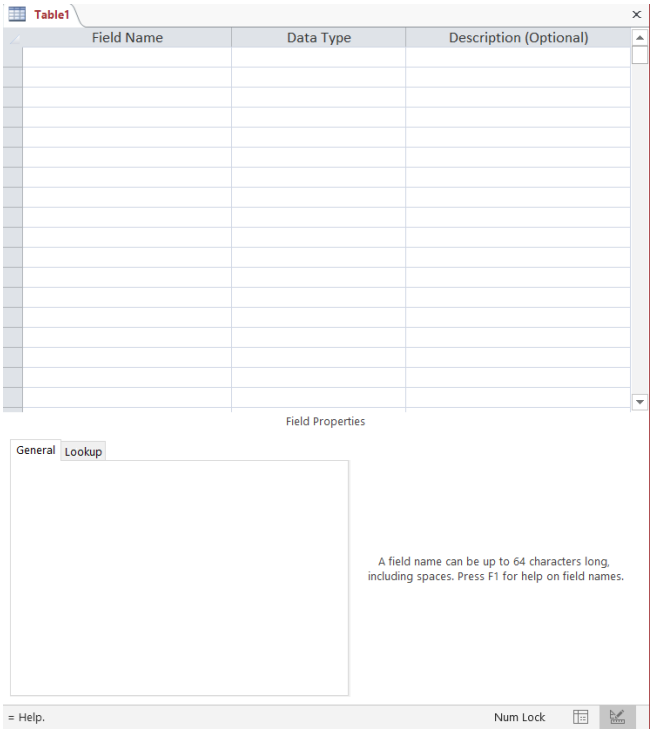


Рис. 2.56: Подання конструктора (Design View)

Тип є конкретизацією домену атрибута, який ми визначали на концептуальному етапі проектування. Всього у Microsoft Access 2016 є можливість для кожного з полів обрати один з 13 типів:

- *Short Text (Короткий текст)* — призначений для збереження в полі тексту, довжина якого не перевищує 255 символів;

- *Long Text (Довгий текст)* — призначений для збереження тексту розміром до 1 Гб з можливістю його форматування;
- *Number (Число)* — призначений для збереження цілих або дійсних чисел;
- *Large Number (Велике число)* — призначений для збереження великих (від -2^{63} до 2^{63}) цілих чисел;
- *Date/Time (Дата й час)* — призначений для збереження дати й часу;
- *Currency (Грошова одиниця)* — призначений для збереження грошових значень;
- *AutoNumber (Автонумерація)* — дає змогу вказати унікальний номер запису, найчастіше використовується для створення автоматичного штучного ключа таблиці;
- *Yes/No (Так/Ні)* — призначений для збереження логічного значення (Так або Ні), а також вибору одного з двох значень;
- *OLE Object (Об'єкт OLE)* — призначений для збереження вкладених об'єктів OLE, наприклад, електронних таблиць Excel; має значно вужчі можливості, ніж тип Вкладення, залишений для зворотної сумісності;
- *Hyperlink (Гіперпосилання)* — призначений для збереження гіперпосилання на локальні або глобальні ресурси;
- *Attachment (Вкладення)* — призначений для збереження зображень, файлів електронних таблиць, документів, діаграм та інших типів файлів; є аналогом вкладених файлів в електронних листах;
- *Calculated (Обчислюваний)* — призначений для збереження результату деякого обчислення на основі полів таблиці;
- *Lookup Wizard (Майстер підстановок)* — призначений для відображення списку значень для вибору; пітримує типи даних Текст та Число.

Типи полів *Обчислюваний* та *Майстер підстановок* будуть детальніше розглянуті в наступних розділах цієї книги.

Отже, для створення таблиці слід в поданні конструктора вказати назви всіх полів та їх типи (заповнити перший та другий стовпці відповідно).

Наприклад, для реалізації реляційної моделі таблиці «працівники» слід заповнити таблицю в режимі конструктора так, як це показано на Рисунку 2.57.

Ви вже мабуть помітили, що ми ще не позначили ключ нашої таблиці. Дійсно, після введення даних про поля таблиці слід додатково встановити, які поля складатимуть ключ. Для цього із затиснутою клавішею Ctrl ми обираємо усі поля, що належать до ключа, та натискаємо на інструмент *Primary Key (Первинний ключ)* (Рисунок 2.58) вкладки *Design (Конструктор)* головного меню. Після цього біля відповідних полів зліва з'явиться

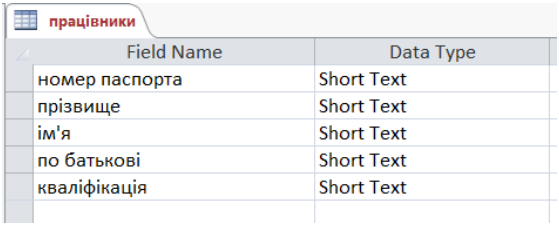


Рис. 2.57: Подання конструктора для таблиці «працівники»

іконка ключа — це означає, що ключ успішно встановлено.

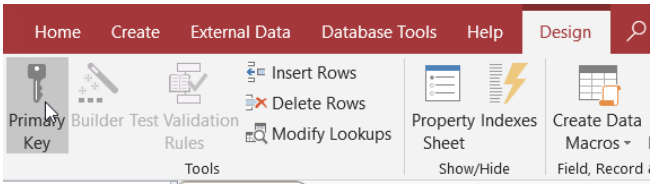


Рис. 2.58: Інструмент *Primary Key (Первинний ключ)*

Врешті-решт подання конструктора для таблиці «працівники» набуде вигляду, поданого на Рисунку 2.59.

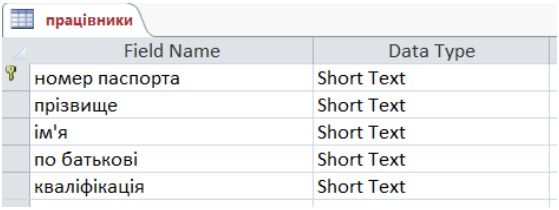


Рис. 2.59: Подання конструктора для таблиці «працівники»

Такій схемі таблиці відповідає табличне подання, подане на Рисунку 2.60.

Вправа 49

Реалізуйте в базі даних таблиці реляційної моделі предметної області «Мережа супермаркетів», яку ми побудували в минулому підрозділі (Рисунок 2.61), не включаючи налаштування унікальних полів. Ви-

працівники

номер паспо	прізвище	ім'я	по батькові	кваліфікація
AA123456	Іванов	Андрій	Сергійович	вища
AH564646	Андрієнко	Роман	Григорович	вища
OH654643	Петренко	Віктор	Борисович	вища
HH564127	Бобров	Семен	Іванович	1 кат.
CP486823	Титаренко	Віктор	Леонідович	3 кат.
BP556465	Боголюбов	Борис	Арсенович	вища

Рис. 2.60: Табличне подання для таблиці «працівники»

значте первинні ключі створених таблиць.

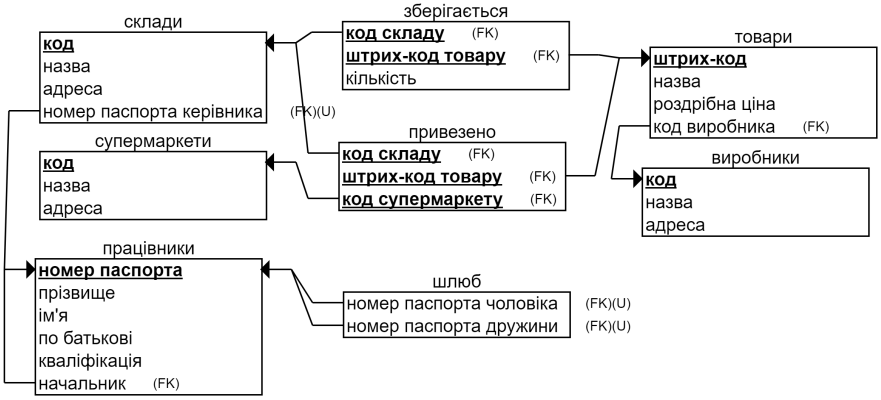


Рис. 2.61: Реляційна модель предметної області «Мережа супермаркетів»

Властивості полів

Окрім типу, поля таблиць в системі управління базами даних Microsoft Access мають ще низку різних властивостей, налаштування яких дозволяє більш точно відтворити вимоги предметної області.

Ці властивості розташовані в нижній частині подання конструктора на аркуші, приклад якого подано на Рисунку 2.62.

Основні властивості полів наступні:

- *Field Size (Розмір поля)* — визначає максимальний розмір даних, що

General | Lookup

Field Size	255
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Text Align	General

Рис. 2.62: Аркуш властивостей поля

можуть бути збережені в полі; для чисел може також уточнювати їх тип: ціле, дійсне тощо;

- *Format (Формат)* — задає формат виводу даних на екран;
- *Decimal Places (Число десяткових знаків)* — визначає кількість знаків після десяткового розділювача;
- *Input Mask (Маска введення)*;
- *Caption (Підпис)* — задає напис, що використовується у формах і звітах замість імені поля;
- *Default Value (Значення за промовчанням)* — задає значення, яке буде присвоєно полю при створенні запису;
- *Validation Rule (Умова на значення)* — вираз умовного типу, який задає правило, яке повинно задовільняти поле;
- *Validation Text (Повідомлення про помилку)* — задає текст повідомлення, який буде відображено, якщо умова на значення хибна;
- *Required (Обов'язкове поле)* — визначає чи є поле обов'язковим для заповнення;
- *Indexed (Індексоване поле)* — визначає чи є поле індексованим.

Далі ми зупинимось на двох елементах вищенаведеного списку: індексації та масках введення. Умови на значення буде детально розглянути згодом.

Індексація та унікальні поля

Індекс дає змогу швидше шукати та сортувати записи в базі даних. В індексі зберігаються відомості про розташування записів з урахуванням поля або полів, вибраних для індексування. Після того як система управління

базами даних отримає дані про розташування з індексу, вона може видобути їх, перейшовши безпосередньо в потрібне розташування. Завдяки цьому використання індексу може займати набагато менше часу, ніж сканування всіх записів для пошуку даних.

Сенс створювати індекс за певним полем або комбінацією полів є тоді, коли це поле чи комбінація містять велику кількість різних значень. Також слід зазначити, що в Access поля, що мають тип *Об'єкт OLE*, *Обчислюваний* або *Вкладення*, до індексу включені бути не можуть.

У разі створення унікального індексу, Access не дозволяє вводити нове значення в поле, якщо це значення призводить до повторення значення набору, за яким здійснюється індексування.

Microsoft Access автоматично створює індекси для ключів таблиць. Решту індексів слід налаштовувати власноруч. Оскільки поля таблиць в цій системі не мають властивості унікальності, додаткова можливість, яку надає індексація в Access, — це створення унікальних полів та унікальних комбінацій полів. Зокрема ця можливість цінна при створенні зв'язків множинністю «один до одного».

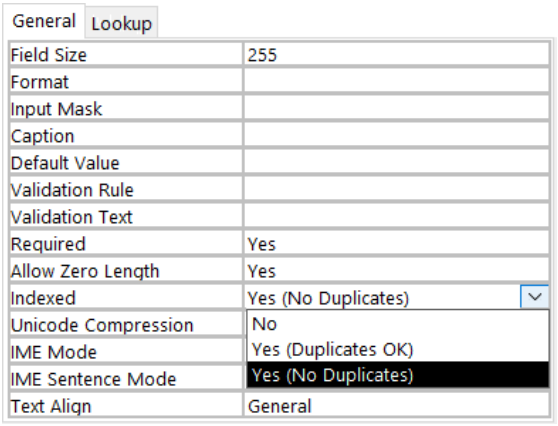


Рис. 2.63: Індексоване поле

Для створення індексу за одним полем достатньо обрати пункт *Yes (Так)* в розкривному списку властивості *Indexed (Індексоване поле)* цього поля. Слід звернути увагу, що в цьому списку є дві опції, що відповідають індексації (Рисунок 2.63):

- *Duplicates OK (Повторення дозволені)* — створює звичайний індекс за поточним виділеним полем;

- *No Duplicates (Без повторень)* — створює унікальний індекс за поточним виділеним полем, тобто це поле також стає унікальним.

Для створення індексу на основі кількох полів одночасно слід скористатися інструментом *Indexes (Індекси)* вкладки *Design (Конструктор)* головного меню (Рисунок 2.64).

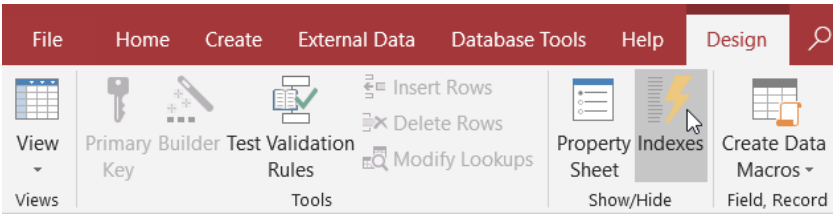


Рис. 2.64: Індексоване поле

Вікно, яке відкриває цей інструмент, містить усі активні індекси поточної таблиці. Наприклад, на Рисунку 2.65 ми можемо переконатися, що таблиця «працівники» має один індекс — він же первинний ключ з одного поля «номер паспорта».

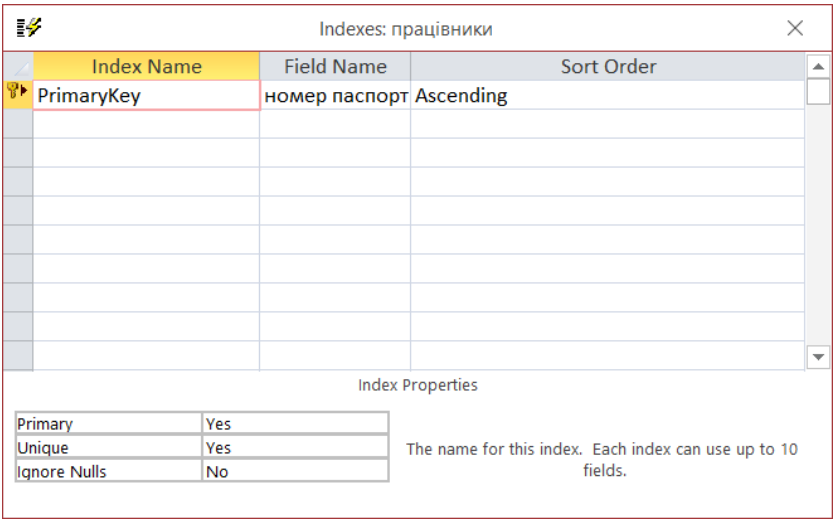


Рис. 2.65: Меню індексів

Аби додати індекс для поточної таблиці, слід ввести назву індексу в ново-

му рядку таблиці в цьому вікні, а також назву поля та порядок сортування.

Якщо ж індекс має складатися з кількох полів одночасно (наприклад, «прізвище», «ім'я» та «по батькові»), достатньо ввести їх один за одним в стовпчик без дублювання назви індексу, як це показано на Рисунку 2.66.

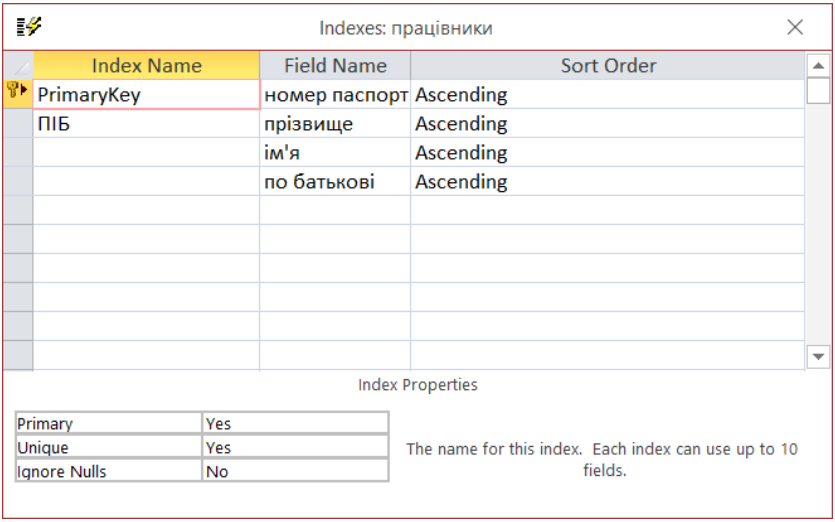


Рис. 2.66: Складений індекс

В нижній частині вікна розташовані параметри індексів (*Index Properties*). Якщо необхідно створити унікальний індекс, слід змінити значення параметра *Unique (Унікальний)* з *No (Ні)* на *Yes (Так)*, клацнувши перед цим на відповідний індекс. Також тут за допомогою властивості *Ignore Nulls (Ігнорування порожніх)* можна встановити, чи потрібно враховувати порожні поля при індексуванні.

Вправа 50

Забезпечте унікальність значень полів «номер паспорта керівника» (таблиця «склади»), «номер паспорта чоловіка» та «номер паспорта дружини» (таблиця «шлюб») для бази даних, побудованої у Вправі 49, для створення зв'язків «один до одного» в майбутньому.

Маска введення даних

Означення 35. *Маска введення – це рядок символів, який задає формат припустимих значень для введення.*

За інформацією довідки Microsoft Access, маска введення використовується у випадках, коли важливо забезпечити дотримання формату введених значень. Наприклад, маскою вводу можна скористатися для поля, у якому зберігаються номери телефонів, щоб програма Access вимагала введення десяти чисел. Якщо який-небудь користувач введе номер телефону без коду населеного пункту, програма Access не записуватиме дані, доки не буде додано код населеного пункту.

Маска введення складається з одного обов'язкового елемента та двох необов'язкових, кожен із яких відокремлюється крапкою з комою:

- перший (обов'язковий) елемент складається із послідовності символів маски, службових та текстових символів;
- другий (необов'язковий) елемент встановлюється в 0, якщо символи маски слід зберігати разом із введеними користувачем даними; значення 1 означає, що символи маски тільки відображаються, проте не зберігаються разом з введеними даними;
- третій (необов'язковий) елемент — це символ-заповнювач місця; за промовчанням в якості символу-заповнювача використовується нижнє підкреслення (_).

Символ	*Множина символів, які дозволяє*	*Обов'язковість*
0	цифра	так
9	цифра	ні
#	цифра, пробіл, знак + або −	так
L	літера	так
?	літера	ні
A	літера або цифра	так
a	літера або цифра	ні
&	символ або пробіл	так
C	символ або пробіл	ні

Табл. 2.2: Символи маски введення

Символи маски введення подано в Таблиці 2.2. Кожен із цих символів дозволяє користувачеві ввести символ із певної множини, причому деякі з

них вимагають це зробити, деякі просто дають можливість.

Окрім власне символів маски, контролювати формат введення допомагають також службові символи, подані в Таблиці 2.3.

Символ(и)	*Пояснення*
. , : ; - /	залежно від локалізації можуть бути елементами запису чисел (десятковий розділювач чи розділювач розрядів), дати та часу
>	всі наступні введені користувачем символи перетворюються на символи верхнього регістру (великі літери)
<	всі наступні введені користувачем символи перетворюються на символи нижнього регістру (малі літери)
!	вимагає заповнення маски зліва направо
\	екранування символів маски та службових символів
"..."	взята в лапки послідовність символів екранується

Табл. 2.3: Службові символи маски введення

Розглянемо наступний приклад.

Приклад 3

Побудуйте маски для введення наступних значень:

1. *номер мобільного телефону в Україні;*
2. *поштовий індекс в Україні;*
3. *код аеропорту за стандартом IATA.*

Відповіді:

1. номер мобільного телефону в Україні можна представити маскою введення *"+38(0"00)000-00-00*, адже частина +38(0 є спільною для усіх номерів, далі слідують дві обов'язкові цифри, далі закриваюча дужка, далі три цифри, дефіс, дві цифри, дефіс та ще дві цифри;
2. поштовий індекс в Україні — це п'ять цифр, а тому маска введення матиме вигляд *00000*;
3. код аеропорту IATA складається з трьох великих літер, а тому маска введення матиме вигляд *>LLL*.

Вправа 51

В новій базі даних створіть таблицю «абітурієнти» з наступними обов'язковими до введення полями:

- окремі поля «прізвище», «ім'я», «по батькові», кожне з яких дозволяє ввести першу літеру тільки у верхньому регістрі, інші – у нижньому (довжина кожного з полів не повинна перевищувати 10 символів);
- «дата народження»;
- «номер сертифікату ЗНО» — унікальне число з шести цифр;
- «email»;
- «медична довідка» — непорожній рядок;
- «серія атестата», яке дозволяє ввести рядок з двох літер верхнього регістру;
- «номер атестата», яке дозволяє ввести рядок з 8 цифр;
- «середній бал атестату» — число з однією або двома цифрами до десяткової коми та однією цифрою після неї.

Визначте потенційні та первинний ключі таблиці, враховуючи, що номер сертифікату ЗНО, email, а також серія разом з номером атестату однозначно визначають абітурієнта в їх переліку.

Цілісність даних

Вище ми розглянули як засобами Microsoft Access можна відтворити таблиці з реляційної моделі, задати для цих таблиць ключі, індекси, унікальні поля, а також детально зупинилися на понятті маски введення і методах контролю вводу даних з її допомогою. Прийшов час розглянути засоби реалізації наступного елементу реляційної моделі — зв'язків між таблицями.

На перший погляд, ми вже додали зовнішні ключі таблиць, то чи не достатньо цього для того, аби вважати зв'язок створеним? Дійсно, доданих полів достатньо аби визначити решту даних про пов'язаний об'єкт. Проте, цього не достатньо аби забезпечити перевірку коректності даних.

Розглянемо випадок, зображений на Рисунку 2.67. В якості номеру паспорта керівника складу «Голосіївський» вказано значення «ШІ687435», яке не відповідає жодному з наявних працівників. В такому разі, інформаційна система, побудована на базі даних, що допускає такого роду помилки в даних, буде працювати некоректно. Отже, необхідні механізми, які б дозволили запобігати таким помилкам ще на етапі введення даних, щоб в базі

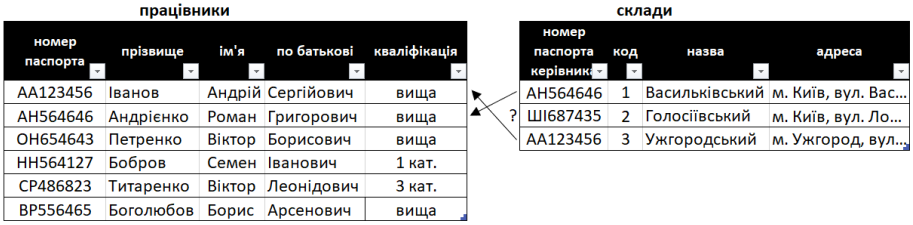


Рис. 2.67: Помилка в паспорті керівника

даних в жоден момент не було подібних неузгодженостей — механізми *підтримки цілісності даних*.

Загалом існує кілька загальноприйнятих критеріїв цілісності реляційних (табличних) даних, хоча в кожній конкретній предметній області може вимагатися дотримання певних додаткових особливих правил. Перш ніж перейти до розгляду базових правил цілісності даних, розглянемо поняття порожнього значення.

Означення 36. *Порожнє значення (часто позначається як NULL) вказує на те, що значення певного поля в даний момент невизначене або незастосовне до даного запису.*

Порожнє значення вказує на відсутність значення, тому його не слід плутати з числом 0 або рядком, що складається лише з пробілів.

Отже, основними обмеженнями цілісності є:

1. *Цілісність сутностей.* Жодне з полів первинного ключа не повинне містити порожнє значення NULL.
2. *Цілісність посилань.* Якщо в таблиці присутній зовнішній ключ, то його значення повинне або відповідати значенню ключа деякої таблиці (можливо, тієї ж), або цей зовнішній ключ повинен повністю складатися з порожніх значень NULL.

Розглянутий нами вище випадок не відповідатиме другому обмеженню, оскільки зовнішній ключ «ШІ687435» не відповідає ключу жодного запису таблиці «працівники», а отже таке значення зовнішнього ключа не повинно бути збережене.

Перейдемо до практичних шляхів забезпечення цілісності даних в Microsoft Access.

Цілісність сутностей активована в цій системі управління базами даних за промовчанням. Access не дозволить жодне з полів ключа залишити по-

рожнім.

Цілісність посилань в цій СУБД реалізовується за допомогою безпосереднього вказування зв'язків між полями таблиць бази даних. Після утворення зв'язку Access починає слідкувати за тим, щоб значення зовнішнього ключа було сумісним з відповідним йому ключем іншої або тієї ж таблиці бази даних.

Реалізація зв'язків між таблицями

Для роботи зі зв'язами Access пропонує окремий інструмент *Relationships (Зв'язки)*, розташований на вкладці *Database Tools (Робота з базою даних)* головного меню (Рисунок 2.68).

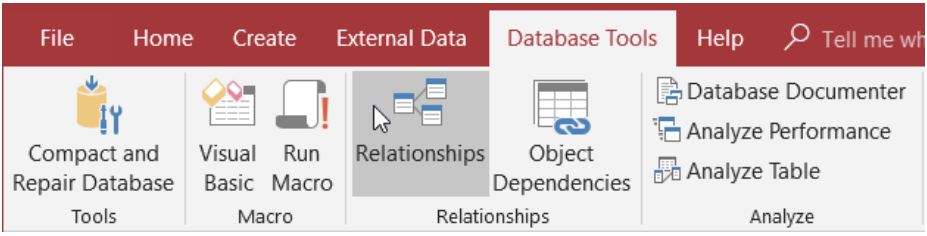


Рис. 2.68: Вкладка *Database Tools (Робота з базою даних)*

Цей інструмент надає доступ до певного аналогу реляційної схеми — схеми бази даних, на яку можна додавати вже створені таблиці та створювати зв'язки, поєднуючи поля цих таблиць.

Перетягнемо таблиці «працівники» та «склади» з переліку в лівій частині екрану на поле нашої схеми. Отримаємо модель, подану на Рисунку 2.69.

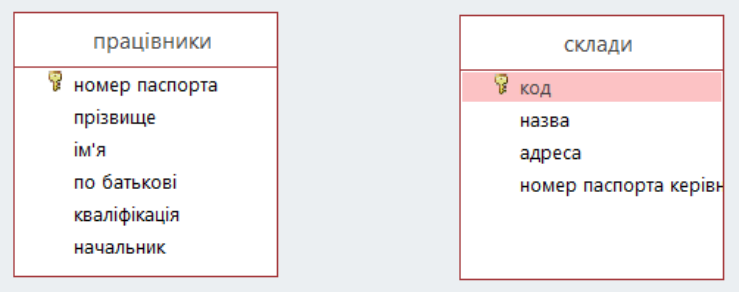


Рис. 2.69: Схема бази даних

На цій схемі таблиці подаються дещо подібно до того, як ми їх раніше подавали на реляційній моделі бази даних, проте ключ тут позначається іконкою ключа біля кожного поля, що цьому ключу належить.

Щоб створити зв'язок між таблицями «працівники» та «склади» за ключем «номер паспорта» таблиці «працівники» та зовнішнім ключем «номер паспорта керівника» таблиці «склади», слід затиснувши ліву клавішу миші перетягнути поле «номер паспорта» на поле «номер паспорта керівника», тобто *ключ на зовнішній ключ*. Після цього Access відобразить вікно створення зв'язку, подане на Рисунку 2.70.

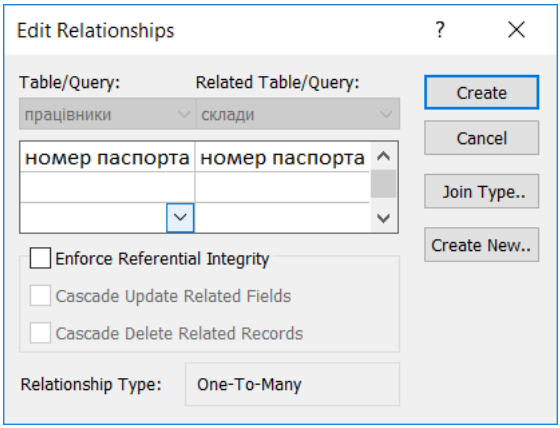


Рис. 2.70: Вікно створення зв'язку

В цьому вікні подається повна інформація про зв'язок: яка таблиця є основною, а яка — зв'язаною, за якими полями створюється зв'язок та яка його множинність (множинності тут відповідає напис *One-To-Many*, тобто «один до багатьох»).

Також в цьому вікні є можливість налаштувати три параметри відповідного зв'язку:

- прапорець *Enforce Referential Integrity (Забезпечити цілісність посилань)* активує перевірку правила цілісності посилань, яке ми розглянули вище;
- прапорець *Cascade Update Related Fields (Каскадне оновлення зв'язаних полів)* активує стан, в якому при оновленні значення ключа основної таблиці автоматично вносяться зміни до відповідних йому значень зовнішніх ключів зв'язаної таблиці;
- прапорець *Cascade Delete Related Records (Каскадне видалення*

зв'язаних записів) активує стан, в якому при видаленні запису основної таблиці автоматично видаляються пов'язані з ним записи зв'язаної таблиці.

Встановимо всі три вищевказані прапорці в даному вікні та натиснемо кнопку *Create (Створити)*. На схемі між таблицями «працівники» та «склади» з'явиться з'єднуюча лінія, що поєднує зовнішній ключ та відповідний йому основний (Рисунок 2.71). Причому поряд з кожною з таблиць з'явиться позначка множинності: 1 — для множинності «один» та ∞ — для множинності «багато». Даний зв'язок має множинність «один до багатьох», як ми раніше і відмічали.

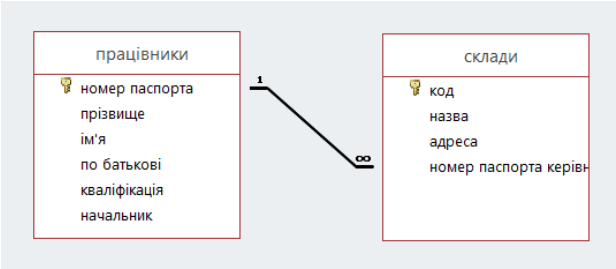


Рис. 2.71: Зв'язок між таблицями «працівники» та «склади»

Аналогічним чином можуть бути реалізовані й решта бінарних зв'язків множинністю «один до багатьох», після чого буде отримано схему, зображену на Рисунку 2.72.

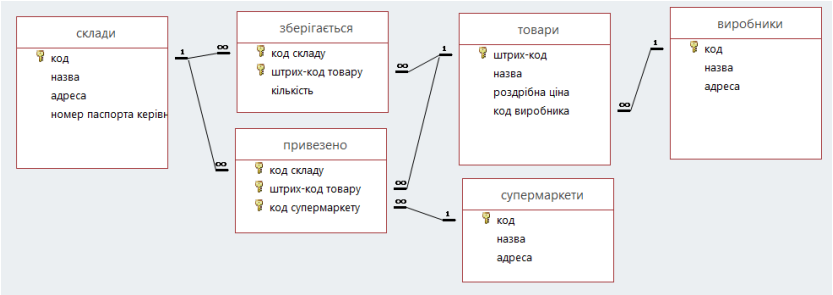


Рис. 2.72: Схема бази даних з бінарними зв'язами «один до багатьох»

Для повноцінної реалізації залишилося додати чотири зв'язки: три бінарних множинністю «один до одного» і один унарний множинністю «один

до багатьох».

Створення бінарних зв'язків «один до одного» ідентичне створенню зв'язків «один до багатьох»: слід *ключ основної таблиці перетягнути на зовнішній ключ зв'язаної*. Причому перетягувати потрібно саме в цей бік, інакше це змінить зміст схеми бази даних. Єдине: зовнішній ключ завжди повинен бути унікальним індексом, оскільки інакше Access розпізнаватиме зв'язок як «один до багатьох».

Вправа 52

1. Поекспериментуйте та перетягніть поле «номер паспорта дружини» на поле «номер паспорта». Спробуйте додати дані у таблиці «працівники» та «шлюб» після успішного додавання зв'язку. З якими проблемами ви зіштовхнулися?
2. Проведіть аналогічний експеримент для зв'язку «один до багатьох». Чи призведе зміна напрямку перетягування до зміни в змісті схеми даних? Як ви думаєте, чому?

Проте наша схема містить деяку особливість: два бінарних зв'язки з'єднують одні й ті ж дві таблиці «працівники» та «шлюб». На жаль, Access не дає змоги додати два зв'язки між одними й тими ж таблицями, проте є можливість все одно реалізувати їх. Для цього необхідно повторно додати таблицю «працівники» на схему бази даних, після чого вона отримає назву «працівники_1». Це все та ж таблиця «працівники», просто під іншим синонімом. Тепер можна додати ці два зв'язки так, ніби вони стосуються різних працівників. Тут одна таблиця «працівники» виступатиме в ролі «чоловік», а інша — в ролі «дружина», відповідно до того, з яким з полів таблиці «шлюб» вони з'єднані. Детальніше дивіться на Рисунку 2.73.

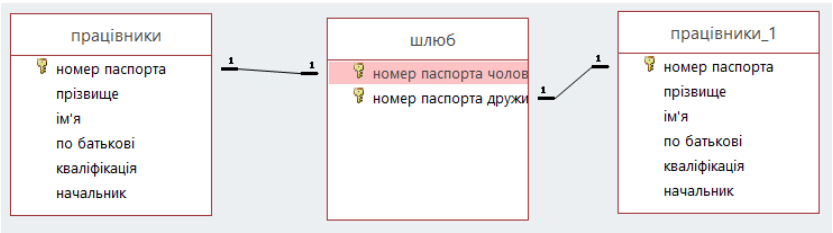


Рис. 2.73: Схема двох зв'язків між одними й тими самими двома таблицями

Створення унарних зв'язків в Access теж має деякі особливості, оскільки

перетягнути поле таблиці на інше поле цієї ж таблиці тут можливості немає.

Для успішної реалізації унарного зв'язку «працівник керує працівником» слід втретє додати таблицю «працівники» на схему даних та з'єднати їх зв'язком так, ніби одна з них відповідає ролі «начальник», а інша — ролі «підлеглий». Перетягувати, знов-таки, потрібно зовнішній ключ ролі «підлеглий» на основний ключ ролі «начальник» (Рисунок 2.74).

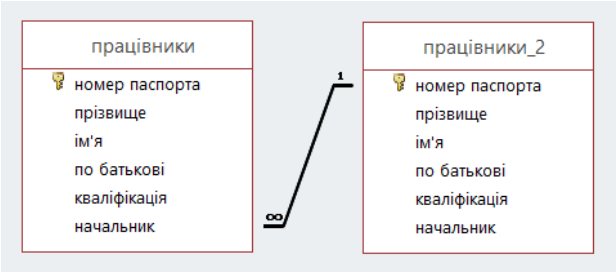


Рис. 2.74: Схема унарного зв'язку

Остаточна схема бази даних предметної області «Мережа супермаркетів» матиме вигляд, поданий на Рисунку 2.75.

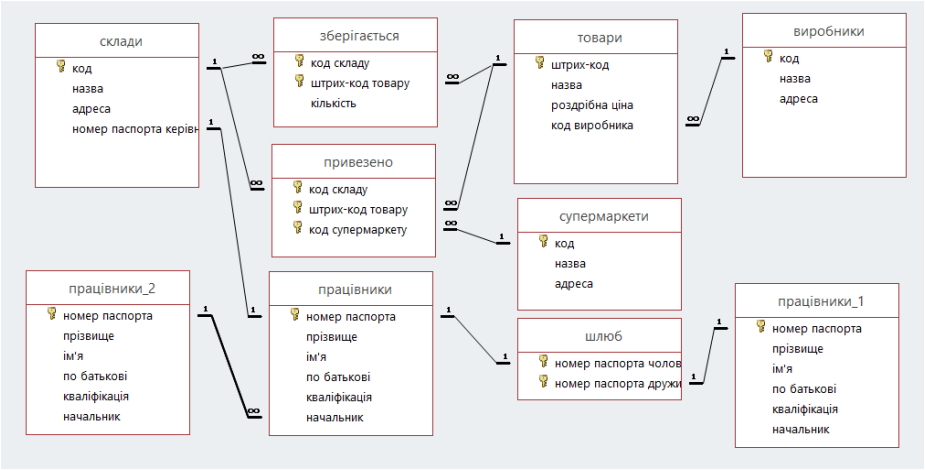


Рис. 2.75: Схема бази даних предметної області «Мережа супермаркетів»

Якщо ж в таблиці «працівники» поле «номер паспорта» розділити на

два: «серія паспорта» та власне шестицифровий «номер паспорта», — зовнішній ключ в таблиці «склади» також доведеться видозмінити.

У випадку складених зовнішніх ключів на схемі бази даних потрібно спочатку перетягнути одне з полів ключа на відповідне йому поле зовнішнього ключа. Після цього у вікні, яке з'явиться, потрібно в другому рядку з розкривних списків обрати решту відповідних полів таблиць (Рисунок 2.76).

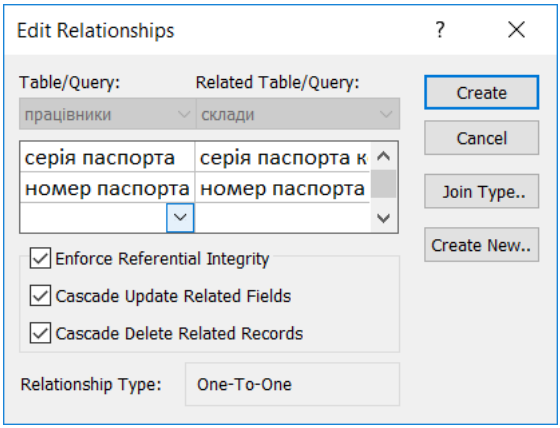


Рис. 2.76: Створення зв'язку за кількома полями

Після створення, такий зв'язок відображатиметься на схемі даних кількома лініями, що з'єднують відповідні поля зовнішнього ключа та основного (2.77).

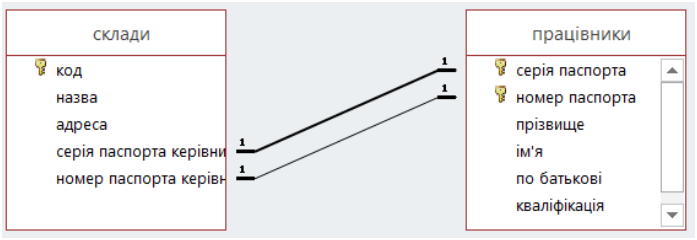


Рис. 2.77: Зв'язок за кількома полями одночасно

Вправа 53

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2017», 10-11 клас
Реалізуйте базу даних предметної області «Енергетика» в СУБД Access, використовуючи реляційну модель, подану на Рисунку 2.78.

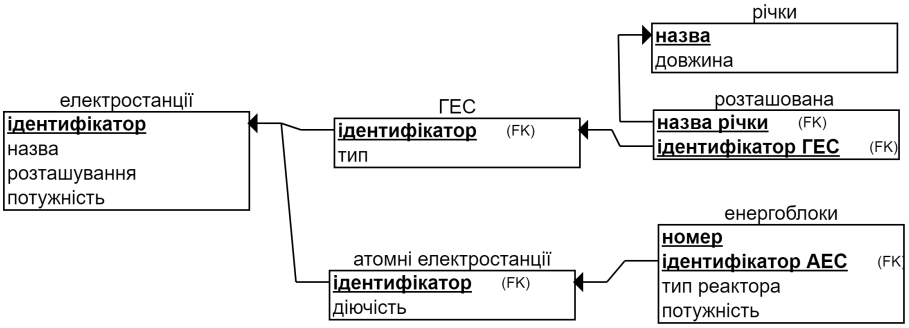


Рис. 2.78: Реляційна модель предметної області «Енергетика»

Вправа 54

ІІ етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2017», 10-11 клас
Реалізуйте базу даних предметної області «Головоломки» в СУБД Access, використовуючи реляційну модель, побудовану у Вправі 34.

Вправа 55

ІV етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2017», 10-11 клас
Реалізуйте базу даних предметної області «Натуральні числа» в СУБД Access, використовуючи реляційну модель, побудовану у Вправі 35.

Вправа 56

І Всеукраїнська учнівська Інтернет-олімпіада з інформаційних технологій, тренувальний тур І етапу
Використовуючи реляційну модель, побудовану для інформаційної системи «Школа» у Вправі 44, реалізуйте базу даних цієї предметної області в СУБД Access. Передбачте, що в один і той самий день в одному й тому самому класі на одному й тому ж уроці не може бути двох занять одночасно. Аналогічно, в один і той самий день один і той самий учитель на одному й тому ж уроці не може викладати два заняття одночасно.

Вправа 57

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2016», 10-11 клас
Використовуючи реляційну модель, побудовану для інформаційної системи «Файлова система» у Вправі 45, реалізуйте базу даних цієї предметної області в СУБД Access.

Вправа 58

Використовуючи реляційну модель, побудовану для пісенного конкурсу «Євробачення» у 2018 році у Вправі 39, реалізуйте базу даних цієї предметної області в СУБД Access.

Вправа 59

IV етап І Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи реляційну модель, побудовану для предметної області «Водні ресурси» у Вправі 46, реалізуйте базу даних цієї предметної області в СУБД Access.

Вправа 60

IV етап III Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи реляційну модель, побудовану для предметної області «Географічна карта» у Вправі 47, реалізуйте базу даних цієї предметної області в СУБД Access.

Вправа 61

IV етап IV Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи реляційну модель, побудовану для предметної області «Лабіринти» у Вправі 48, реалізуйте базу даних цієї предметної області в СУБД Access.

2.3.2 Реалізація реляційної моделі в СУБД MySQL

MySQL — вільна система управління базами даних, що наразі підтримується компанією Oracle.

Для опанування цього підрозділу книги вам знадобляться встановлені СУБД MySQL та застосунок phpMyAdmin. Обидва програмні продукти є вільним програмним забезпеченням, тобто для їх використання не потрібна ліцензія.

Для встановлення необхідного програмного забезпечення ви можете скористатися посиланням https://www.apachefriends.org/index.html для завантаження пакету програм XAMPP, який зокрема містить в своїй поставці MySQL та phpMyAdmin та налаштовує їх необхідним чином під час процесу інсталяції. Для вивчення матеріалу достатньо програмного забезпечення, яке виділено на Рисунку 2.79. Решту програм пакету ви можете встановлювати за бажанням.

Для початку роботи з сервером досить натиснути кнопки «Start» навпроти сервера Apache і бази даних MySQL (Рисунок 2.80). Запущені служби підсвічені зеленим фоном та стають активними їх кнопки *Admin*. Кнопка *Admin* для MySQL запускає phpMyAdmin – інструмент для роботи з базою даних.

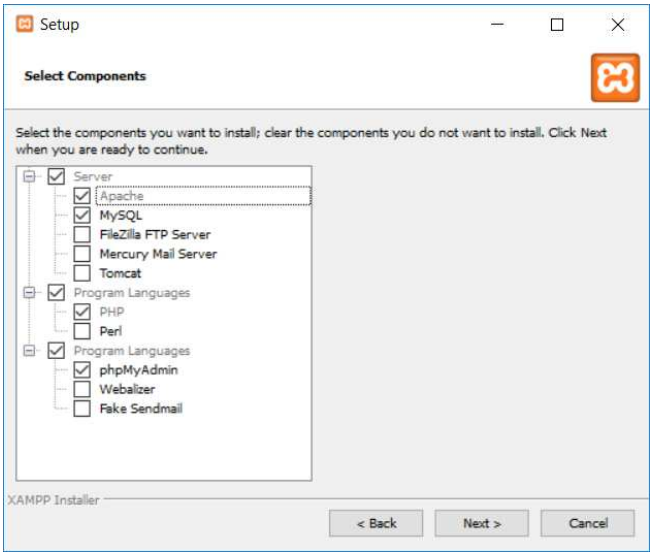


Рис. 2.79: Вікно вибору програмного забезпечення

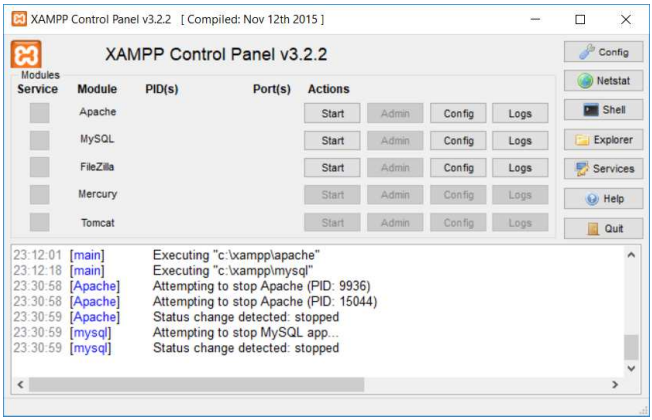


Рис. 2.80: Панель управління XAMPP

Огляд середовища

MySQL являє собою службу операційної системи, що надає можливість маніпуляції базами даних.

Система управління базами даних MySQL, на відміну від Microsoft Access, не надає засобів для створення інформаційних систем, а цілком і повністю присвячена роботі з базами даних. В рамках цього підрозділу книги ми, аналогічно до того як це зробили для Microsoft Access, докладно розглянемо принципи реалізації реляційних моделей в MySQL за допомогою застосунку phpMyAdmin — одного з найбільш популярних засобів адміністрування баз даних MySQL.

Після натиснення на кнопку *Admin* для MySQL, XAMPP відкриє для вас головну сторінку засобу phpMyAdmin (Рисунок 2.81), за допомогою якого ми реалізовуватимемо реляційні схеми різноманітних баз даних. Якщо вас не влаштовує мова засобу за промовчанням, на цій сторінці ви можете її змінити, обравши зручну для вас із розкривного списку.

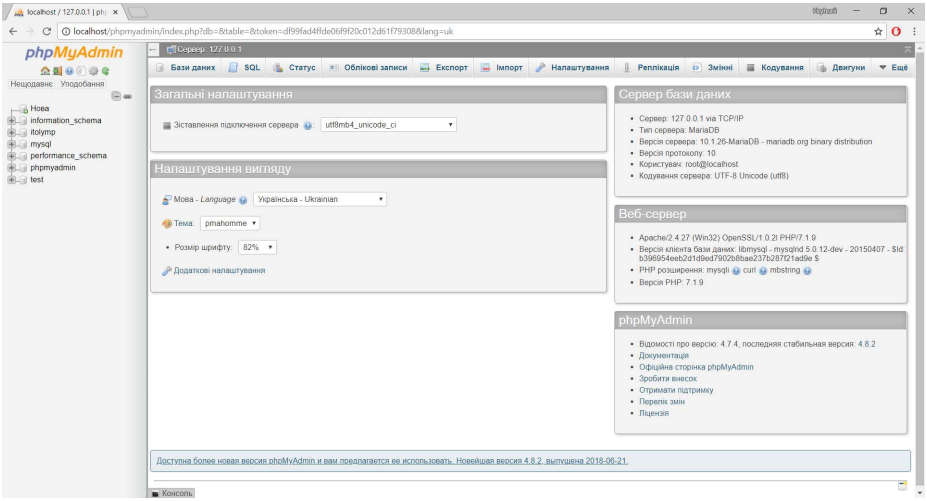


Рис. 2.81: Головна сторінка phpMyAdmin

Створення баз даних

В лівій частині головної сторінки ви можете помітити перелік встановлених на ваш комп'ютер баз даних. Аби додати нову базу даних, слід натиснути на посилання *Нова* вгорі переліку. Після цього буде відображено вікно додавання нової бази даних (Рисунок 2.82), в якому потрібно ввести назву бази даних та її кодування.

Зверніть увагу! Правилом хорошого тону в створенні баз даних є англо-

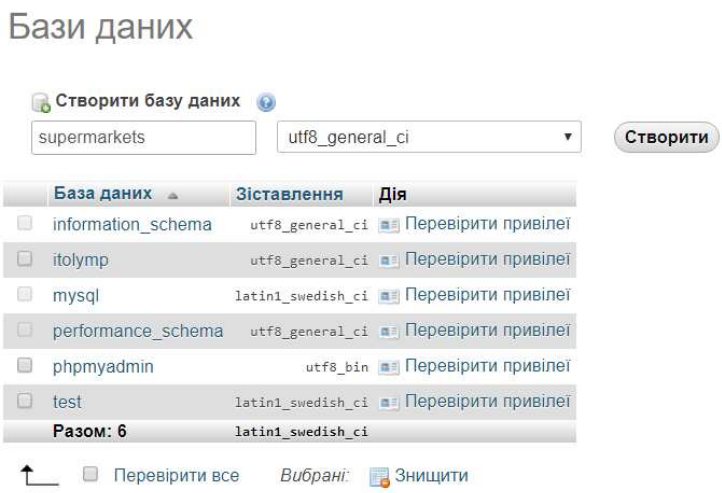


Рис. 2.82: Вікно додавання нової бази даних

мовні назви всіх їх елементів без пробілів та тире. Деякі системи управління базами даних та сторонні програмні засоби можуть некоректно реагувати на кириличні назви та назви, що містять усіляки спецсимволи, тому не рекомендуємо використовувати їх при іменуванні об'єктів, навіть якщо СУБД це дозволяє.

На даний момент назвімо нашу базу даних *supermarkets* та встановімо кодування *utf8_general_ci* та натиснемо на кнопку *Створити*.

На відміну від Microsoft Access, система не створює таблицю автоматично, але її можна легко створити у вікні, що відображається одразу після створення бази даних.

На будь-якому етапі роботи з системою ви можете перейти до будь-якої бази даних за посиланням у переліку баз даних в лівій частині будь-якої сторінки.

Робота з таблицями

Створити нову таблицю можна в будь-який момент за допомогою форми *Створити таблицю* головного меню програми (Рисунок 2.83). Для цього достатньо ввести назву майбутньої таблиці та кількість її полів, а потім натиснути кнопку *Виконати*.

Після цього буде відкрито вікно з кількістю рядків, відповідною введе-

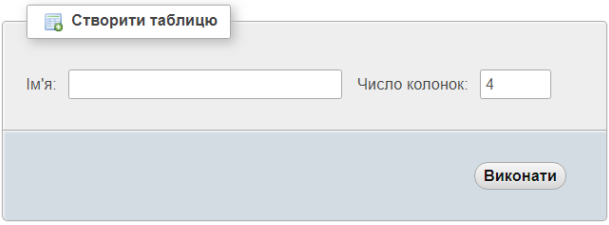


Рис. 2.83: Форма створення таблиці

ній кількості полів таблиці (Рисунок 2.84). В кожному з цих рядків подані налаштування полів таблиці, як-от в першому — їх імена, в другому — типи, в третьому — довжина, у восьмому — індекс, в дев'ятому — можливість вказати, що поле є автоматичним лічильником. Зосередимось наразі на трьох перших стовпцях.

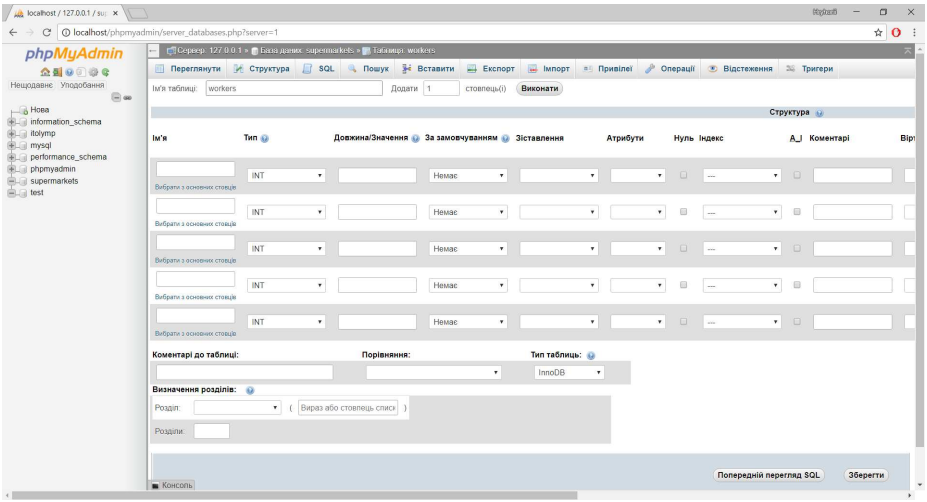


Рис. 2.84: Створення таблиці (додавання полів)

MySQL пропонує значно більший вибір типів полів, аніж Microsoft Access. Основні типи полів включають:

- *TINYINT* — призначений для збереження в полі цілого числа від -128 до 127; в якості довжини вказується максимальна кількість цифр числа;

- *SMALLINT* — призначений для збереження в полі цілого числа від -32768 до 32767; в якості довжини вказується максимальна кількість цифр числа;
- *MEDIUMINT* — призначений для збереження в полі цілого числа від -8388608 до 8388607; в якості довжини вказується максимальна кількість цифр числа;
- *INT* — призначений для збереження в полі цілого числа від -2147483648 до 2147483647; в якості довжини вказується максимальна кількість цифр числа;
- *BIGINT* — призначений для збереження в полі цілого числа від -9223372036854775808 до 9223372036854775807; в якості довжини вказується максимальна кількість цифр числа;
- *FLOAT* — призначений для збереження в полі числа з плаваючою крапкою; в якості довжини вказується два числа через кому: максимальна загальна кількість цифр числа та максимальна кількість цифр після десяткового розділювача;
- *DOUBLE* — призначений для збереження в полі більш точного числа з плаваючою крапкою; в якості довжини вказується два числа через кому: максимальна загальна кількість цифр числа та максимальна кількість цифр після десяткового розділювача;
- *DECIMAL* — призначений для збереження в полі числа як рядка з фіксованою крапкою; в якості довжини вказується два числа через кому: максимальна загальна кількість цифр числа та максимальна кількість цифр після десяткового розділювача;
- *DATE* — призначений для збереження в полі дати у форматі YYYY-MM-DD;
- *DATETIME* — призначений для збереження в полі дати та часу у форматі YYYY-MM-DD HH:MM:SS;
- *TIME* — призначений для збереження в полі часу у форматі HH:MM:SS;
- *CHAR* — призначений для збереження в полі рядка фіксованої довжини до 255 символів; в якості довжини вказується довжина рядка;
- *VARCHAR* — призначений для збереження в полі рядка змінної довжини до 255 символів; в якості довжини вказується максимальна довжина рядка;
- *TINYTEXT* — призначений для збереження в полі рядка змінної довжини до 255 символів;
- *TEXT* — призначений для збереження в полі рядка змінної довжини до 65535 символів;
- *MEDIUMTEXT* — призначений для збереження в полі рядка змінної

довжини до 16777215 символів;
- *LONGTEXT* — призначений для збереження в полі рядка змінної довжини до 4294967295 символів.

З повним переліком типів полів можна познайомитися в довідці MySQL.

Зверніть увагу! Поля типів TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT зберігають свої значення поза таблицею в інших частинах пам'яті, а тому не можуть використовуватися в ключах та індексах таблиць. Замість них може бути використане поле типів VARCHAR або CHAR.

Заповнимо форму даними про таблицю «працівники» (тут і далі — таблиця «workers») відповідно до її реляційної схеми (Рисунок 2.85).

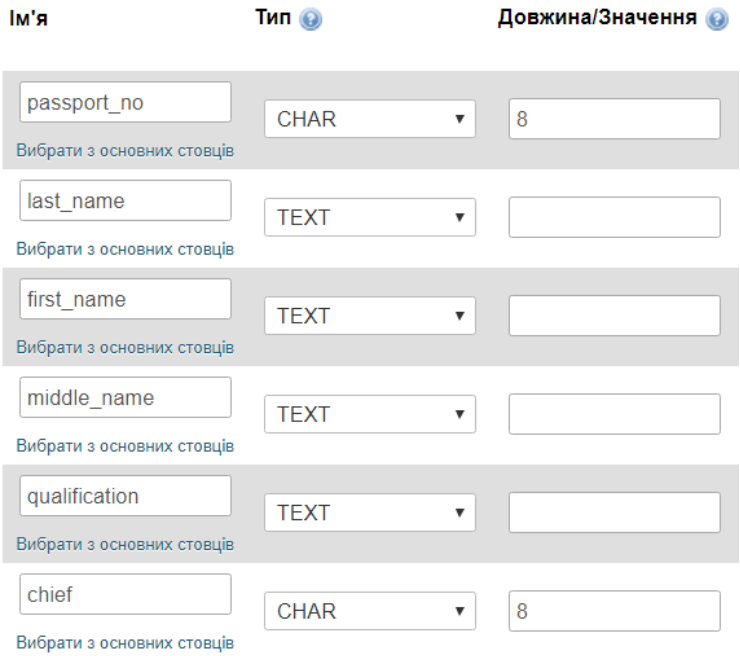


Рис. 2.85: Створення таблиці «workers»

Після натиснення на кнопку *Зберегти* система відкриє подання структури створеної таблиці (Рисунок 2.86), в якому зокрема можна здійснювати налаштування ключів та індексів.

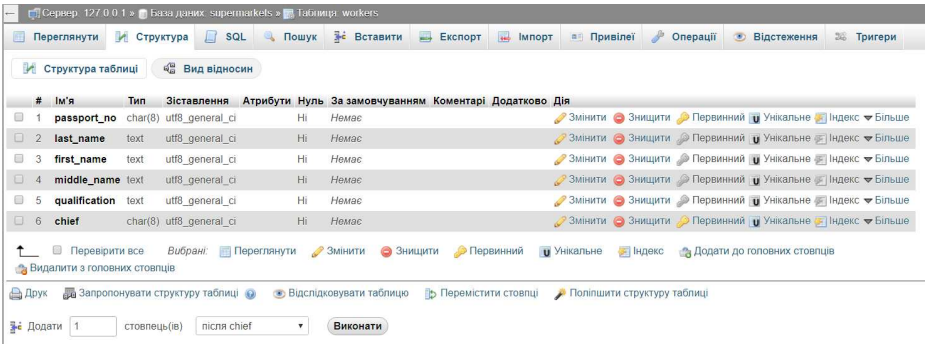


Рис. 2.86: Структура таблиці «workers»

Аби задати первинний ключ таблиці, слід відмітити прапорцями усі поля таблиці, що повинні входити до ключа, та натиснути на посилання *Первинний*, розташоване одразу під переліком полів. Після успішного встановлення ключа, поряд з усіма його полями відображатиметься піктограма ключа (Рисунок 2.87).

	#	Ім'я	Тип
☐	1	passport_no	char(8)
☐	2	last_name	text
☐	3	first_name	text
☐	4	middle_name	text
☐	5	qualification	text
☐	6	chief	char(8)

Рис. 2.87: Структура таблиці «workers»

Для створення індексу або унікального ключа (дивіться тему Індексація та унікальні поля) слід аналогічно відмітити поля прапорцями та натиснути відповідне посилання одразу під переліком полів.

Такій схемі таблиці відповідає табличне подання, подане на Рисунку 2.88. Його можна переглянути за посиланням *Переглянути* головного меню таблиці.

passport_no	last_name	first_name	middle_name	qualification	chief
AA123456	Іванов	Андрій	Сергійович	вища	*NULL*
AH564646	Андрієнко	Роман	Григорович	вища	AA123456
BP556465	Боголюбов	Борис	Арсенович	вища	AH564646
HH564127	Бобров	Семен	Іванович	1 кат.	AH564646
OH654643	Петренко	Віктор	Борисович	вища	CP486823
CP486823	Титаренко	Віктор	Леонідович	3 кат.	AA123456

Рис. 2.88: Табличне подання для таблиці «workers»

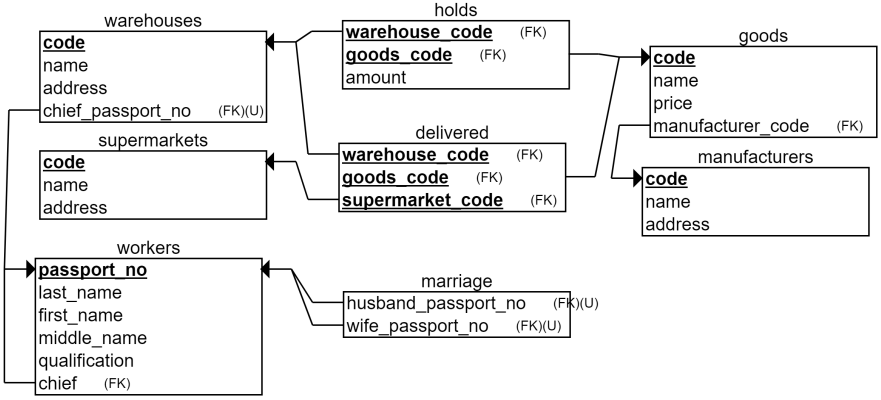


Рис. 2.89: Реляційна модель предметної області «Мережа супермаркетів»

Вправа 62

1. Реалізуйте в базі даних таблиці реляційної моделі предметної області «Мережа супермаркетів», яку ми побудували в минулому підрозділі (Рисунок 2.89).
2. Забезпечте унікальність значень полів «chief_passport_no» («номер паспорта керівника», таблиця «warehouses»), «husband_passport_no» та «wife_passport_no» («номер паспорта чоловіка» та «номер паспорта дружини» відповідно, таблиця «marriage») для створення зв'язків «один до одного»

в майбутньому.

Вправа 63

В новій базі даних створіть таблицю «entrants» з наступними обов'язковими до введення полями:

- окремі поля «last_name», «first_name», «middle_name» (довжина кожного з полів не повинна перевищувати 10 символів);
- «birthdate»;
- «test_certificate_no» — унікальне число з шести цифр;
- «email»;
- «medical_certificate» — непорожній рядок;
- «certificate_series», яке дозволяє ввести рядок з двох літер;
- «certificate_no», яке дозволяє ввести рядок з 8 цифр;
- «average_score» — число з однією або двома цифрами до десяткової коми та однією цифрою після неї.

Визначте потенційні та первинний ключі таблиці, враховуючи, що номер сертифікату ЗНО («test_certificate_no»), email, а також серія («certificate_series») разом з номером атестату («certificate_no») однозначно визначають абітурієнта в їх переліку.

Реалізація зв'язків між таблицями

Для роботи зі зв'язами MySQL пропонує окремий інструмент *Вид відносин*, розташований на вкладці *Структура* головного меню (Рисунок 2.90).

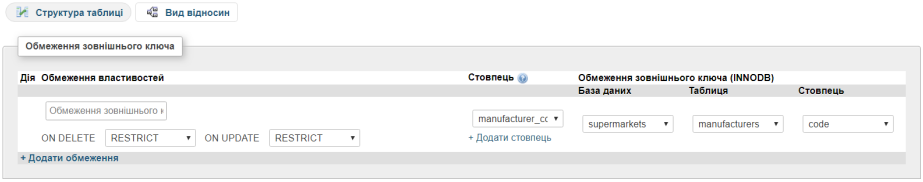


Рис. 2.90: Подання відносин таблиці «goods»

На Рисунку 2.90 зображено заповнені поля форми для успішного створення зв'язку між таблицями «goods» та «manufacturers». Для цього в по-

данні відносин для таблиці «goods» в блоці *Обмеження зовнішнього ключа* слід вказати у відповідних полях стовпець зовнішнього ключа (тут «manufacturer_code»), базу даних (тут «supermarkets»), таблицю, з якої запозичено зовнішній ключ (тут «manufacturers») та відповідний її стовпець (тут «code»). Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*. Після цього слід натиснути кнопку *Зберегти* і відповідний зв'язок буде успішно збережено.

На відміну від Microsoft Access, MySQL не має відмінностей у створенні зв'язків різного типу, тому зв'язки будь-яких множинностей та ступенів (в тому числі й унарні) створюються вищенаведеним способом.

Наприклад, для унарного зв'язку «працівник керує працівником» заповнення полів форми додавання зв'язку матиме вигляд, поданий на Рисунку 2.91.

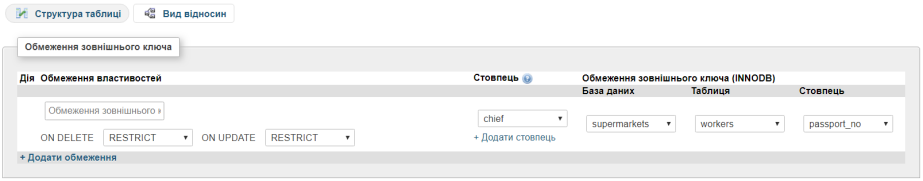


Рис. 2.91: Створення унарного зв'язку «працівник керує працівником»

Якщо є потреба у створенні зв'язку за кількома полями одночасно, достатньо натиснути на посилання *+Додати стовпець* і ввести дані про нову пару зв'язаних полів (Рисунок 2.92).

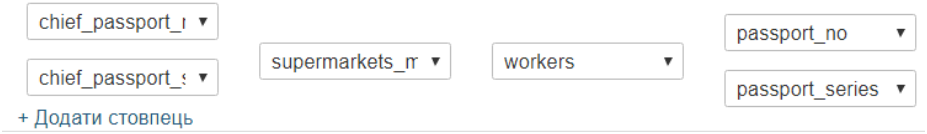


Рис. 2.92: Створення зв'язку за кількома полями

За потреби, в застосунку phpMyAdmin також можна побачити візуальну схему бази даних. Для цього потрібно натиснути на посилання потрібної бази даних та в головному меню обрати *Більше — Designer*. Після цього система відобразить графічно зв'язки між наявними в базі даних таблицями (Рисунок 2.93).

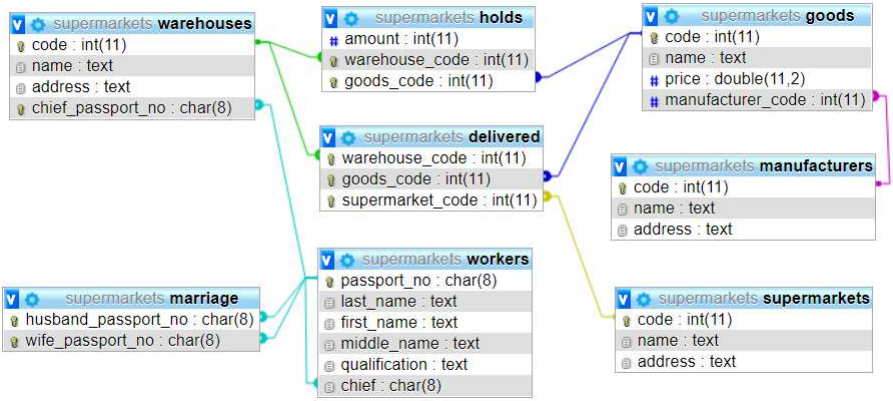


Рис. 2.93: Схема бази даних предметної області «Мережа супермаркетів»

Вправа 64

Доповніть базу даних предметної області «Мережа супермаркетів», створену у Вправі 62, зв'язами між наявними там таблицями відповідно до Рисунку 2.93.

Вправа 65

І етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас

Реалізуйте базу даних предметної області «Енергетика» в СУБД MySQL, використовуючи реляційну модель, подану на Рисунку 2.94.

Вправа 66

ІІ етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас

Реалізуйте базу даних предметної області «Головоломки» в СУБД MySQL, використовуючи реляційну модель, побудовану у Вправі 34 (перекладіть назви полів та таблиць англійською).

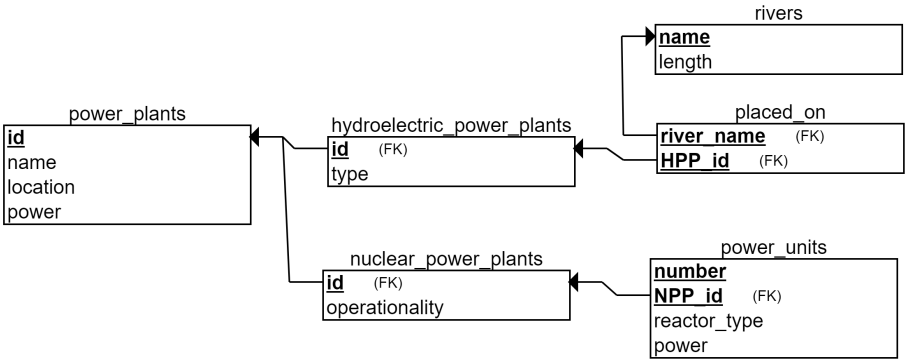


Рис. 2.94: Реляційна модель предметної області «Енергетика»

Вправа 67

IV етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас
Реалізуйте базу даних предметної області «Натуральні числа» в СУБД MySQL, використовуючи реляційну модель, побудовану у Вправі 35 (перекладіть назви полів та таблиць англійською).

Вправа 68

I Всеукраїнська учнівська Інтернет-олімпіада з інформаційних технологій, тренувальний тур I етапу
Використовуючи реляційну модель, побудовану для інформаційної системи «Школа» у Вправі 44, реалізуйте базу даних цієї предметної області в СУБД MySQL (перекладіть назви полів та таблиць англійською). Передбачте, що в один і той самий день в одному й тому самому класі на одному й тому ж уроці не може бути двох занять одночасно. Аналогічно, в один і той самий день один і той самий учитель на одному й тому ж уроці не може викладати два заняття одночасно.

Вправа 69

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2016», 10-11 клас
Використовуючи реляційну модель, побудовану для інформаційної системи «Файлова система» у Вправі 45, реалізуйте базу даних цієї предметної області в СУБД MySQL (перекладіть назви полів та таблиць англійською).

Вправа 70

Використовуючи реляційну модель, побудовану для пісенного конкурсу «Євробачення» у 2018 році у Вправі 39, реалізуйте базу даних цієї предметної області в СУБД MySQL (перекладіть назви полів та таблиць англійською).

Вправа 71

IV етап І Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи реляційну модель, побудовану для предметної області «Водні ресурси» у Вправі 46, реалізуйте базу даних цієї предметної області в СУБД MySQL.

Вправа 72

IV етап ІІІ Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи реляційну модель, побудовану для предметної області «Географічна карта» у Вправі 47, реалізуйте базу даних цієї предметної області в СУБД MySQL.

Вправа 73

IV етап IV Всеукраїнської учнівської олімпіади з інформаційних технологій
Використовуючи реляційну модель, побудовану для предметної області «Лабіринти» у Вправі 48, реалізуйте базу даних цієї предметної області в СУБД MySQL.

2.3.3 Побудова фізичної моделі предметної області

Після ґрунтовного розгляду засобів фізичного проектування реляційних баз даних, ми можемо сформулювати покроковий план, за яким побудова такої моделі повинна відбуватися.

Отож, проектування на даному рівні передбачає виконання наступних етапів:

1. **Вибір СУБД.** Система управління базами даних надає інструменти для роботи з елементами баз даних, тому вибір СУБД повинен залежати від набору засобів, які необхідні для реалізації вимог до бази даних та її супроводу.
2. **Створення бази даних.** Втілення реляційної моделі розпочинається створенням бази даних на фізичному носії засобами обраної системи управління базами даних.
3. **Створення таблиць.** На цьому кроці здійснюється відтворення таблиць з реляційної моделі зі встановленням необхідних первинних та унікальних ключів, налаштуванням полів тощо.
4. **Відтворення зв'язків.** В рамках цього кроку фізичного проектування здійснюється налаштування зв'язків з реляційної моделі в середовищі системи управління базами даних. На цьому рівні здійснюється також налаштування необхідної обробки подій (як-от оновлення чи видалення записів).
5. **Аналіз транзакцій.** На цьому кроці здійснюється визначення основних послідовностей дій над даними бази даних, як-от видобування даних, їх додавання, оновлення, видалення тощо. Аналіз транзакцій є дуже важливим для прогнозування вузьких місць у швидкодії бази даних та їх максимальне усунення, в тому числі за рахунок створення індексів.
6. **Визначення індексів.** На основі результатів попереднього кроку приймається рішення щодо створення індексів в таблицях бази даних. Слід зауважити, що індекси покращують швидкодію бази даних за

рахунок додаткової пам'яті, а тому надмірна кількість індексів може негативно впливати на розмір зайнятого базою дискового простору та, як наслідок, швидкодію видобування даних з нього.

7. **Розробка механізмів захисту даних.** На цьому кроці, за потреби, здійснюється захист бази даних в цілому, а також окремих її елементів. Зокрема, одним із методів захисту є розмежування прав доступу для різних користувачів бази даних, яке дозволяє керувати правами на редагування структури бази даних, перегляд її даних та їх модифікацію. В промислових інформаційних системах захист даних є критично важливим, адже від нього залежить коректність роботи всієї системи.

Етап фізичного проектування є завершальним в низхідному підході до проектування баз даних.

2.4 Висхідний підхід до проектування

На противагу низхідному, переважно для невеликих за розмірами баз даних використовується висхідний підхід, який передбачає спочатку ідентифікацію всіх необхідних атрибутів, а потім, на основі отриманої сукупності полів та зв'язків між ними, — вибудовування відношень (таблиць), що представляють собою сутності предметної області та зв'язки між ними. Найчастіше цей процес здійснюється за допомогою нормалізації даних.

2.4.1 Надлишковість та неузгодженість даних. Аномалії оновлення

Збереження одного й того самого факту в базі даних кілька разів в кількох місцях має очевидні негативні наслідки. По-перше, це збільшує кількість пам'яті, яку займає база даних. По-друге, це часто призводить до виникнення неузгодженості даних через складний процес їх внесення, редагування та видалення — так званих *аномалій оновлення*.

Розглянемо таблицю, зображену на Рисунку 2.95. Вважатимемо, що в мережі супермаркетів, що нами розглядається, є центральний офіс, в якому відповідальний працівник веде таку таблицю обліку товарів на складах компанії.

Розглянемо наступну можливу ситуацію. Товар «Молоко українське» було завезено також на склад «Голосіївський». Відповідальний працівник вніс дані до таблиці, але випадково помилився в ціні молока (Рисунок 2.96). Виникає неузгодженість в ціні товару «Молоко українське», скільки він коштує: 25,30 грн чи 28,30 грн?

штрих-код	назва товару	роздрібна ціна	код виробника	назва виробника	адреса виробника
8215562868	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Меж...
8215562868	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Меж...
9896514456	Хліб житній	10,30 ₴	1	Київхліб	м. Київ, вул. Меж...
6546468784	Молоко українське	28,30 ₴	2	1-й Київсь	м. Київ, вул. Жил...
6546872698	Сир голландський	231,10 ₴	2	1-й Київсь	м. Київ, вул. Жил...
8635434366	Сирок плавлений	13,20 ₴	2	1-й Київсь	м. Київ, вул. Жил...
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Гр...
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Гр...

код складу	назва складу	адреса складу	номер паспорта керівника	Прізвище	Ім'я	Кількість
1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман	29
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	30
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	14
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	25
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	13
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	18
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	20
1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман	15

Рис. 2.95: Таблиця товарів на складах мережі супермаркетів

штрих-код	назва товару	роздрібна ціна
8215562868	Хліб білий	12,80 ₴
8215562868	Хліб білий	12,80 ₴
9896514456	Хліб житній	10,30 ₴
6546468784	Молоко українське	28,30 ₴
6546872698	Сир голландський	231,10 ₴
8635434366	Сирок плавлений	13,20 ₴
2455585436	Батон білий	14,30 ₴
2455585436	Батон білий	14,30 ₴
6546468784	Молоко українське	25,30 ₴

код складу	назва складу
1	Васильківський
2	Голосіївський
2	Голосіївський
3	Ужгородський
3	Ужгородський
3	Ужгородський
2	Голосіївський
1	Васильківський
2	Голосіївський

Рис. 2.96: Таблиця товарів на складах мережі супермаркетів

Вищенаведений приклад демонструє *аномалію вставки*: неузгодженість, яка виникає під час додавання даних в таблицю.

Припустимо, що всі одиниці товару «Батон білий» були успішно продані. Таким чином, інформація про всі згадування цього товару з таблиці буде видалена. Але виробник «Кулиничі» виготовляє тільки цей товар, тому з таблиці разом з товаром зникнуть і далі про виробника, зокрема назва і адреса. А цей виробник ймовірно ще поставлятиме нашій мережі свою продукцію в майбутньому.

Цей приклад демонструє *аномалію видалення*: зникнення даних про пов'язані об'єкти при видаленні деяких записів таблиці.

Якщо ціна товару «Хліб білий» зміниться і цю зміну буде відображено тільки в одному записі (Рисунок 2.97), то виникне неузгодженість в ціні товару «Хліб білий», скільки він коштує: 15,00 грн чи 12,80 грн?

штрих-код	назва товару	роздрібна ціна
8215562868	Хліб білий	12,80 ₴
8215562868	Хліб білий	15,00 ₴
9896514456	Хліб житній	10,30 ₴
6546468784	Молоко українське	28,30 ₴
6546872698	Сир голландський	231,10 ₴
8635434366	Сирок плавлений	13,20 ₴
2455585436	Батон білий	14,30 ₴
2455585436	Батон білий	14,30 ₴

Рис. 2.97: Таблиця товарів на складах мережі супермаркетів

Вищенаведений приклад демонструє *аномалію модифікації*: неузгодженість, яка виникає під час модифікації даних таблиці.

З розглянутих вище прикладів можна переконатися, що надлишкові дані в базі даних можуть стати джерелом неузгодженостей, які в результаті згубно впливатимуть на роботу всієї інформаційної системи. Дійсно, навіть при такій організації даних можна побудувати програмне забезпечення, що працюватиме з такою базою даних, так, аби аномалії взагалі не мали місце, проте така неструктурованість даних значно ускладнює і без того нелегкий процес розробки програмних систем.

Саме тому Едгаром Коддом був запропонований процес, який би дозволив чітко структурувати реляційні дані так, щоб максимально запобігти виникненню аномалій, — процес нормалізації даних.

2.4.2 Процес нормалізації

Означення 37. *Нормалізація — це техніка отримання набору відношень (таблиць), що задовільняють певним властивостям, за вимогами до даних програмного продукту.*

Зазвичай нормалізація відбувається в кілька етапів, кожному з яких відповідає певна нормальна форма. В ході цього процесу відношення стають все менш сприятливими для виникнення аномалій.

В цій книзі ми розглянемо детально процес нормалізації лише до третьої нормальної форми, чого достатньо для уникнення описаних вище аномалій оновлення. Якщо читач хоче познайомитися з нормальними формами вищих порядків, він може звернутися до книги Томаса Коннолі та Керолін Бегг «Бази даних. Проектування, реалізація та супровід. Теорія і практика», де цей процес описаний більш детально.

Перша нормальна форма

Входом процесу нормалізації є таблиця з визначеним ключем (мінімальним набором полів, що однозначно визначає рядок цієї таблиці).

штрих-код	назва товару	роздрібна ціна	код виробника	назва виробника	адреса виробника	код складу
8215562968	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Ме	1 2
9896514456	Хліб житній	10,30 ₴	1	Київхліб	м. Київ, вул. Ме	2
6546568784	Молоко українське	28,30 ₴	2	1-й Київськи	м. Київ, вул. Жи	3
6546872698	Сир голландський	231,10 ₴	2	1-й Київськи	м. Київ, вул. Жи	3
8635343663	Сирок плавлений	13,20 ₴	2	1-й Київськи	м. Київ, вул. Жи	3
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Г	2 1

код складу	назва складу	адреса складу	номер паспорта керівника	прізвище	ім'я	кількість
1	Васильківський	м. Київ, вул. Васильківська	АН564646	Андрієнко	Роман	29
2	Голосіївський	м. Київ, вул. Ломоносова	СР486823	Титаренко	Віктор	30
2	Голосіївський	м. Київ, вул. Ломоносова	СР486823	Титаренко	Віктор	14
3	Ужгородський	м. Ужгород, вул. 8 березня	АА123456	Іванова	Наталія	25
3	Ужгородський	м. Ужгород, вул. 8 березня	АА123456	Іванова	Наталія	13
3	Ужгородський	м. Ужгород, вул. 8 березня	АА123456	Іванова	Наталія	18
2	Голосіївський	м. Київ, вул. Ломоносова	СР486823	Титаренко	Віктор	20
1	Васильківський	м. Київ, вул. Васильківська	АН564646	Андрієнко	Роман	15

Рис. 2.98: Ненормалізована таблиця «товари»

Розглянемо таблицю «товари», подану на Рисунку 2.98. Очевидно, що її ключем буде поле «штрих-код». Така таблиця може бути з легкістю представлена в табличному чи текстовому процесорі, проте зовсім не адаптована до збереження в реляційній базі даних через об'єднання комірок. Кажуть, що така таблиця перебуває у ненормалізованій формі.

Означення 38. *Ненормалізована форма — це таблиця, одне або декілька полів якої мають декілька значень в один момент часу.*

В нашому випадку поля «код складу», «назва складу», «адреса складу», «номер паспорта керівника», «прізвище», «ім'я», «кількість» є багатозначними, бо в даному конкретному прикладі для двох категорій товарів мають по два значення.

Означення 39. *Перша нормальна форма — це таблиця, що не має повторюваних груп полів та кожне з полів якої має рівно одне значення в один момент часу.*

Загалом існує два підходи до усунення багатозначності полів ненормалізованої таблиці:

1. Прибирання «об'єднаних комірок» шляхом розбиття рядків з багатозначними полями та дублювання неповторюваних даних.
2. Винесення багатозначних полів разом з копією ключа поточної таблиці в окрему таблицю.

При застосуванні першого підходу до нашого прикладу ненормалізованої таблиці, ми отримаємо таблицю, подану на Рисунку 2.99.

В цій таблиці було прибрано об'єднання комірок, а дані цих об'єднаних комірок продубльовано на усі відповідні їм рядки таблиці.

Другий підхід призведе до утворення двох таблиць, які зображено на Рисунках 2.100–2.101.

В цій книзі ми притримуватимемось першого з цих підходів. Пізніше ми переконаємось, що ці підходи цілком еквівалентні. Отже, результатом цього кроку для нашого прикладу є таблиця «товари» у першій нормальній формі, подана на Рисунку 2.99. Ця таблиця має два потенційні ключі:

- з полів «штрих-код» та «код складу»;
- з полів «штрих-код» та «номер паспорта керівника».

Нагадуємо, що в цій предметній області працівник може керувати не більш, ніж одним складом.

штрих-код	назва товару	роздрібна ціна	код виробника	назва виробника	адреса виробника
8215562868	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Меж...
8215562868	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Меж...
9896514456	Хліб житній	10,30 ₴	1	Київхліб	м. Київ, вул. Меж...
6546468784	Молоко українське	28,30 ₴	2	1-й Київсь	м. Київ, вул. Жил...
6546872698	Сир голландський	231,10 ₴	2	1-й Київсь	м. Київ, вул. Жил...
8635434366	Сирок плавлений	13,20 ₴	2	1-й Київсь	м. Київ, вул. Жил...
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Гр...
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Гр...

код складу	назва складу	адреса складу	номер паспорта керівника	Прізвище	Ім'я	Кількість
1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман	29
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	30
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	14
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	25
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	13
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	18
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	20
1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман	15

Рис. 2.99: Таблиця «товари» у першій нормальній формі

штрих-код	назва товару	роздрібна ціна	код виробника	назва виробника	адреса виробника
8215562868	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Меж...
9896514456	Хліб житній	10,30 ₴	1	Київхліб	м. Київ, вул. Меж...
6546468784	Молоко українське	28,30 ₴	2	1-й Київський молокозавод	м. Київ, вул. Жил...
6546872698	Сир голландський	231,10 ₴	2	1-й Київський молокозавод	м. Київ, вул. Жил...
8635434366	Сирок плавлений	13,20 ₴	2	1-й Київський молокозавод	м. Київ, вул. Жил...
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Гр...

Рис. 2.100: Таблиця «товари» у першій нормальній формі

штрих-код	код складу	назва складу	адреса складу	номер паспорта керівника	Прізвище	Ім'я	Кількість
8215562868	1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман	29
8215562868	2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	30
9896514456	2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	14
6546468784	3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	25
6546872698	3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	13
8635434366	3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія	18
2455585436	2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор	20
2455585436	1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман	15

Рис. 2.101: Таблиця «зберігається» у першій нормальній формі

Функціональна залежність

Для розгляду наступних нормальних форм нам потрібно познайомитися з поняттям функціональної залежності.

Означення 40. *Набір полів B функціонально залежить від набору полів A (A → B), якщо кожному значенню набору A відповідає рівно одне значення набору полів B.*

Наприклад, поле «назва товару» функціонально залежить від набору полів «штрих-код» та «код складу», оскільки для кожного значення цієї пари полів назва товару визначається однозначно. Цей факт можна записати так:

«штрих-код», «код складу» → «назва товару»

Означення 41. *Набір полів B перебуває у повній функціональній залежності від набору полів A, якщо набір B функціонально залежить від набору A, але не залежить функціонально від будь-якої власної підмножини полів набору A.*

Іншими словами, набір полів B перебуває у повній функціональній залежності від набору полів A, якщо $A \to B$ і відкидання будь-якого з полів A призводить до зникнення функціональної залежності.

Наприклад, поле «назва товару» перебуває у повній функціональній залежності від поля «штрих-код», оскільки для кожного значення цієї пари полів назва товару визначається однозначно, а більше зменшити такий набір не можна. З іншого боку, розглянута раніше функціональна залежність повною не є, оскільки є підмножина набору («штрих-код», «код складу»), від якої також функціонально залежить поле «назва товару».

Перелік повних функціональних залежностей таблиці «товари» з Рисунку 2.99 є наступним:

1. «штрих-код» → «назва товару», «роздрібна ціна», «код виробника», «назва виробника», «адреса виробника»;
2. «код складу» → «назва складу», «адреса складу», «номер паспорта керівника», «прізвище», «ім'я»;
3. «номер паспорта керівника» → «код складу», «назва складу», «адреса складу», «прізвище», «ім'я»;
4. «код виробника» → «назва виробника», «адреса виробника»;
5. «штрих-код», «код складу» → «кількість»;
6. «штрих-код», «номер паспорта керівника» → «кількість».

Будь-який набір полів перебуває у повній функціональній залежності від самого себе. Така залежність називається *тривіальною*. Тут і далі ми опускатимемо такі залежності, оскільки вони не впливають на процес нормалізації.

Означення 42. *Набір полів C перебуває у транзитивній залежності від набору полів A, якщо існує такий набір полів B, що $A \to B$ та $B \to C$ і $B \not\to A, C \not\to A$.*

Наприклад, в транзитивній залежності від поля «штрих-код» перебуває поле «назва виробника», оскільки:

«штрих-код» → «код виробника»

«код виробника» → «назва виробника»

Друга нормальна форма

Означення 43. *Друга нормальна форма — це таблиця у першій нормальній формі, кожне з неключових полів якої перебуває у повній функціональній залежності від кожного потенційного ключа.*

Наша таблиця має два потенційні ключі:

- з полів «штрих-код» та «код складу»;
- з полів «штрих-код» та «номер паспорта керівника».

З визначеного вище ми бачимо, що у повній функціональній залежності від обох потенційних ключів перебуває лише поле «кількість», решта ж полів залежить від частин ключа, але не від всього ключа.

Для приведення таблиці до другої нормальної форми слід всі повні функціональні залежності від частин ключа винести в окремі таблиці разом з копією їх лівої частини.

В нашому випадку ми маємо справу з трьома повними залежностями від частин ключа, дві з яких складаються з ідентичного набору полів:

1. «штрих-код» → «назва товару», «роздрібна ціна», «код виробника», «назва виробника», «адреса виробника»;
2. «код складу» → «назва складу», «адреса складу», «номер паспорта керівника», «прізвище», «ім'я»;
3. «номер паспорта керівника» → «код складу», «назва складу», «адреса складу», «прізвище», «ім'я».

Отже, ми повинні створити дві нові таблиці, кожна з яких містить всі поля відповідної їй залежності з вищенаведених. З метою кращої змістовно-

сті третю функціональну залежність ми наразі опустимо, оскільки наявна друга залежність з тим же набором полів.

На цьому етапі ми отримаємо три пов'язані таблиці у другій нормальній формі, зображені на Рисунках 2.102–2.104:

- таблицю «зберігається» з ключем «штрих-код» та «код складу»;
- таблицю «склади» з двома потенційними ключами «код складу» та «номер паспорта керівника»;
- таблицю «товари» з ключем «штрих-код».

штрих-код	код складу	Кількість
8215562868	1	29
8215562868	2	30
9896514456	2	14
6546468784	3	25
6546872698	3	13
8635434366	3	18
2455585436	2	20
2455585436	1	15

Рис. 2.102: Таблиця «зберігається» у другій нормальній формі

код складу	назва складу	адреса складу	номер паспорта керівника	Прізвище	Ім'я
1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія

Рис. 2.103: Таблиця «склади» у другій нормальній формі

штрих-код	назва товару	роздрібна ціна	код виробника	назва виробника	адреса виробника
8215562868	Хліб білий	12,80 ₴	1	Київхліб	м. Київ, вул. Меж...
9896514456	Хліб житній	10,30 ₴	1	Київхліб	м. Київ, вул. Меж...
6546468784	Молоко українське	28,30 ₴	2	1-й Київський молокозавод	м. Київ, вул. Жил...
6546872698	Сир голландський	231,10 ₴	2	1-й Київський молокозавод	м. Київ, вул. Жил...
8635434366	Сирок плавлений	13,20 ₴	2	1-й Київський молокозавод	м. Київ, вул. Жил...
2455585436	Батон білий	14,30 ₴	3	Кулиничі	м. Харків, вул. Гр...

Рис. 2.104: Таблиця «товари» у другій нормальній формі

Реляційна модель бази даних на даному етапі матиме вигляд, зображений на Рисунку 2.105.

Рис. 2.105: Реляційна схема бази даних

Слід зазначити, що реляційна модель поступово стає схожою на ту, яку ми побудували в ході низхідного підходу. Дійсно, результат цих двох підходів в переважній більшості випадків повинен бути ідентичним — це власне і є способом перевірки коректності побудови.

Третя нормальна форма

Означення 44. *Третя нормальна форма — це таблиця у другій нормальній формі, жодне з неключових полів якої не перебуває у транзитивній залежності від жодного з потенційних ключів.*

Очевидно, що таблиця «зберігається» вже перебуває в третій нормальній формі.

Таблиця «склади» не містить транзитивних залежностей від потенційних ключів. Функціональна залежність «код складу» → «номер паспорта керівника» → «прізвище», «ім'я» не є транзитивною, оскільки «номер паспорта керівника» → «код складу».

Таблиця «товари» містить одну транзитивну залежність від ключа: «штрих-код» → «код виробника» → «назва виробника», «адреса виробника»

Отже, здійснювати перетворення ми будемо тільки над таблицею «товари».

Для приведення таблиці до третьої нормальної форми слід позбутися транзитивних залежностей. Для цього потрібно винести поля, що транзитивно залежать від ключа, в окрему таблицю, разом з копією поля, через яке відбувається ця транзитивна залежність.

Наприклад, залежність полів «назва виробника», «адреса виробника» від ключа «штрих-код» таблиці «товари» відбувається через поле «код виробни-

ка», тому поля «назва виробника», «адреса виробника», разом з копією поля «код виробника», формують нову таблицю «виробники» (Рисунок 2.106).

Таблиця «товари» набуває вигляду, поданого на Рисунку 2.107.

код виробника	назва виробника	адреса виробника
1	Київхліб	м. Київ, вул. Меж...
2	1-й Київський молокозавод	м. Київ, вул. Жил...
3	Кулиничі	м. Харків, вул. Гр...

Рис. 2.106: Таблиця «виробники» у третій нормальній формі

штрих-код	назва товару	роздрібна ціна	код виробника
8215562868	Хліб білий	12,80 ₴	1
9896514456	Хліб житній	10,30 ₴	1
6546468784	Молоко українське	28,30 ₴	2
6546872698	Сир голландський	231,10 ₴	2
8635434366	Сирок плавлений	13,20 ₴	2
2455585436	Батон білий	14,30 ₴	3

Рис. 2.107: Таблиця «товари» у третій нормальній формі

Реляційна модель бази даних, що відповідає третій нормальній формі всіх таблиць, зображена на Рисунку 2.108.

Рис. 2.108: Реляційна схема бази даних

Як ви можете помітити на даному етапі, результат процесу нормалізації дещо відрізняється від реляційної моделі, яка мала місце раніше, на логічному етапі низхідного підходу до проектування цієї бази даних. Річ в тому,

що в поточних таблицях відсутні дані про працівників, що не є керівниками складів, а тому і немає підстав для виокремлення їх в окрему таблицю «працівники». Коли ми розглядали методи низхідного підходу, ми визначали, що не обов'язково всі працівники залучені до керування складами, тому там таблиця «працівники» була виокремлена.

Якщо розглянути таблицю «склади», подану на Рисунку 2.109, де для працівників, що не є керівниками, характеристики складу невизначені (NULL), поле «код складу» більше не буде потенційним ключем таблиці.

код складу	назва складу	адреса складу	номер паспорта керівника	Прізвище	Ім'я
1	Васильківський	м. Київ, вул. Вас...	АН564646	Андрієнко	Роман
2	Голосіївський	м. Київ, вул. Ло...	СР486823	Титаренко	Віктор
3	Ужгородський	м. Ужгород, вул...	АА123456	Іванова	Наталія
NULL	*NULL*	*NULL*	ВР556465	Боголюбов	Борис
NULL	*NULL*	*NULL*	НН564127	Бобров	Семен
NULL	*NULL*	*NULL*	ОН654643	Петренко	Віктор

Рис. 2.109: Таблиця «склади» з усіма працівниками

Звідси існуватиме транзитивна залежність «номер паспорта керівника» → «код складу» → «назва складу», «адреса складу» і таблиця «склади» підлягатиме розділенню.

В такому випадку третя нормальна форма таблиць «склади» та «працівники» матиме вигляд, поданий на Рисунку 2.110.

код складу	назва складу	адреса складу	номер паспорта керівника
1	Васильківський	м. Київ, вул. Вас...	АН564646
2	Голосіївський	м. Київ, вул. Ло...	СР486823
3	Ужгородський	м. Ужгород, вул...	АА123456

а)

номер паспорта керівника	Прізвище	Ім'я
АН564646	Андрієнко	Роман
СР486823	Титаренко	Віктор
АА123456	Іванова	Наталія
ВР556465	Боголюбов	Борис
НН564127	Бобров	Семен
ОН654643	Петренко	Віктор

б)

Рис. 2.110: Таблиці (а) «склади» та (б) «працівники» у третій нормальній формі

Для цього випадку реляційна схема (Рисунок 2.111) повністю відповіда-

тиме тій, яка була побудована для бази даних в ході логічного етапу низхідного проектування, за винятком кількох випущених полів.

Рис. 2.111: Реляційна схема бази даних

Примітка. Насправді, процес нормалізації незастосовний до таблиць, що містять значення NULL, оскільки до таких таблиць в умовах невизначеності не застосовне поняття функціональної залежності в звичному його розумінні. В викладеному вище матеріалі нормалізацію було застосовано з припущенням, що решта працівників є керівниками деякого абстрактного складу NULL, а далі просто опущено інформацію про їх керування цим складом. Це можливо з огляду на те, що в даному випадку NULL позначає відсутність значення.

Вправа 74

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2017», 10-11 клас

База даних предметної області «Енергетика» представлена таблицями, зображеними на Рисунку 2.112.

Вважайте, що тип реактора не обов'язково однозначно визначає його потужність, а назва електростанції не обов'язково унікальна.

1. Наведіть приклади аномалій вставки, видалення та модифікації для наведених таблиць.
2. Наведіть повні функціональні залежності для цих таблиць.
3. Приведіть таблицю до третьої нормальної форми. Побудуйте логічну модель отриманої бази даних.
4. Яких змін зазнає процес нормалізації таблиці «атомні електростанції», якщо тип реактора однозначно визначатиме його потужність?

ідентифікатор електростанції	назва електростанції	розташування	потужність	тип	назва річки	довжина
1	Канівська ГЕС	Канів	444	руслово-греблева	Дніпро	2201
2	Київська ГАЕС	Нові Петрівці	235,5	гідроакумулююча	Дніпро	2201
3	Дністровська ГАЕС	Розкопинці	324	гідроакумулююча	Дністер	1362
4	Теребле-Ріцька ГЕС	Хустський район	27	деріваційна	Теребля	91
					Ріка	92

ідентифікатор електростанції	назва електростанції	розташування	потужність	діючість	номер реактора	тип реактора	потужність
1	Чорнобильська АЕС	Прип'ять	3800	ні	1	РБМК-1000	1000
					2	РБМК-1000	1000
					3	РБМК-1000	1000
					4	РБМК-1000	1000
2	Хмельницька АЕС	Нетішин	2000	так	1	ВВЕР-1000	1000
					2	ВВЕР-1000	1000
3	Рівненська АЕС	Вараш	2835	так	1	ВВЕР-440	440
					2	ВВЕР-440	440
					3	ВВЕР-1000	1000
					4	ВВЕР-1000	1000

Рис. 2.112: Таблиці «гідроелектростанції» та «атомні електростанції»

Вправа 75

Переклад з англійської мови, з матеріалів Університету Теннессі
Таблиця на Рисунку 2.113 демонструє типове резервування номеру у готелі мережі «XYZ». Ви можете зробити наступні припущення:

- номер готелю однозначно ідентифікує назву готеля та його поштовий індекс;
- поштовий індекс однозначно визначає місто як для готеля, так і для гостя;
- номер гостя однозначно визначає його ім'я та поштовий індекс;
- номер кімнати і номер готелю однозначно визначають тип кімнати та її ціну;
- гість не має резервувань, що перетинаються в часі;
- одну кімнату на один і той самий день може зарезервувати лише один гість.

1. Наведіть приклади аномалій вставки, видалення та модифікації для наведеної таблиці.
2. Наведіть повні функціональні залежності для цієї таблиці.

3. Приведіть таблицю до третьої нормальної форми. Побудуйте логічну модель отриманої бази даних.

Номер готелю	Номер гостя	Назва готелю	Місто розташування готелю	Поштовий індекс готелю	Ім'я гостя	Місто проживання гостя
3	232	Хілтон	Сан-Дієго	83835	Brad VZ	Ноксвілл

Поштовий індекс гостя	Дата заселення	Дата виселення	Номер кімнати	Тип кімнати	Ціна кімнати
37996	22.06.2018	27.06.2018	635	Королівський	89,99

Рис. 2.113: Типове резервування номеру

Вправа 76

Переклад з англійської мови
Ви — дизайнер бази даних та розробник для магазину пончиків «Donuts-R-Us», який бажає створити застосунок для смартфонів, де клієнти зможуть купувати пончики.

1. Використовуючи форму замовлення клієнта (Рисунок 2.114) побудуйте відповідну ненормалізовану таблицю.
2. Наведіть приклади аномалій вставки, видалення та модифікації для побудованої таблиці.
3. Приведіть таблицю до першої нормальної форми.
4. Наведіть повні функціональні залежності для одержаної таблиці (таблиць).
5. Приведіть таблицю (таблиці) до третьої нормальної форми. Побудуйте логічну модель отриманої бази даних.

Замовлення клієнта

Donuts-R-Us

You eat them fresh, we bake them fresh.

Дата: 22 червня, 2018

Номер замовлення: [1]

Номер клієнта [1]

Клієнт: **[Ім'я] [Прізвище]**
[Вулиця] [Будинок] [Квартира]
[Місто, Індекс]
[Домашній телефон] [Мобільний телефон] [Інші телефони]

Кількість	ID	Назва пончика	Опис	Ціна	Сума
1	1	Звичайний	Звичайний пончик	$1.50	$1.50
5	2	Глазурований	Глазурований пончик	$1.75	$8.75
12	3	З корицею	Пончик з корицею	$1.75	$21.00
3	4	Шоколадний	Шоколадний пончик	$1.75	$5.25
4	5	З присипкою	Пончик з присипкою	$1.75	$7.00
5	6	Без глютену	Пончик без глютену	$2.00	$10.00
				Всього	$53.50
				Податок	10%
				Total	**$58.85**

Спеціальні побажання:

Будь ласка, додайте тарілки та серветки.

Рис. 2.114: Форма замовлення клієнта

2.5 Відповіді та вказівки до розв'язання вправ

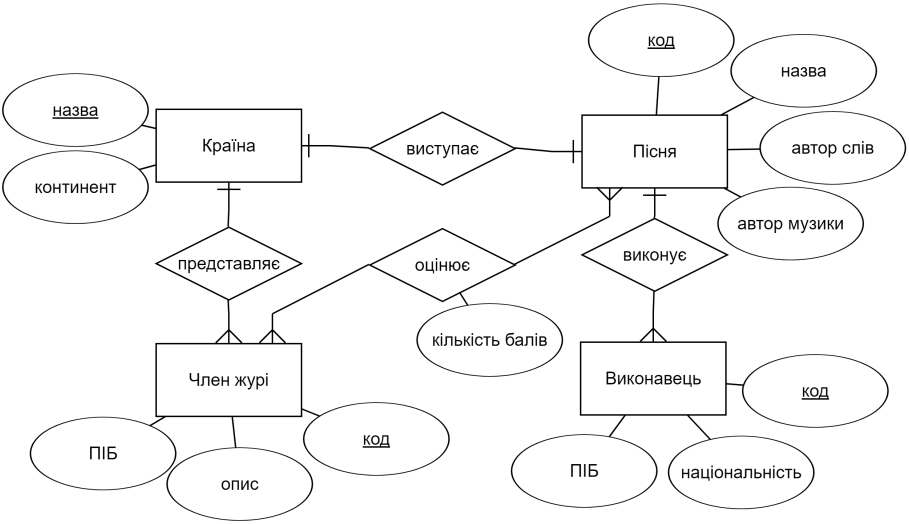
№8.

1.
   - «вчитель навчає клас» — «багато до багатьох»;
   - «чоловік одружений з жінкою» — «один до одного»;
   - «студент отримав результат оцінювання з предмету» — «багато до багатьох»;
   - «мати народила дитину» — «один до багатьох»;
   - «місто належить до країни» — «один до багатьох»;
   - «гвіздок забито в дошку» — «багато до багатьох» (адже гвіздком можна з'єднати разом дві дошки).
2. ER-моделі зображено на рисунку:

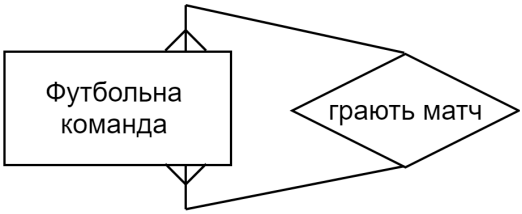
№9. В якості прикладів можна навести наступні зв'язки предметної області «Євробачення»:

- «країна виступає з піснею» — «один до одного»;
- «виконавець виконує пісню», «член журі представляє країну» — «один до багатьох»;
- «член журі оцінює пісню» — «багато до багатьох».

Відповідну ER-модель зображено на рисунку:



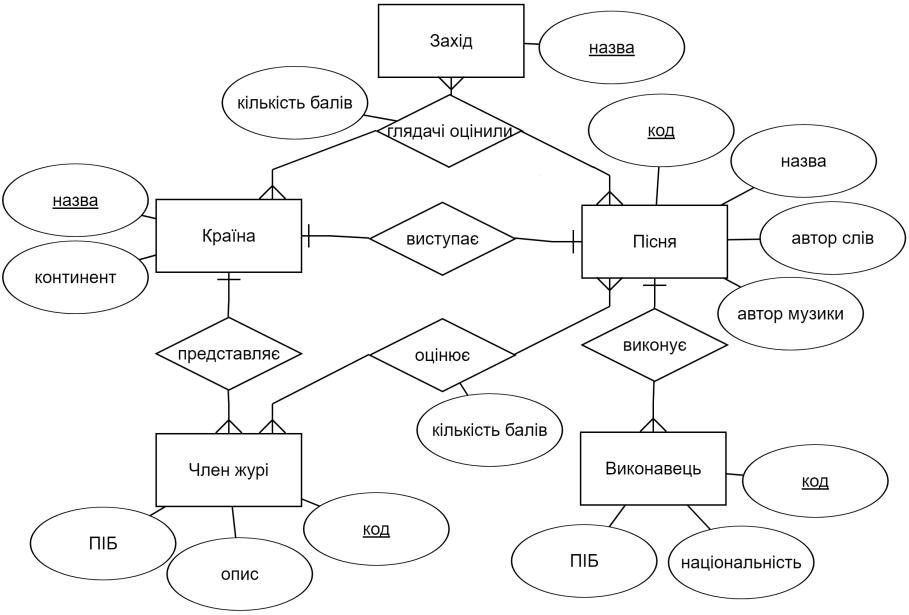
№10. Відповідну ER-модель зображено на рисунку:



Футбольна команда під час Ліги Чемпіонів зазвичай грає більш, ніж з однією командою, а отже множинність — «багато до багатьох».

№11. Прикладом тернарного зв'язку предметної області «Євробачення» є зв'язок «глядачі країни присудили пісні в певному заході (фінал чи півфінал) певну кількість балів».

Відповідну ER-модель зображено на рисунку:



№12. Формулювання «супермаркет є приміщенням» та «склад є приміщенням» представляють завжди істинні твердження: супермаркет завжди є приміщенням, склад завжди є приміщенням. Саме в цій обов'язковості й полягає суть зв'язку типу «загальний вид — різновид».

№13. Розглянемо питання, які використовуються для визначення множинності зв'язку:

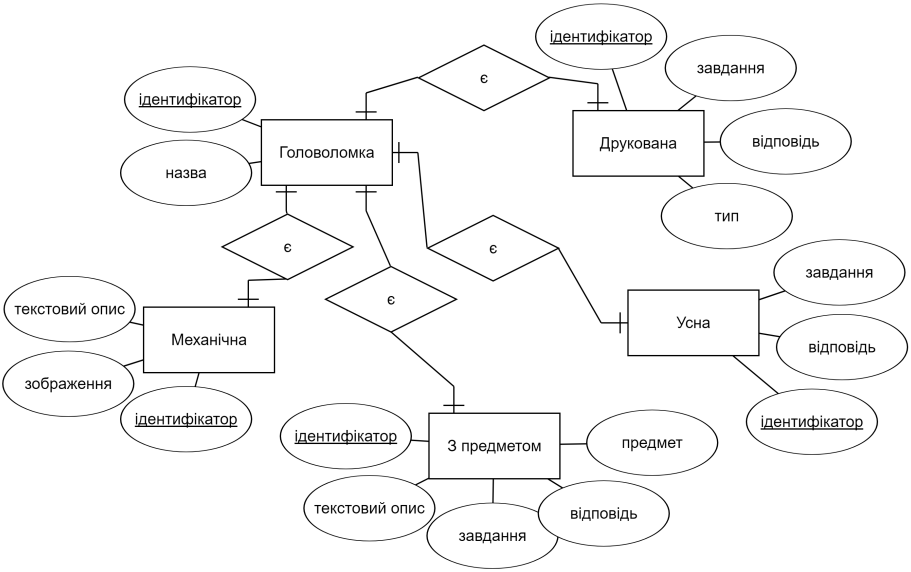
- один об'єкт сутності-різновиду скількома об'єктами сутності-загального виду може бути? — рівно одним;
- один об'єктом сутності-загального виду скількома об'єктами сутності-різновиду може бути? — не більш, ніж одним.

Отже, множинність будь-якого зв'язку типу «загальний вид — різновид» — «один до одного».

№14. Сутностями цієї предметної області є «Головоломка» (атрибути *«ідентифікатор»*, «назва»), «Друкована» (атрибути *«ідентифікатор»*, «завдання», «відповідь», «тип»), «Усна» (атрибути *«ідентифікатор»*, «завдання», «відповідь»), «Механічна» (атрибути *«ідентифікатор»*, «текстовий опис», «зображення») та «З предметом» (атрибути *«ідентифікатор»*, «на-

зва», «текстовий опис», «завдання», «відповідь», «предмет»). Ключі сутностей виділені жирним в переліку їх атрибутів і є очевидними. Слід зазначити, що в сутностях «Друкована», «Усна», «Механічна» та «З предметом» для формування ключа було запозичено ключ пов'язаної сутності «Головоломка».

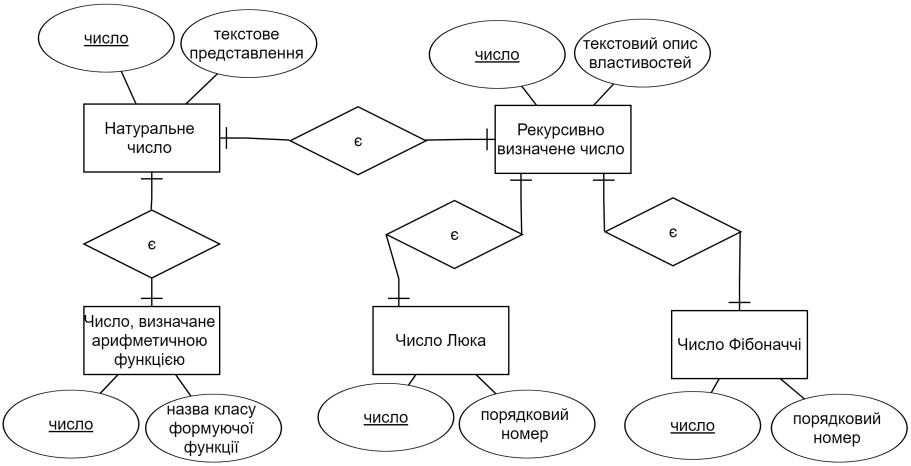
В цій предметній області всі чотири зв'язки — типу «загальний вид — різновид». З їх допомогою реалізовується класифікація гомоволомок. Відповідну ER-модель зображено на рисунку:



№15. Сутностями цієї предметної області є «Натуральне число» (атрибути *«число»*, «текстове представлення»), «Рекурсивно визначене число» (атрибути *«число»*, «текстовий опис властивостей»), «Число, визначене арифметичною функцією» (атрибути *«число»*, «назва класу формуючої функції»), «Число Люка» (атрибути *«число»*, «порядковий номер») та «Число Фібоначчі» (атрибути *«число»*, «порядковий номер»). Ключі сутностей виділені жирним в переліку їх атрибутів і є очевидними. Слід зазначити, що в сутностях «Рекурсивно визначене число», «Число, визначене арифметичною функцією», «Число Люка» та «Число Фібоначчі» для формування ключа було запозичено ключ пов'язаної сутності «Натуральне число».

В цій предметній області всі чотири зв'язки — типу «загальний вид — різновид». З їх допомогою реалізовується класифікація натуральних чисел.

Відповідну ER-модель зображено на рисунку:



№16. Ні. Оскільки за описом предметної області на річках розташовані тільки гідроелектростанції, а енергоблоки мають тільки атомні електростанції, така модель не буде цілком відображати описану предметну область.

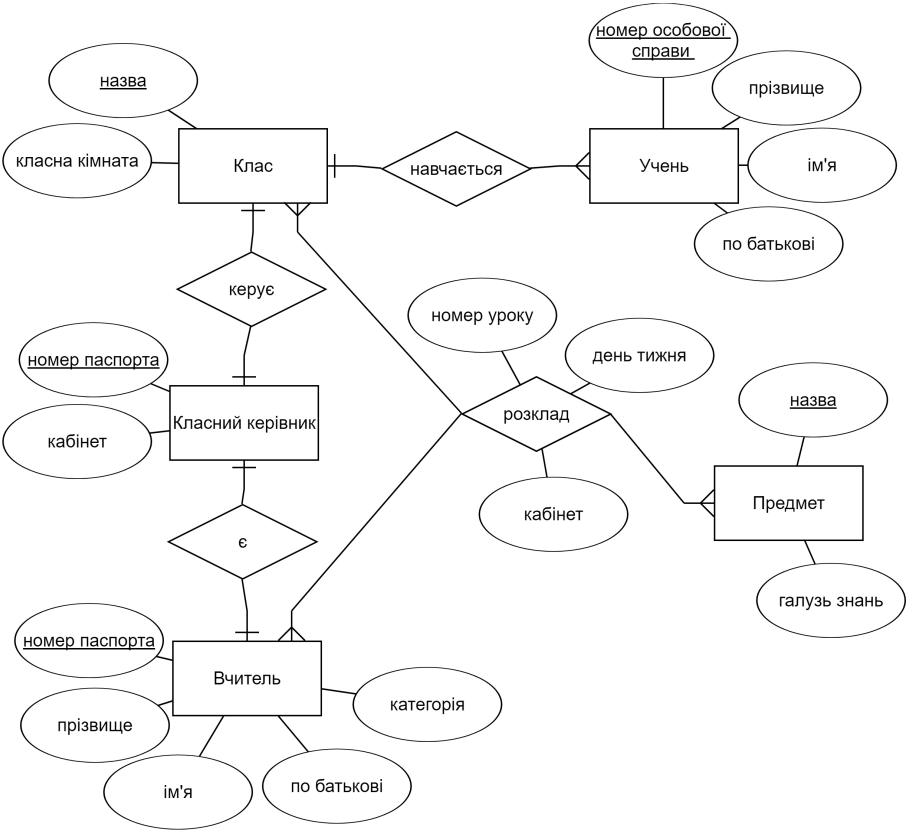
№17.

1. Як зазначено в матеріалах розділу, сутності в текстовому описі предметної області зазвичай представлені іменниками. Сутностями цієї предметної області є: «Клас» (атрибути *«назва»*, «класна кімната»), «Учень» (атрибути *«номер особової справи»*, «прізвище», «ім'я», «по батькові»), «Предмет» (атрибути *«назва»*, «галузь знань»), «Вчитель» (атрибути *«номер паспорта»*, «прізвище», «ім'я», «по батькові», «категорія»), «Класний керівник» (атрибути *«номер паспорта»*, «кабінет»). Ключі сутностей виділені курсивом в переліку їх атрибутів та є очевидними. Слід зазначити, що для сутності «Класний керівник» її атрибутів недостатньо для формування ключа, а тому було запозичено ключ пов'язаної сутності «Вчитель».
   З опису предметної області випливає наявність бінарних зв'язків «учень навчається в класі» («один до багатьох»), «класний керівник керує класом» («один до одного»).
   Оскільки для класного керівника виділено особливий атрибут в порівнянні з учителем, слід виділяти цей клас об'єктів як окрему сутність та додати зв'язок «класний керівник є вчителем» («один до одного»)

типу «загальний вид — різновид». Якщо б класний керівник не мав окремих атрибутів, то достатньо було б побудувати зв'язок «вчитель керує класом» («один до одного»), з якого очевидним чином можна було б отримати перелік класних керівників.

Також предметна область передбачає один тернарний зв'язок «розклад» («багато до багатьох до багатьох») між сутностями «Вчитель», «Клас» та «Предмет». Замінити цей тернарний зв'язок бінарними неможливо, оскільки інформація з цих зв'язків не буде повною (дивіться матеріали розділу). Цей зв'язок також матиме свої атрибути: «номер уроку», «день тижня», «кабінет».

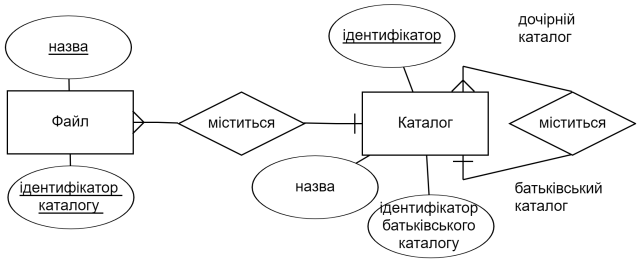
2. Відповідну ER-модель зображено на рисунку:



№18. Як зазначено в матеріалах розділу, сутності в текстовому описі предметної області зазвичай представлені іменниками. Сутностями цієї предметної області є: «Файл» (атрибути *«назва», «ідентифікатор каталогу»*), «Каталог» (атрибути *«ідентифікатор»*, «назва», «ідентифікатор батьківського каталогу»). Ключі сутностей виділені курсивом в переліку їх атрибутів. Оскільки в одному каталозі не може бути однаково названих файлів, то ключ сутності «Файл» є комбінованим з полів «назва» та «код каталогу», останнє з яких є зовнішнім ключем. Ключ сутності «Каталог» є очевидним. Слід зазначити, що окрім первинного, для сутності «Каталог» можна визначити ще один потенційний ключ з двох полів: «назва» та «код батьківського каталогу», з чим власне й пов'язане додавання зовнішнього ключа «код батьківського каталогу» до атрибутів сутності «Каталог».

З опису предметної області випливає наявність бінарного зв'язку «файл розташований в каталозі» («один до багатьох»). Також предметна область передбачає унарний зв'язок «каталог розташований в каталозі». Для визначення його множинності визначимо ролі сутності в зв'язку як «дочірній каталог» та «батьківський каталог» та інтерпретуватимемо зв'язок як бінарний. За умовою множинність цього зв'язку «один до багатьох».

Відповідну ER-модель зображено на рисунку:

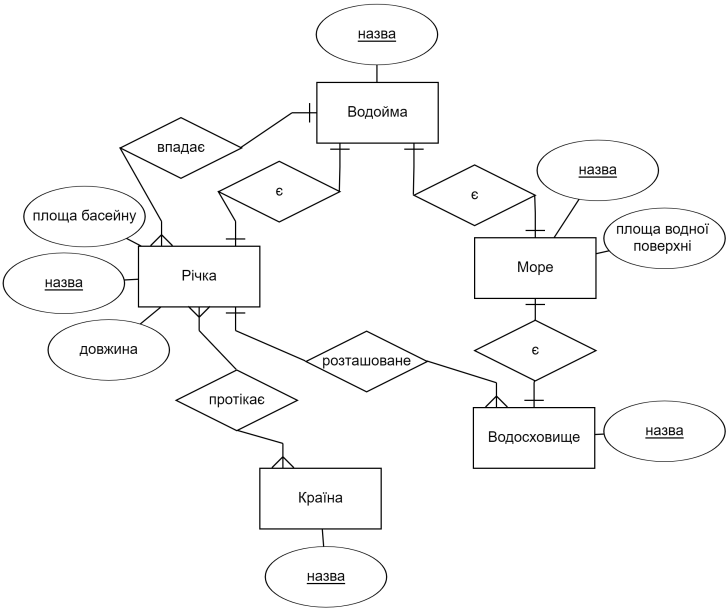


№19. В цьому завданні визначення переліку сутностей не є таким очевидним, як в попередніх, тому перед цим зробимо короткий аналіз на перспективу. З опису предметної області, річка може впадати в іншу річку, море або водосховище. Якщо створити для збереження цього факту три зв'язки «річка впадає в річку», «річка впадає в море» та «річка впадає у водосховище», буде можливим збереження факту, що річка одночасно впадає, наприклад, і в річку, і в море, що суперечить опису предметної області. Виходом з цієї ситуації може бути виділення окремої сутності-загального виду «Водойми», що об'єднує спільні атрибути («назва») та спільні зв'язки (впадання річки в водойму), та поєднання її з різновидами зв'язами типу «є».

Отже, сутностями цієї предметної області є: «Водойма» (атрибут *«назва»*), «Річка» (атрибути *«назва»*, «довжина», «площа басейну»), «Море» (атрибути *«назва»*, «площа водної поверхні»), «Водосховище» (атрибути *«назва»*), «Країна» (атрибути *«назва»*). Ключі сутностей виділені курсивом в переліку їх атрибутів і є очевидними. Слід зазначити, що в сутностях «Річка», «Море» та «Водосховище» для формування ключа було запозичено ключ пов'язаної сутності «Водойма».

З опису предметної області випливає наявність бінарних зв'язків «річка протікає через країну» («багато до багатьох»), «річка впадає в водойму» («один до багатьох»), «річка є водоймою» («один до одного»), «море є водоймою» («один до одного»), «водосховище є (штучним) морем» («один до одного»), «водосховище розташоване на річці» («один до багатьох»).

Відповідну ER-модель зображено на рисунку:



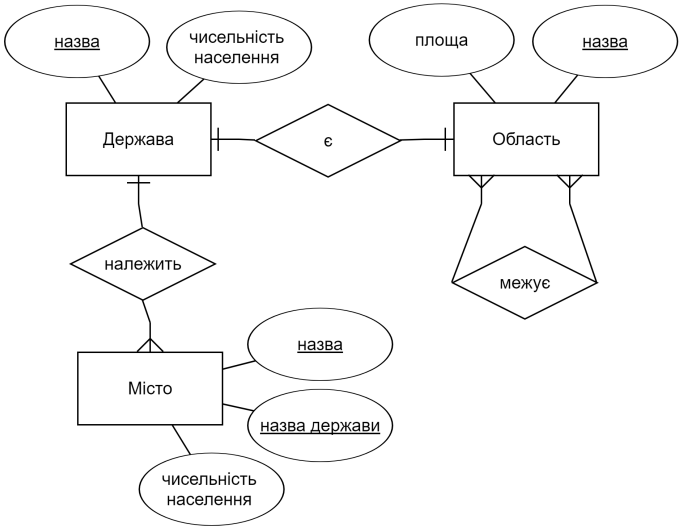
№20. В цьому завданні визначення переліку сутностей не є очевидним, тому перед цим зробимо короткий аналіз на перспективу. З опису предметної області, області географічної карти можуть межувати між собою. Найбільш прямолінійним буде розв'язок, в якому буде виділено окремі сутності для держав та морів та чотири зв'язки для кожної можливої пари об'єктів, які можуть межувати. Це теж коректний з точки зору процесу моделюва-

ння варіант рішення завдання, проте його результат є надто громіздким і загальна схема бази даних буде малоінформативною. Виходом з цієї ситуації може бути виділення окремої сутності-загального виду «Області» (на що власне натякає умова), що об'єднує спільні атрибути («назва» та «площа») та спільні зв'язки (межування областей між собою), та поєднання її з різновидами зв'язами типу «є». Причому, оскільки моря не мають окремих атрибутів, їх можна не виділяти в окрему сутність. Визначення морів серед областей є простим: всі області, що не є державами, є морями.

Отже, сутностями цієї предметної області є: «Область» (атрибути *«назва»*, «площа»), «Держава» (атрибути *«назва»*, «чисельність населення»), «Місто» (атрибути *«назва»*, *«назва держави»*, «чисельність населення»). Ключі сутностей виділені курсивом в переліку їх атрибутів і є очевидними. Слід зазначити, що в сутностях «Держава» та «Місто» для формування ключа було запозичено ключі пов'язаних сутностей «Область» та «Держава», відповідно.

З опису предметної області випливає наявність бінарних зв'язків «місто належить державі» («один до багатьох»), «держава є областю» («один до одного») та унарного зв'язку «область межує з областю» («багато до багатьох»).

Відповідну ER-модель зображено на рисунку:



№21. Найбільш прямолінійним варіантом збереження лабіринту є збереження бітового поля: 100 об'єктів з двома атрибутами: номером та бітом,

який зберігає значення 1, якщо в комірці є тунель, та 0, якщо тунель в комірці відсутній. Але такий варіант збереження не надає змогу накласти на тунелі обмеження, передбачені текстовим описом предметної області, і є надто затратним.

Більш оптимальним є збереження чотирьох значень для кожного тунелю: по дві координати його початку та кінця. У цьому випадку пам'ять, що займатиме таке збереження, буде меншою, проте можливості накласти заборону на тунелі по діагоналі все одно немає.

Оскільки описом предметної області передбачено лише два типи тунелів, то розглянемо такі тунелі окремо. Вертикальні тунелі містяться в одному стовпці квадрату, а отже для його збереження цілком достатньо трьох чисел: номеру стовпця та номерів рядків початку і кінця. Аналогічною є ситуація для горизонтальних тунелів, їх можна подати числами номеру рядка та номерів стовпців початку і кінця тунелю. Якщо обрати ці дві множини об'єктів в якості сутностей предметної області, всіх вимог текстового опису буде дотримано. Дійсно, діагональний тунель в такій моделі не може бути збережений, оскільки для нього не має відповідної сутності.

Отже, сутностями цієї предметної області є «Горизонтальний тунель» (атрибути *«номер рядка», «номер стовпця початку», «номер стовпця кінця»*) та «Вертикальний тунель» (атрибути *«номер стовпця», «номер рядка початку», «номер рядка кінця»*). Ключі сутностей виділені курсивом в переліку їх атрибутів і є очевидними, адже кожен тунель однозначно визначається трьома числами.

Зв'язків описом предметної області тут не передбачено.

Відповідну ER-модель зображено на рисунку:

Одним із можливих варіантів розв'язання цього завдання є також об'єднання цих сутностей в одну сутність «Тунель» з атрибутами «номер рядка/стовпця», «координата початку», «координата кінця» та «тип тунелю». Проте такий варіант організації значно програє викладеному вище за

змістовністю схеми та легкістю подальшого використання, в чому ви переконаєтесь в подальших розділах книги.

№22.

1.
   - $\{5,9\} \cap \{1,2,9,12\} = \{9\}$; $\{5,9\} \cup \{1,2,9,12\} = \{1,2,5,9,12\}$; $\{5,9\} \setminus \{1,2,9,12\} = \{5\}$;
   - $\{\frac{p}{q} | p \in N, q \in N\} \cap Z = Z$ (перша множина представляє собою множину раціональних дробів, а множина цілих чисел є її підмножиною); $\{\frac{p}{q} | p \in N, q \in N\} \cup Z = \{\frac{p}{q} | p \in N, q \in N\}$; $\{\frac{p}{q} | p \in N, q \in N\} \setminus Z = \{\frac{p}{q} | p \in N, q \in N, p$ не ділиться на $q\}$;
   - $\{x^2 | x - $парне$\} \cap \{x | x - $парне$\} = \{x^2 | x - $парне$\}$; $\{x^2 | x - $парне$\} \cup \{x | x - $парне$\} = \{x | x - $парне$\}$; $\{x^2 | x - $парне$\} \setminus \{x | x - $парне$\} = \varnothing$;
   - $\{6x | x \in N\} \cap \{6x + 3 | x \in N\} = \varnothing$ (адже числа мають різну остачу при діленні на 6); $\{6x | x \in N\} \cup \{6x + 3 | x \in N\} = \{3x | x \in N\}$ (адже числа, кратні 3, мають при діленні на 6 остачу або 0, або 3); $\{6x | x \in N\} \setminus \{6x + 3 | x \in N\} = \{6x | x \in N\}$.
2.
   - Нехай $x \in A$. Тоді $x \in A \cup B$, адже, за означенням об'єднання множин, $A \cup B$ містить всі об'єкти, що є елементами множини A. Отже, $A \subseteq A \cup B$.
   - Нехай $x \in A \cap B$. За означенням перетину множин, $A \cap B$ складається з усіх об'єктів, що є елементами A і B одночасно. Звідси $x \in A$ і $A \cap B \subseteq A$.
   - Нехай $x \in A \setminus B$. Тоді, за означенням різниці, $x \in A$ і $x \notin B$. Отже, існує елемент множини A, що не є елементом множини B. Отже, $A \nsubseteq B$. Суперечність. Отже, початкове припущення невірне і $A \setminus B = \varnothing$ (не містить елементів).

№23.

1. Нехай маємо множину математиків M та філософів P. Тоді за умовою $|M \cap P| = \frac{|M|}{7}$ і $|M \cap P| = \frac{|P|}{9}$. Отже, $|M| = 7|M \cap P|$ і $|P| = 9|M \cap P|$. Отже, філософів більше в $\frac{9}{7}$ разів.
2. Іван та Ганна, окрім трьох найкрасивіших, разом полили $1006 + 1006 = 2012$ кущів. Отже, неполитими залишились $2018 - 2012 - 3 = 3$ кущі.
3. Нехай в усіх колах по три сосни. Тоді в трьох малих колах загалом $3 + 3 + 3 = 9$ сосен, адже вони не перетинаються. З іншого боку, в двох великих колах максимум може бути $3 + 3 = 6$ сосен. Оскільки перша множина є підмножиною другої, досягаємо суперечності. Звідси не в усіх колах по 3 сосни.

№24. Нехай дано таблицю, кожен рядок якої відповідає певному елементу множини A, а кожен стовпець — множини B. На перетині рядка x та стовпця y поставимо пару (x, y). За побудовою, в комірках таблиці розташований декартів добуток множин A та B. Очевидно, що всі елементи комірок різні, загальна їх кількість $m * n$. Звідси потужність множини $A \times B$ — теж $m * n$.

№25.

1. - $\{1,2\} \times \{1,3\} = \{(1,1),(1,3),(2,1),(2,3)\}$;
   - $\{a,b\} \times \varnothing = \varnothing$;
   - $\{$яблуко, груша, кокос$\} \times \{$яблуня, груша, пальма$\}$ = (яблуко, яблуня), (яблуко, груша), (яблуко, пальма), (груша, яблуня), (груша, груша), (груша, пальма), (кокос, яблуня), (кокос, груша), (кокос, пальма) $\}$;
   - $R \times R = \{(x,y) | x \in R, y \in R\}$ (декартів добуток двох прямих — декартова площина).
2. $\{1,2,3\} \times \{1,2,3\}$.

№26.

1. $\{(1,1),(2,4),(3,9)\}$;
2. $\{(x,x) | x \in N\}$;
3. $\{(1,2x+1) | x \in N\} \cup \{(3,2x+1) | x \in N\} \cup \{(2,2x) | x \in N\}$.

№27.

1. Нехай дано значення поля «код виробника» таблиці «товари». Здійснюємо прохід записами таблиці «виробники», доки значення поля «код» поточного запису не рівний значенню поля «код виробника» таблиці «товари». Повертаємо значення поля «назву виробника» для поточного запису після завершення проходу.
2. Пропонуємо виконати самостійно за текстовим описом. В табличному процесорі здійснити пошук може допомогти функція вертикального пошуку VLOOKUP (ВПР).

№28. Для відтворення зв'язку множинністю «один до багатьох» на реляційній моделі слід до полів таблиці, якій відповідає позначка «багато», додати ключ таблиці, якій відповідає позначка «один». В такому разі кожному запису таблиці з позначкою «багато» ставиться у відповідність ключ деякого запиту таблиці з позначкою «один». Отже, кожен запис таблиці з позначкою «багато» бере участь у зв'язку не більше одного разу (адже одному

запису відповідає не більше одного значення ключа іншої таблиці). Звідси, таблиця з позначкою «багато» є зв'язаною. З іншого боку, таблиця з позначкою «один» може брати участь у зв'язку багато разів, якщо код певного її запису принаймні двічі зазначений для деяких записів таблиці з позначкою «багато». Тому таблиця з позначкою «один» не є зв'язаною, а отже є основною.

№29.

- у зв'язку «чоловік одружений з жінкою» і чоловік, і жінка беруть участь не більше одного разу і однаково часто, а отже вони обидві можуть бути зв'язаними; визначити зв'язаною можна, наприклад, таблицю чоловіків, а таблицю жінок — основною;
- у зв'язку «планета є небесним тілом» і планета, і небесне тіло беруть участь не більше одного разу, а отже вони обидві можуть бути зв'язаними; оскільки всі планети беруть участь у зв'язку, то зв'язаною визначається таблиця планет, а основною — таблиця небесних тіл;
- у зв'язку «мати народила дитину» дитина бере участь рівно один раз, а мати може брати участь більше одного разу, тому таблиця матерів тут є основною, а таблиця дітей — зв'язаною;
- у зв'язку «вчитель є класним керівником певного класу» і вчитель, і класний керівник беруть участь не більше одного разу, а отже вони обидві можуть бути зв'язаними; оскільки всі класні керівники беруть участь у зв'язку, то зв'язаною визначається таблиця класних керівників, а основною — таблиця вчителів;
- у зв'язку «місто належить до країни» місто бере участь рівно один раз, а країна може брати участь більше одного разу, тому таблиця країн тут є основною, а таблиця міст — зв'язаною.

№30. У зв'язку «загальний вид — різновид» і таблиця-загальний вид, і таблиця-різновид беруть участь не більше одного разу, а отже вони обидві можуть бути зв'язаними. Проте, оскільки всі записи таблиці-різновиду беруть участь у зв'язку, то зв'язаною визначається саме таблиця-різновид, а основною — таблиця-загальний вид.

№31. Ситуація, коли обидві таблиці є основними, є в реляційних базах даних нереалізовною. Дійсно, оскільки будь-якому запису будь-якої з цих двох таблиць може відповідати багато записів іншої таблиці, до кожного запису нам довелося б додати цілий перелік ключів пов'язаних з ним записів, що в звичайному табличному поданні не є можливим.

№32. Попри те, що у відповідному полі таблиці нічого не буде записано, комп'ютер все одно виділятиме пам'ять під цю комірку пам'яті. Через те, що більшість працівників не є керівниками складів, при великій кількості працівників зайвий резерв пам'яті буде значно більшим, що впливатиме на ефективність роботи з базою даних.

№33. Нехай справедливо протилежне: зовнішній ключ у зв'язку множинністю «один до одного» не є потенційним ключем таблиці. Тоді існує можливість повторення певного значення цього ключа, тобто можуть існувати два записи, у яких значення цього зовнішнього ключа співпадає. Тоді одному запису зв'язаної таблиці відповідатиме два записи поточної, що суперечить визначенню множинності «один до одного». Суперечність. Отже, зовнішній ключ у зв'язку множинністю «один до одного» є потенційним ключем таблиці.

№34. В цій предметній області всі зв'язки — типу «загальний вид — різновид», а тому мають множинність «один до одного». Оскільки за доведеною в матеріалах теоремою в зв'язку типу «загальний вид — різновид» зв'язаною таблицею є завжди таблиця-різновид, то для забезпечення цих зв'язків слід з таблиці-загального виду ключ винести в таблицю-різновид та створити зв'язок за відповідними полями.

В нашому випадку поле «ідентифікатор» вже було винесено у сутності-різновиди задля створення ключа, а тому додавати знову його не потрібно.

Реляційна модель цієї предметної області матиме наступний вигляд:

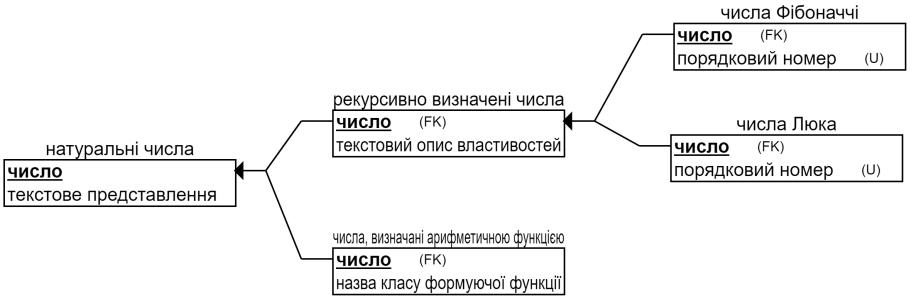
Оскільки зовнішні ключі є одночасно первинними ключами своїх таблиць, то множинність «один до одного» досягається автоматично.

№35. В цій предметній області всі зв'язки — типу «загальний вид — різновид», а тому мають множинність «один до одного». Для забезпечення цих зв'язків слід з таблиці-загального виду ключ винести в таблицю-різновид та створити зв'язок за відповідними полями. В нашому випадку поле «число»

вже було винесено у сутності-різновиди задля створення ключа, а тому додавати знову його не потрібно.

Слід звернути увагу, що потенційними ключами є також поля «порядковий номер» для таблиць «числа Фібоначчі» та «числа Люка», а тому їх необхідно визначити як унікальні на реляційній моделі предметної області.

Реляційна модель цієї предметної області матиме наступний вигляд:



Оскільки зовнішні ключі є одночасно первинними ключами своїх таблиць, то множинність «один до одного» досягається автоматично.

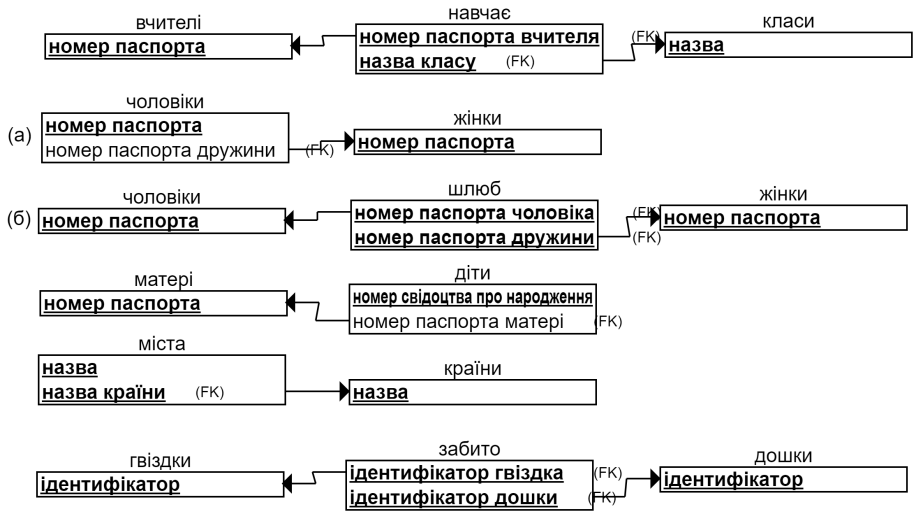
№36. Зв'язки «вчитель навчає клас» та «гвіздок забито в дошку» мають множинність «багато до багатьох», а тому реалізуються за допомогою винесення ключів в допоміжну таблицю та зв'язування таблиць за відповідними полями. Ключем цієї таблиці визначається пара відповідних зовнішній ключів аби не повторювати одну й ту ж інформацію кілька разів.

Зв'язок «чоловік одружений з жінкою» має множинність «один до одного». В даному випадку зв'язаною може бути визначена будь-яка з двох таблиць. Наприклад, оберемо таблицю «чоловіки» зв'язаною, а таблицю «жінки» — основною. Тоді до таблиці «чоловіки» слід винести ключ таблиці «жінки» та зв'язати таблиці за цими полями (а). Слід зазначити, що для забезпечення множинності «один до одного» слід помітити зовнішній ключ унікальним.

З іншого боку, цей же зв'язок може бути реалізований за допомогою допоміжної таблиці (б). Такий варіант втілення зв'язку буде кращим з точки зору зайвої пам'яті, якщо неодружених чоловіків та жінок доволі висока частка.

Зв'язки «мати народила дитину» та «місто належить до країни» є зв'язками множинністю «один до багатьох». Для їх відтворення слід ключ таблиці з позначкою «один» винести в таблицю з позначкою «багато» та зв'язати таблиці за цими полями.

Реляційні моделі зв'язків матимуть наступний вигляд:

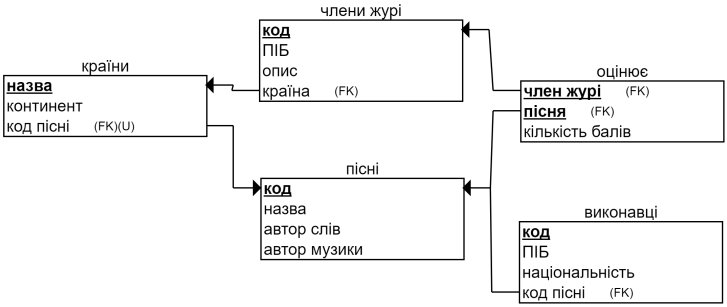


№37. Зв'язок «член журі оцінює пісню» має множинність «багато до багатьох», а тому реалізується за допомогою винесення ключів в допоміжну таблицю «оцінює» та зв'язування таблиць за відповідними полями. Ключем цієї таблиці визначається пара відповідних зовнішній ключів аби не повторювати одну й ту ж інформацію кілька разів. Атрибут зв'язку відтворюється на реляційній моделі додатковим полем «кількість балів» відповідної допоміжної таблиці.

Зв'язок «країна виступає з піснею» має множинність «один до одного». В даному випадку зв'язаною може бути визначена будь-яка з двох таблиць, оскільки всі записи обох таблиць беруть участь у цьому зв'язку. Наприклад, оберемо таблицю «країни» зв'язаною, а таблицю «пісні» — основною. Тоді до таблиці «країни» слід винести ключ таблиці «пісні» та зв'язати таблиці за цими полями. Слід зазначити, що для забезпечення множинності «один до одного» слід помітити зовнішній ключ унікальним.

Зв'язки «член журі представляє країну» та «виконавець виконує пісню» є зв'язками множинністю «один до багатьох». Для їх відтворення слід ключ таблиці з позначкою «один» винести в таблицю з позначкою «багато» та зв'язати таблиці за цими полями.

Реляційна модель цієї предметної області матиме наступний вигляд:



№38.

Зв'язок «чоловік одружений з жінкою» має множинність «один до одного». В даному випадку атрибут зв'язку «дата шлюбу» разом із зовнішнім ключем вноситься в таблицю «чоловіки» і може цілком інтерпретуватися як атрибут чоловіка.

Зв'язок «студент отримав результат оцінювання з предмету» мають множинність «багато до багатьох», а тому реалізуються за допомогою винесення ключів в допоміжну таблицю та зв'язування таблиць за відповідними полями. Ключем цієї таблиці визначається пара відповідних зовнішній ключів аби не повторювати одну й ту ж інформацію кілька разів. Атрибут зв'язку «результат» вноситься в допоміжну таблицю як окреме поле.

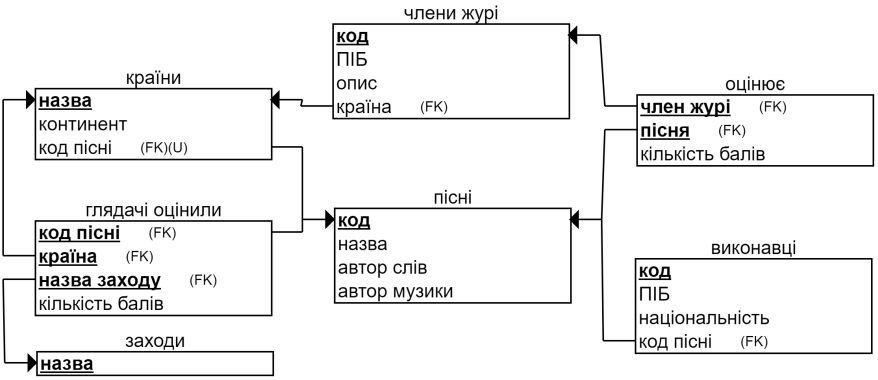
Зв'язок «мати народила дитину в певний день» має множинність «один до багатьох». В цьому випадку атрибут зв'язку «дата народження» разом із зовнішнім ключем вноситься в таблицю «діти» і може цілком інтерпретуватися як атрибут дитини.

Реляційна модель предметної області матиме наступний вигляд:

№39. Тернарний зв'язок «глядачі країни присудили пісні в певному заході певну кількість балів» має множинність «багато до багатьох до багатьох»,

а тому відтворити його на реляційній моделі можна за допомогою допоміжної таблиці «глядачі оцінили», в яку слід знести ключі таблиць «країни», «пісні» та «заходи». Атрибут цього зв'язку «кількість балів» вноситься в допоміжну таблицю як окреме поле. Ключем таблиці є комбінація усіх зовнішніх ключів з метою уникнення повторюваного збереження інформації.

Реляційні моделі зв'язків мають наступний вигляд:



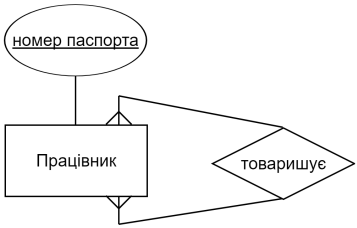
№40. Це може призвести до ситуації, коли, наприклад, для працівника *Іванової Лариси Трохимівни* вказано, що вона одружена з *Івановим Петром Сергійовичем*, а для нього вказано, що він неодружений. Виникає неузгодженість інформації в таблиці.

№41. Слід розділити таблицю «працівники» на дві таблиці: «працівники-чоловіки» та «працівники-жінки», — після чого замінити унарний зв'язок бінарним зв'язком «шлюб» між створеними двома таблицями.

№42. Оскільки множинність зв'язку — «багато до багатьох», то навіть попри його унарність, його слід реалізовувати за допомогою допоміжної таблиці. До неї слід двічі внести ключ таблиці «футбольні команди» — назву команди, а також атрибут зв'язку — результат матчу.

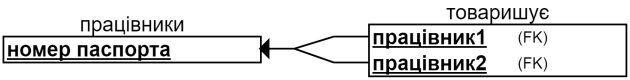
Реляційна модель матиме наступний вигляд:

№43. Зв'язок «працівник товаришує з працівником» має множинність «багато до багатьох» і його ER-модель має наступний вигляд:



Оскільки множинність зв'язку — «багато до багатьох», то навіть попри його унарність, його слід реалізовувати за допомогою допоміжної таблиці. До неї слід двічі внести ключ таблиці «працівники» — номер паспорта працівника.

Коректна реляційна модель цього зв'язку має наступний вигляд:



№44. Оскільки відтворення сутностей з ER-діаграми на реляційній схемі є тривіальним, зосередимо увагу на відтворені наявних в предметній області зв'язків.

Загалом в предметній області чотири зв'язки: два бінарні множинністю «один до одного», один бінарний множинністю «один до багатьох» та один тернарний множинністю «багато до багатьох до багатьох» з атрибутами зв'язку. Зупинимось на кожному зв'язку окремо.

Зв'язок «класний керівник є вчителем» є зв'язком типу «загальний вид — різновид». Як зазначено в матеріалах розділу, в цьому зв'язку зв'язаною є таблиця, що представляє різновид, а тому для реалізації зв'язку на реляційній схемі слід ключ таблиці «вчителі» винести в таблицю «класні керівники» в якості зовнішнього ключа та побудувати зв'язок за цими полями. Оскільки в даному випадку поле «номер паспорта» вже було додано для утворення ключа сутності «Класний керівник», вдруге його додавати не потрібно, а достатньо просто зв'язати цей зовнішній ключ з відповідним йому ключем таблиці «вчителі». Слід зазначити, що зовнішній ключ є одночасно ключем власної таблиці, а тому множинність «один до одного» зв'язку дотримано коректно.

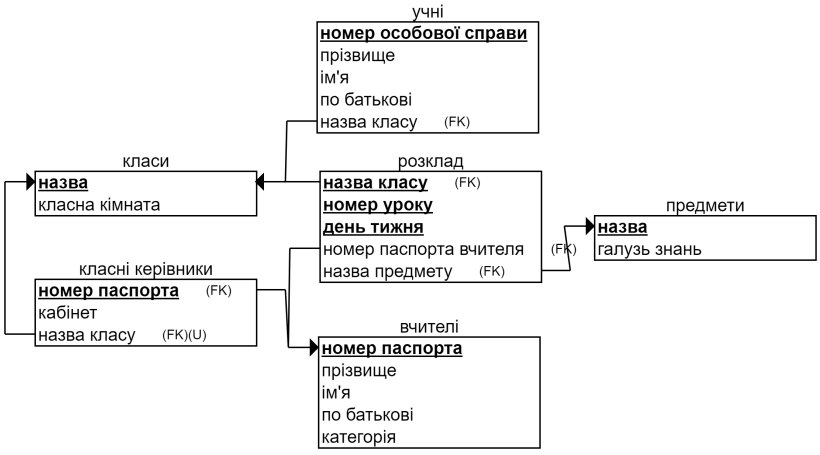
Зв'язок «класний керівник керує класом» є зв'язком множинністю «один до одного». В цьому випадку всі записи обох таблиць братимуть участь у

зв'язку, тож оберемо таблицю «класні керівники» зв'язаною в цьому зв'язку. В такому разі для реалізації зв'язку на реляційній схемі слід ключ таблиці «класи» винести в таблицю «класні керівники» в якості зовнішнього ключа та побудувати зв'язок за цими полями. Слід зазначити, що аби дотримати множинність «один до одного» зв'язку, слід позначити зовнішній ключ як унікальний.

Зв'язок «учень навчається в класі» є зв'язком множинністю «один до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключ таблиці «класи» винести в таблицю «учні» в якості зовнішнього ключа та побудувати зв'язок за цими полями (адже ключ таблиці з позначкою «один» виноситься в таблицю з позначкою «багато»).

Зв'язок «розклад» є тернарним зв'язком множинністю «багато до багатьох до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід скористатися допоміжною таблицею «розклад», в яку винести ключі трьох поєднуваних таблиць: таблиці «класи», таблиці «вчителі» та таблиці «предмети», — та побудувати зв'язок за цими полями. Оскільки цей зв'язок також має власні атрибути («номер уроку», «день тижня» та «кабінет») необхідно додати відповідні їм поля до допоміжної таблиці. Ключем допоміжної таблиці слід зазначити комбінацію полів «номер уроку», «день тижня» та «назва класу».

Коректна реляційна модель цієї предметної області має наступний вигляд:



№45. Оскільки відтворення сутностей з ER-діаграми на реляційній схемі є тривіальним, зосередимо увагу на відтворені наявних в предметній області

зв'язків.

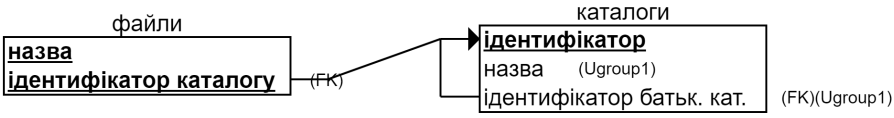
Загалом в предметній області наявні два зв'язки: один бінарний та один унарний — обидва множинністю «один до багатьох» без власних атрибутів.

Зв'язок «файл міститься в каталозі» є зв'язком множинністю «один до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключ таблиці «каталоги» винести в таблицю «файли» в якості зовнішнього ключа та побудувати зв'язок за цими полями (адже ключ таблиці з позначкою «один» виноситься в таблицю з позначкою «багато»). Оскільки в даному випадку поле «ідентифікатор каталогу» вже було додано для утворення ключа сутності «Файл», вдруге його додавати не потрібно, а достатньо просто зв'язати цей зовнішній ключ з відповідним йому ключем таблиці «каталоги».

Зв'язок «каталог міститься в каталозі» є унарним зв'язком множинністю «один до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключ таблиці «каталоги» винести в цю ж таблицю в якості зовнішнього рекурсивного ключа та побудувати зв'язок за цими полями (адже ключ таблиці з позначкою «один» виноситься в таблицю з позначкою «багато»). Оскільки в даному випадку поле «ідентифікатор каталогу» вже було додано для утворення потенційного ключа сутності «Каталог», вдруге його додавати не потрібно, а достатньо просто зв'язати цей зовнішній ключ з відповідним йому ключем таблиці «каталоги».

Відмітимо, що оскільки раніше було визначено додатковий потенційний ключ сутності «Каталог» з двох атрибутів «назва» та «ідентифікатор батьківського каталогу», то відповідні їм поля таблиці «каталоги» повинні увійти до унікального ключа таблиці (на рисунку нижче їх позначено Ugroup1 — перша група унікальних полів).

Коректна реляційна модель цієї предметної області має наступний вигляд:



№46. Оскільки відтворення сутностей з ER-діаграми на реляційній схемі є тривіальним, зосередимо увагу на відтворені наявних в предметній області зв'язків.

Загалом в предметній області наявні: три бінарних зв'язки типу «загальний вид — різновид» множинністю «один до одного» та два бінарних зв'язки множинністю «один до багатьох» без власних атрибутів.

Зв'язки «річка є водоймою» та «море є водоймою» визначають класифікацію водойм. Як зазначено в матеріалах розділу, в зв'язках типу «загальний вид — різновид» зв'язаною є таблиця, що представляє різновид, а тому для реалізації цих зв'язків на реляційній схемі слід ключ таблиці «водойми» винести в таблиці-різновиди в якості зовнішніх ключів та побудувати зв'язки за цими полями. Оскільки в даному випадку поле «назва» вже було додано для утворення ключа сутностей-різновидів «Річка» та «Море», вдруге їх додавати не потрібно, а достатньо просто зв'язати ці зовнішні ключі з відповідним йому ключем таблиці «водойми».

Зв'язок «водосховище є морем» визначає різновид морів. З аналогічних міркувань, для його реалізації на реляційній схемі слід ключ таблиці «моря» винести в таблицю «водосховища» в якості зовнішнього ключа та побудувати зв'язок за цими полями. Оскільки в даному випадку поле «назва» теж вже було додано для утворення ключа сутності «Водосховище», вдруге його додавати не потрібно, а достатньо просто зв'язати зовнішній ключ з відповідним йому ключем таблиці «моря».

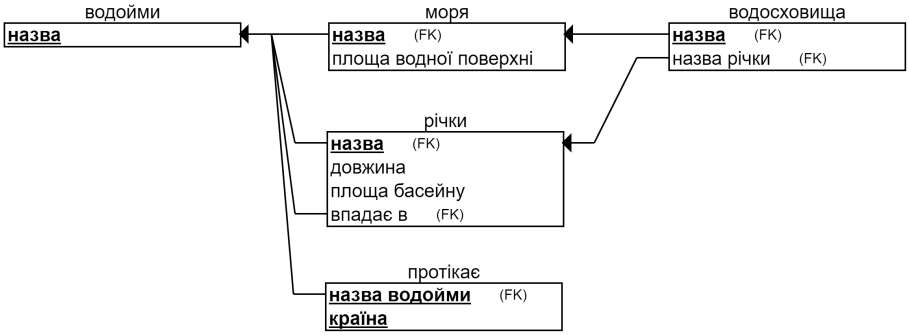
Слід зазначити, що зовнішній ключ в усіх трьох вище розглянутих зв'язках є одночасно ключем власної таблиці, а тому множинності «один до одного» зв'язку дотримано коректно.

Зв'язок «річка впадає у водойму» є зв'язком множинністю «один до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключ таблиці «водойми» винести в таблицю «річки» в якості зовнішнього ключа та побудувати зв'язок за цими полями (адже ключ таблиці з позначкою «один» виноситься в таблицю з позначкою «багато»).

Зв'язок «водосховище розташоване на річці» теж є зв'язком множинністю «один до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключ таблиці «річки» винести в таблицю «водосховища» в якості зовнішнього ключа та побудувати зв'язок за цими полями (адже ключ таблиці з позначкою «один» виноситься в таблицю з позначкою «багато»).

Зв'язок «річка протікає через країну» є зв'язком множинністю «багато до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключі таблиць «річки» та «країни» винести в допоміжну таблицю «протікає» в якості зовнішнього ключа та побудувати зв'язок за цими полями. В цьому випадку країна характеризується лише одним полем і не має інших зв'язків, тож не обов'язково виділяти окрему таблицю країн.

Коректна реляційна модель цієї предметної області має наступний вигляд:



№47. Оскільки відтворення сутностей з ER-діаграми на реляційній схемі є тривіальним, зосередимо увагу на відтворені наявних в предметній області зв'язків.

Загалом в предметній області наявні: бінарний зв'язок типу «загальний вид — різновид» множинністю «один до одного», бінарний зв'язок множинністю «один до багатьох» та унарний зв'язок множинністю «багато до багатьох» без власних атрибутів.

Зв'язок «держава є областю» є зв'язком типу «загальний вид — різновид». Як зазначено в матеріалах розділу, в цьому зв'язку зв'язаною є таблиця, що представляє різновид, а тому для реалізації зв'язку на реляційній схемі слід ключ таблиці «області» винести в таблицю «держави» в якості зовнішнього ключа та побудувати зв'язок за цими полями. Оскільки в даному випадку поле «назва» вже було додано для утворення ключа сутності «Держава», вдруге його додавати не потрібно, а достатньо просто зв'язати цей зовнішній ключ з відповідним йому ключем таблиці «області». Слід зазначити, що зовнішній ключ є одночасно ключем власної таблиці, а тому множинність «один до одного» зв'язку дотримано коректно.

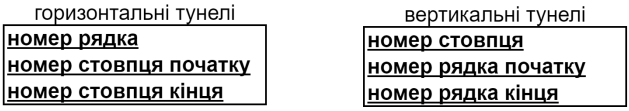
Зв'язок «місто належить до держави» є зв'язком множинністю «один до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід ключ таблиці «держави» винести в таблицю «міста» в якості зовнішнього ключа та побудувати зв'язок за цими полями (адже ключ таблиці з позначкою «один» виноситься в таблицю з позначкою «багато»).

Зв'язок «область межує з областю» є унарним зв'язком множинністю «багато до багатьох». В такому випадку для реалізації зв'язку на реляційній схемі слід скористатися допоміжною таблицею «межує», в яку двічі винести ключ таблиці «області», що бере участь в цьому зв'язку, та побудувати зв'язок за цими полями. Ключем допоміжної таблиці (задля уникнення надлишковості даних) слід зазначити комбінацію з доданих в неї зовнішніх

ключів.

Коректна реляційна модель цієї предметної області має наступний вигляд:

№48. Оскільки відтворення сутностей з ER-діаграми на реляційній схемі є тривіальним, а зв'язки в цій предметній області відсутні, то реляційна модель є тривіальною:



№49. Виконання завдання зводиться в цілому до підбору типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі конструктора СУБД Microsoft Access.

Наступні поля повинні мати тип *Short Text (Короткий текст)*, оскільки передбачають текстові дані:

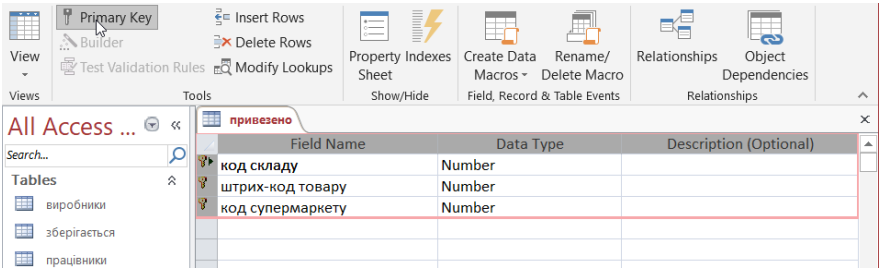
- «назва», «адреса», «номер паспорта керівника» таблиці «склади»;
- «назва» таблиці «товари»;
- «назва», «адреса» таблиці «супермаркети»;
- «назва», «адреса» таблиці «виробники»;
- всі поля таблиць «працівники» та «шлюб».

Поле «роздрібна ціна» таблиці «товари» зберігає грошові дані, а тому повинне мати тип *Currency (Грошова одиниця)*.

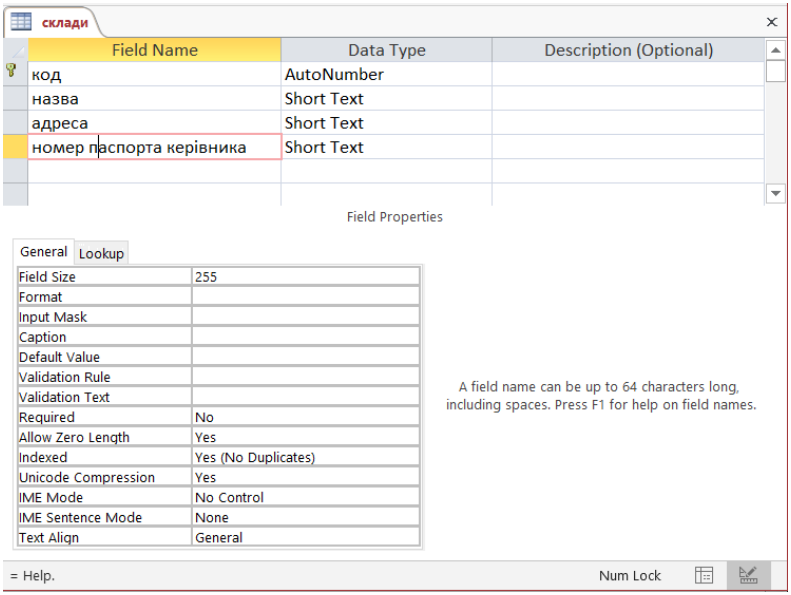
Поля «код» таблиць «склади», «виробники», «супермаркети» можуть мати тип *AutoNumber (Автонумерація)*, якщо передбачається надання їм послідовних унікальних значень. Це полегшить процес введення даних в таблиці.

Решта полів зберігають числові дані, а тому їх тип — *Number (Число)*.

Первинний ключ кожної з таблиць, який на реляційній моделі підкреслено, слід визначити в середовищі Access. Для цього всі поля таблиці, що входять до первинного ключа, слід виділити з затисненою клавішею Ctrl, та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці, як це показано на рисунку:



№50. Налаштування унікальних полів здійснюється в режимі конструктора. При виділенні відповідного поля, в блоці *Field Properties (Властивості полів)* для властивості *Indexed (Індексоване поле)* слід обрати пункт *Yes (No Duplicates) — Так (Без повторень)*, як це показано на рисунку для поля «номер паспорта керівника» таблиці «склади»:

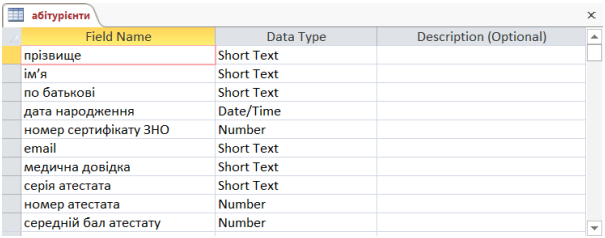


Аналогічні дії потрібно виконати і для полів «номер паспорта чоловіка» та «номер паспорта дружини» таблиці «шлюб».

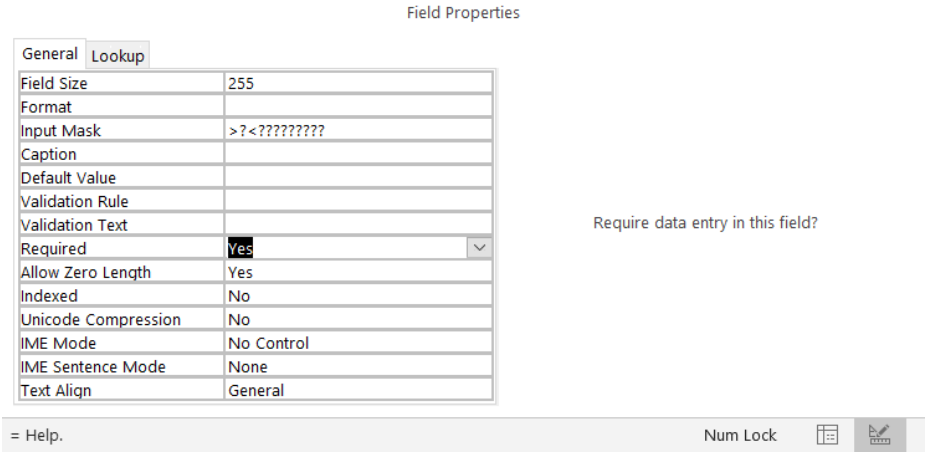
№51. Для полів таблиці «абітурієнти» слід налаштувати такі параметри:

- поля «прізвище», «ім'я», «по-батькові» повинні мати тип *Short Text (Короткий текст)*; для них слід налаштувати маску: *>?<??????????* (спочатку всі наступні символи повинні бути відображені у верхньому регістрі, потім користувач повинен ввести один символ, потім ще до 9 символів, що будуть відображені у нижньому регістрі);
- поле «дата народження» повинно мати тип *Date/Time (Дата й час)*;
- поле «номер сертифікату ЗНО» повинно мати тип *Number (Число)*; для цього поля слід встановити маску введення наступного вигляду: *000000* (слід обов'язково ввести рівно шість цифр);
- поле «email» повинно мати тип *Short Text (Короткий текст)*;
- поле «медична довідка» повинно мати тип *Short Text (Короткий текст)*;
- поле «серія атестата» повинно мати тип *Short Text (Короткий текст)*; для цього поля слід встановити маску введення: *>LL* (слід обов'язково ввести рівно дві літери, обидві — у верхньому регістрі);
- поле «номер атестата» повинно мати тип *Number (Число)*; для цього поля слід встановити маску введення наступного вигляду: *00000000* (слід обов'язково ввести рівно вісім цифр);
- поле «середній бал атестату» повинно мати тип *Number (Число)*; для цього поля слід встановити значення *Double (Подвійне дійсне)* властивості *Field Size (Розмір поля)* аби в ньому можна було зберігати дійсні числа, а також маску введення наступного вигляду: *09,0* (слід обов'язково ввести одну цифру, потім можна ввести ще одну, після цього повинні слідувати десяткова кома та ще одна цифра).

Оскільки всі поля є за умовою вправи обов'язковими, необхідно встановити для кожного з вище наведених полів значення *Yes (Так)* властивості *Required (Обов'язкове поле)* так, як це показано на рисунку для поля «прізвище»:


абітурієнти

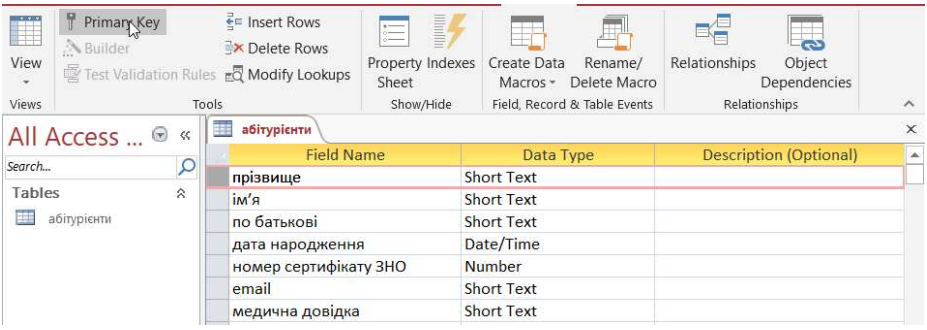
Field Name	Data Type	Description (Optional)
прізвище	Short Text	
ім'я	Short Text	
по батькові	Short Text	
дата народження	Date/Time	
номер сертифікату ЗНО	Number	
email	Short Text	
медична довідка	Short Text	
серія атестата	Short Text	
номер атестата	Number	
середній бал атестату	Number	



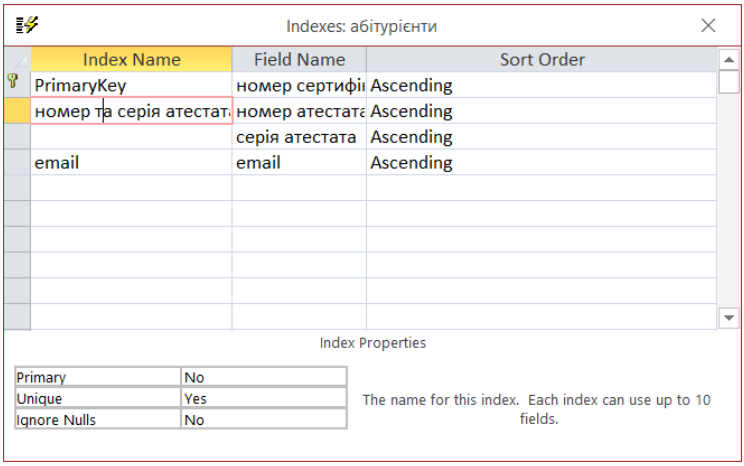
За умовою вправи, потенційними ключами таблиці є:

- поле «номер сертифікату ЗНО»;
- поле «email»;
- комбінація полів «серія атестата» та «номер атестата».

Оберемо серед них первинним ключем, наприклад, поле «номер сертифікату ЗНО». Тоді його слід виділити та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці, як це показано на рисунку:



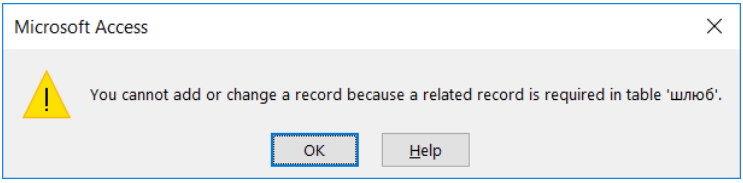
Для решти ключів слід в меню *Indexes (Індекси)* додати індекси без повторень так, як це показано на рисунку:



Зверніть увагу! Властивість *Unique (Унікальний)* повинна мати значення *Yes (Так)* для всіх створених індексів.

№52.

1. Якщо з такими налаштуваннями зв'язку у порожню базу даних почати додавати працівників, то Access видобразить при збереженні першого ж запису наступну помилку:



   В помилці повідомляється, що ви не можете додати або змінити запис, оскільки пов'язаний з ним запис відсутній в таблиці «шлюб».
   Дійсно, зміна напрямку перетягування змінила основну та зв'язану таблиці у цьому зв'язку. Таким чином, тепер саме поле «номер паспорта» таблиці «працівники» вважається зовнішнім ключем. Оскільки зв'язок в Access є засобом для слідкування за цілісністю, зокрема й за цілісністю посилань, то зовнішній ключ повинен значення NULL або його значення повинно відповідати значенню відповідного йому ключа таблиці «шлюб».
2. У випадку множинності «один до багатьох» завжди однозначно відомо, з якого боку розташована основна таблиця (позначка «один»), а

з якого — зв'язана (позначка «багато»). Саме тому СУБД власноруч може визначити помилку в напрямку перетягування і виправити її автоматично без втручання користувача.

№53. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Наступні поля повинні мати тип *Short Text (Короткий текст)*, оскільки передбачають текстові дані:

- «назва», «розташування» таблиці «електростанції»;
- «тип» таблиці «ГЕС»;
- «назва» таблиці «річки»;
- «назва річки» таблиці «розташована»;
- «тип реактора» таблиці «енергоблоки».

Поле «діючість» таблиці «атомні електростанції» зберігає логічне значення, а тому повинне мати тип *Yes/No (Так/Ні)*.

Поле «ідентифікатор» таблиці «електростанції» може мати тип *AutoNumber (Автонумерація)*, якщо передбачається надання електростанціям послідовних унікальних значень. Це полегшить процес введення даних в таблиці. З іншого боку це ж поле в таблицях «ГЕС» та «атомні електростанції» є зовнішнім ключем і не може мати аналогічний тип значень, оскільки використовує значення ідентифікаторів електростанцій, призначені їм в таблиці «електростанції».

Решта полів зберігають числові дані, а тому їх тип — *Number (Число)*.

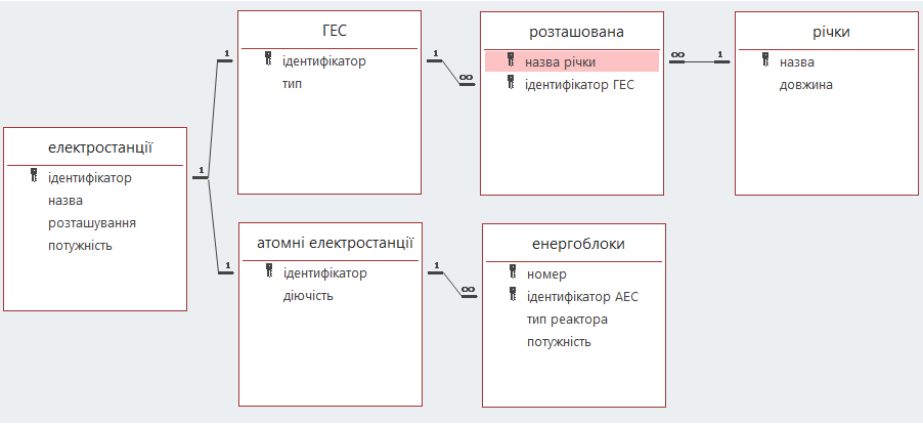
Первинний ключ кожної з таблиць, який на реляційній моделі підкреслено, слід визначити в середовищі Access. Для цього всі поля таблиці, що входять до первинного ключа, слід виділити з затисненою клавішею Ctrl, та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці. Для зв'язків між таблицею «електростанції» та таблицями «ГЕС» і «атомні електростанції» принциповим є перетягування поля «ідентифікатор» саме з таблиці «електростанції» на поля «ідентифікатор» таблиць-різновидів, адже це визначатиме основну та зв'язану таблиці в кожному з цих зв'язків множинністю «один до одного».

Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення.

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не

створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№54. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Поля «ідентифікатор» всіх таблиць повинні мати тип *Number (Число)*, оскільки передбачають числові дані. Поле «зображення» таблиці «механічні» повинно мати тип *Attachment (Вкладення)*, оскільки передбачає збереження в ньому файлу зображення. Решта полів зберігають текстові дані, а тому їх тип — *Short Text (Короткий текст)*.

Примітка: поле «ідентифікатор» в таблиці «головоломки» може також мати тип AutoNumber (Автонумерація) *для полегшення подальшого вводу даних в таблицю. Таблиці-різновиди використовують ідентифікатор головоломки, визначений в таблиці «головоломки», а тому мати тип* AutoNumber (Автонумерація) *не можуть.*

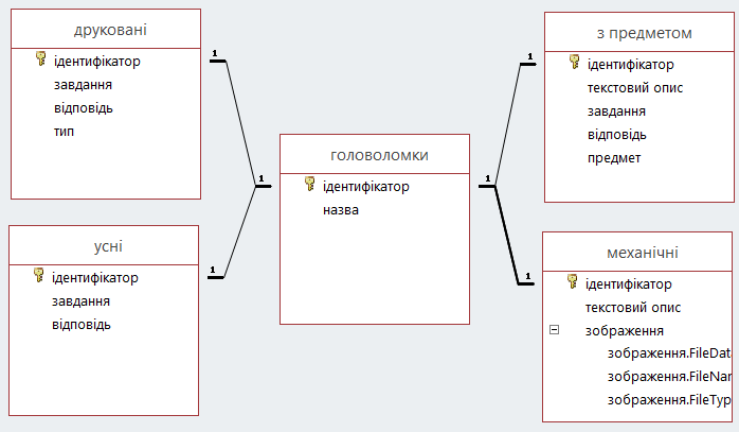
Первинний ключ кожної з таблиць, а це поля «ідентифікатор», слід визначити в середовищі Access. Для цього в кожній таблиці це поле слід виділити та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення. Тут принциповим є перетягування поля «ідентифікатор» саме з таблиці «головоломки»

на поля «ідентифікатор» таблиць-різновидів, адже це визначатиме основну та зв'язану таблиці в кожному з чотирьох зв'язків множинністю «один до одного».

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:

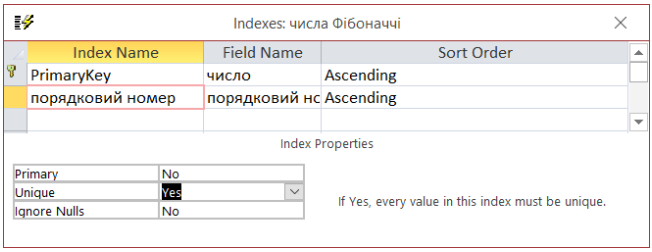


№55. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Поля «число» всіх таблиць та «порядковий номер» таблиць «числа Фібоначчі» і «числа Люка» повинні мати тип *Number (Число)*, оскільки передбачають числові дані. Решта полів зберігають текстові дані, а тому їх тип — *Short Text (Короткий текст)*.

Первинний ключ кожної з таблиць, а це поля «число», слід визначити в середовищі Access. Для цього в кожній таблиці це поле слід виділити та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

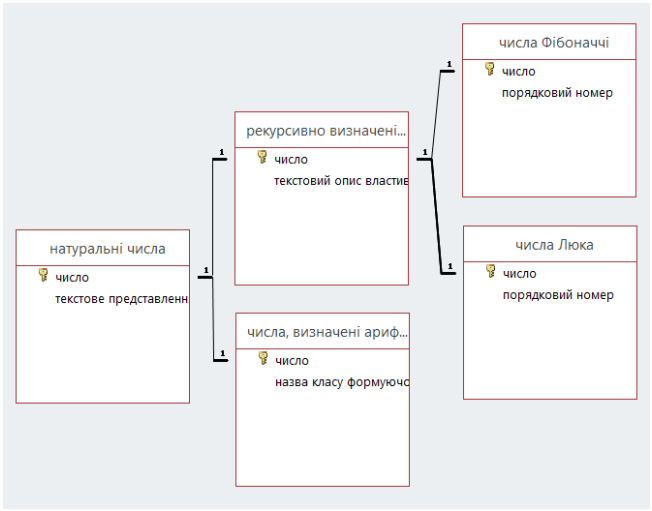
Слід зазначити, що таблиці «числа Фібоначчі» і «числа Люка» мають окрім первинного також і додатковий потенційний ключ «порядковий номер». А тому в кожній з цих таблиць слід виділити це поле та встановити для нього значення *Yes (No Duplicates) — Так (Без повторень)* для властивості *Indexed (Індексоване поле)*.



Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення. Тут принциповим є перетягування поля «число» саме з таблиці «натуральні числа» на поля «число» таблиць «рекурсивно визначені числа» та «числа, визначені арифметичною функцією», а також поля «число» з таблиці «рекурсивно визначені числа» на поля «число» таблиць «числа Фібоначчі» та «числа Люка», адже це визначатиме основну та зв'язану таблиці в кожному з чотирьох зв'язків множинністю «один до одного».

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№56. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Наступні поля таблиць повинні мати тип *Number (Число)*, оскільки передбачають числові дані:

- «класна кімната» таблиці «класи»;
- «кабінет» таблиці «класні керівники»;
- «номер уроку» таблиці «розклад».

Примітка: номер особової справи учня зазвичай складається з першої літери прізвища та числа, тому не може вважатися числом.

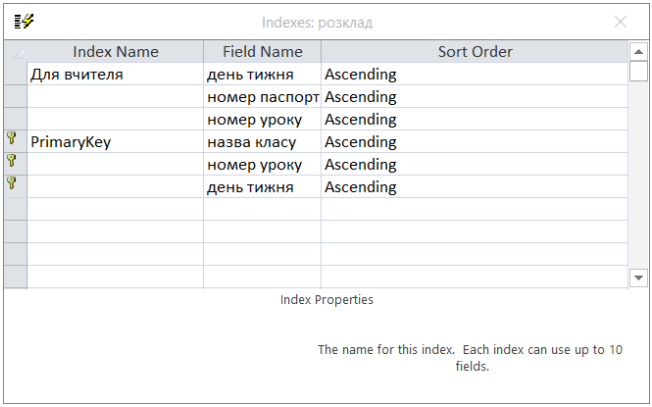
Решта полів зберігають текстові дані, а тому їх тип — *Short Text (Короткий текст)*.

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі Access. Для цього в кожній таблиці поля, що належать до первинного ключа, слід виділити із затиснутою клавішею Ctrl та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

Слід зазначити, що таблиця «розклад» матиме (у зв'язку з додатковими умовами на розклад занять) окрім первинного також і потенційні ключі:

- комбінація полів «день тижня», «назва класу», «номер уроку»;
- комбінація полів «день тижня», «номер паспорта вчителя», «номер уроку».

А тому в таблиці «розклад» в меню *Indexes (Індекси)* режиму конструктора слід додати індекси без повторень так, як це показано на рисунку:



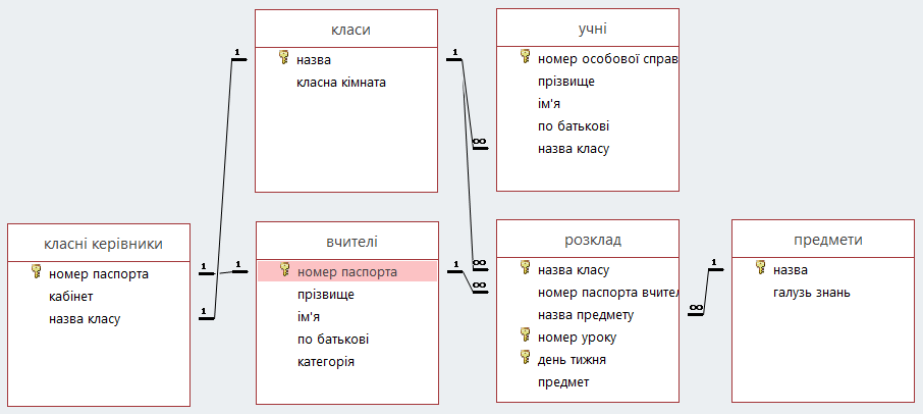
Зверніть увагу! Властивість *Unique (Унікальний)* повинна мати значення *Yes (Так)* для всіх створених індексів.

Також, таблиця «класні керівники» містить унікальне поле «назва класу», необідне для підтримання множинності зв'язку між класами та їх керівниками. А тому в цій таблиці слід виділити поле «назва класу» та встановити для нього значення *Yes (No Duplicates) — Так (Без повторень)* для властивості *Indexed (Індексоване поле)*.

Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення. Тут принциповим є перетягування поля «номер паспорта» саме з таблиці «вчителі» на поле «номер паспорта» таблиці «класні керівники», а також поля «назва» з таблиці «класи» на поле «назва класу» таблиці «класні керівники», адже це визначатиме основну та зв'язану таблиці в кожному з цих зв'язків множинністю «один до одного».

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№57. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

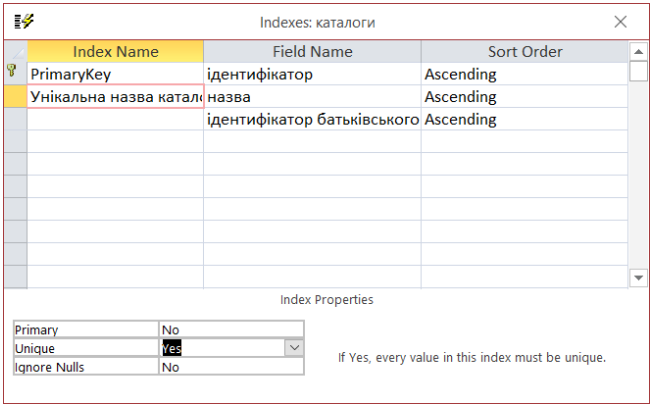
Поля «назва» обох таблиць повинні мати тип *Short Text (Короткий*

текст), оскільки вони передбачають текстові дані. Решта полів зберігають числові дані, а тому їх тип — *Number (Число)*.

Примітка: поле «ідентифікатор» в таблиці «каталоги» може також мати тип AutoNumber (Автонумерація) *для полегшення подальшого вводу даних в таблицю.*

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі Access. Для цього в кожній таблиці поля, що належать до первинного ключа, слід виділити із затиснутою клавішею Ctrl та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

Слід зазначити, що таблиця «каталоги» має окрім первинного також і додатковий потенційний ключ — комбінацію полів «назва» та «ідентифікатор батьківського каталогу». А тому в цій таблиці в меню *Indexes (Індекси)* режиму конструктора слід додати індекс без повторень так, як це показано на рисунку:



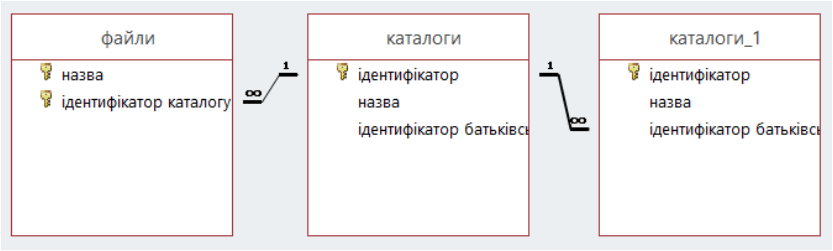
Для реалізації зв'язку між таблицями «файли» та «каталоги» з реляційної моделі після створення таблиць слід перетягнути поле «ідентифікатор» таблиці «каталоги» на відповідне йому поле зовнішнього ключа «ідентифікатор каталогу» таблиці «файли» та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення.

Аби відтворити рекурсивний зв'язок каталогу самим з собою слід вдруге додати таблицю «каталоги» на схему даних та створити зв'язок між зображеними на ній таблицями «каталоги» та «каталоги_1» так, ніби перша представляє батьківський каталог, а друга — дочірній. Іншими словами, слід перетягнути поле «ідентифікатор» таблиці «каталоги» на відповідне йому

поле зовнішнього ключа «ідентифікатор батьківського каталогу» таблиці «каталоги_1» та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення.

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№58. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Наступні поля таблиць повинні мати тип *Number (Число)*, оскільки передбачають числові дані:

- «код пісні» таблиці «країни»;
- «код пісні», «кількість балів» таблиці «глядачі оцінили»;
- «код» таблиці «члени журі»;
- «код» таблиці «пісні»;
- «член журі», «пісня» таблиці «оцінює»;
- «код», «код пісні» таблиці «виконавці».

Решта полів зберігають текстові дані, а тому їх тип — *Short Text (Короткий текст)*.

Примітка: поля «код» в таблицях «члени журі», «пісні» та «виконавці» можуть також мати тип AutoNumber (Автонумерація) *для полегшення подальшого вводу даних в таблиці.*

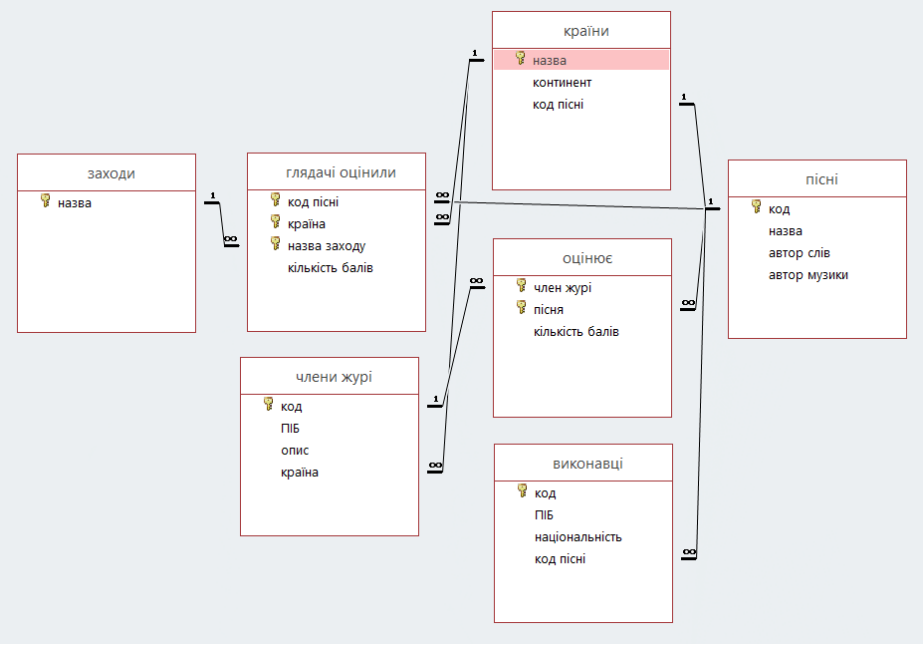
Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі Access. Для цього в кожній таблиці поля, що належать до первинного ключа, слід виділити із затиснутою клавішею Ctrl та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

Слід зазначити, що таблиця «країни» має окрім первинного також і додатковий потенційний ключ — поле «код пісні», яке встановлено унікальним для підтримання множинності зв'язку між країнами та піснями, якими вони представлені. А тому в цій таблиці слід виділити поле «код пісні» та встановити для нього значення *Yes (No Duplicates) — Так (Без повторень)* для властивості *Indexed (Індексоване поле)*.

Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення. Тут принциповим є перетягування поля «код» саме з таблиці «пісні» на поле «код пісні» таблиці «країни», адже це визначатиме основну та зв'язану таблиці в цьому зв'язку множинністю «один до одного».

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№59. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Наступні поля таблиць повинні мати тип *Number (Число)*, оскільки передбачають числові дані:

- «площа водної поверхні» таблиці «моря»;
- «довжина», «площа басейну» таблиці «річки».

Решта полів зберігають текстові дані, а тому їх тип — *Short Text (Короткий текст)*.

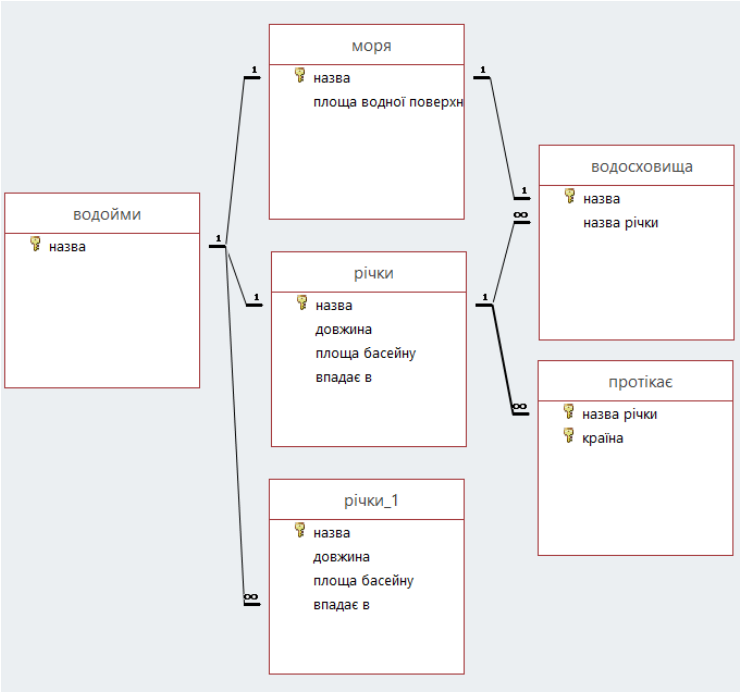
Первинний ключ кожної з таблиць, що складається в усіх таблицях з єдиного поля «назва», слід визначити в середовищі Access. Для цього в кожній таблиці поле «назва» слід виділити та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення. Тут принциповим є перетягування поля «назва» саме з таблиці «водойми» на поля «назва» таблиць «моря» і «річки», а також поля «назва» з таблиці «моря» на поле «назва» таблиці «водосховища», адже це визначатиме основну та зв'язану таблиці в цих зв'язках множинністю «один до одного».

Слід звернути увагу на те, що між таблицями «водойми» та «річки» на реляційній моделі подано одночасно два зв'язки. Оскільки це не підтримується СУБД Access напряму, для відтворення, наприклад, зв'язку «річка впадає у водойму» слід додати таблицю «річки» на схему даних вдруге. Отриманий дублікат цієї таблиці з назвою «річки_1» слід з'єднати з таблицею «водойми», перетягнувши поле «назва» таблиці «водойми» на поле «впадає в» таблиці «річки_1» та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення.

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№60. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access.

Наступні поля таблиць повинні мати тип *Number (Число)*, оскільки передбачають числові дані:

- «площа» таблиці «області»;
- «чисельність населення» таблиць «міста» та «держави».

Решта полів зберігають текстові дані, а тому їх тип — *Short Text (Короткий текст)*.

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі Access. Для цього в кожній таблиці поля, що належать до первинного ключа, слід виділити із затиснутою клавішею Ctrl та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

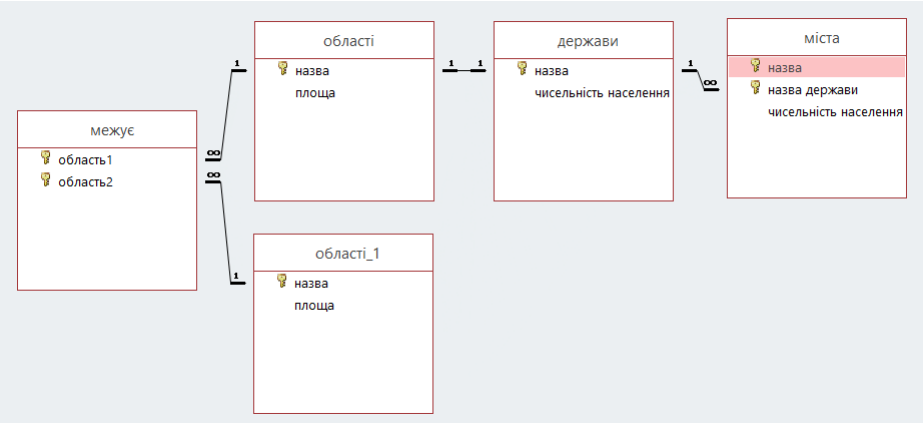
Для реалізації зв'язків з реляційної моделі після створення таблиць слід перетягнути поле основного ключа на відповідне йому поле зовнішнього

ключа та у новому вікні відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення. Тут принциповим є перетягування поля «назва» саме з таблиці «області» на поле «назва» таблиці «держави», адже це визначатиме основну та зв'язану таблиці в цьому зв'язку множинністю «один до одного».

Слід звернути увагу на те, що між таблицями «межує» та «області» на реляційній моделі подано одночасно два зв'язки. Оскільки це не підтримується СУБД Access напряму, для відтворення одного зі зв'язків слід додати таблицю «області» на схему даних вдруге. Отриманий дублікат цієї таблиці з назвою «області_1» слід з'єднати з таблицею «межує», перетягнувши поле «назва» таблиці «області_1» на поле «область2» таблиці «межує», а поле «назва» таблиці «області» при цьому перетягнути на поле «область1» таблиці «межує». Знов-таки, у новому вікні при створенні зв'язку слід відмітити прапорці, що відповідають за підтримку цілісності даних, їх каскадне оновлення та видалення.

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку Access не створюватиме зв'язок та відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:

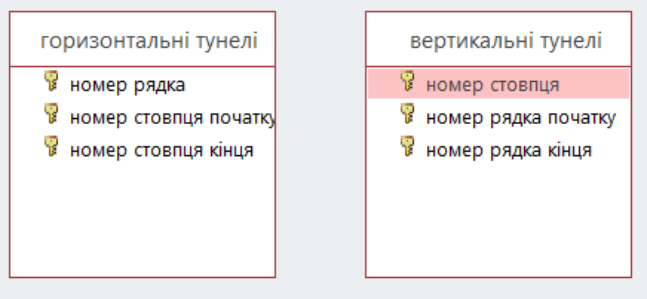


№61. Завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі конструктора СУБД Microsoft Access, оскільки зв'язки між таблицями відсутні.

Всі поля обох таблиць повинні мати тип *Number (Число)*, оскільки вони передбачають числові дані.

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі Access. Для цього в кожній таблиці поля, що належать до первинного ключа, слід виділити із затиснутою клавішею Ctrl та натиснути на інструмент *Primary Key (Первинний ключ)* для вказання первинного ключа таблиці.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№62. Виконання завдання зводиться в цілому до підбору типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Наступні поля повинні мати тип *VARCHAR(255)* або *TEXT*, оскільки передбачають текстові дані змінної довжини:

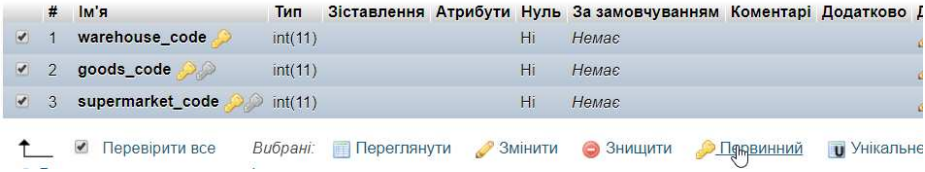
- «name», «address» таблиці «warehouses»;
- «name» таблиці «goods»;
- «name», «address» таблиці «supermarkets»;
- «name», «address» таблиці «manufacturers»;
- «last_name», «first_name», «middle_name», «qualification» таблиці «workers».

Поля «passport_no», «chief» таблиці «workers», «chief_passport_no» таблиці «warehouses» та «husband_passport_no», «wife_passport_no» таблиці «marriage» представляють номер паспорта, а тому є фіксованими рядками довжини 8 (дві літери і 6 цифр). Таким чином, найбільш зручним для них буде тип *CHAR(8)*. Поле «price» таблиці «goods» зберігає грошові дані, а тому повинне мати тип, наприклад, *DOUBLE(10,2)*.

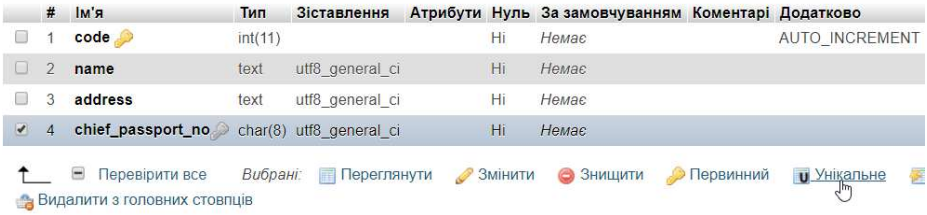
Поля «code» таблиць «warehouses», «manufacturers», «supermarkets» можуть мати тип *INT* з властивістю *A_I (автоматичний лічильник)*, якщо передбачається надання їм послідовних унікальних значень. Це полегшить процес введення даних в таблиці.

Решта полів зберігають числові дані, а тому їх тип — *INT*.

Первинний ключ кожної з таблиць, який на реляційній моделі підкреслено, слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці, як це показано на рисунку для таблиці «delivered»:



Налаштування унікальних полів у застосунку phpMyAdmin найлегше теж здійснюється в поданні структури таблиці вже після її створення. Для цього всі поля таблиці, що входять до унікального ключа, слід помітити прапорцем та натиснути на інструмент *Унікальне* під переліком полів таблиці для вказання унікального ключа таблиці, як це показано на рисунку для таблиці «warehouses»:



Аналогічні дії потрібно виконати і для полів «husband_passport_no», «wife_passport_no» таблиці «marriage».

№63.

Для полів таблиці «entrants» слід налаштувати такі параметри:

- поля «last_name», «first_name», «middle_name» повинні мати тип *VARCHAR (10)*, оскільки передбачають введення тексту довжиною до 10 символів;
- поле «birthdate» повинно мати тип *DATE*;
- для поля «test_certificate_no» достатньо типу *MEDIUMINT(6)*, оскільки воно передбачає введення шестицифрового числа;

- поле «email» повинно мати тип *VARCHAR(255)*, оскільки про його довжину нічого не відомо, але за ним вимагається створення унікального ключа (а за полями типу *TEXT*, як відомо, ключ створити в СУБД MySQL не дозволяється);
- поле «medical_certificate» повинно мати тип *TEXT*, оскільки про його довжину нічого не відомо;
- поле «certificate_series» повинно мати тип *CHAR(2)*, оскільки воно передбачає введення рівно двох літер;
- для поля «certificate_no» достатньо типу *MEDIUMINT(6)*, оскільки воно передбачає введення шестицифрового числа;
- поле «average_score» повинно мати тип *DOUBLE(3,1)*, оскільки максимальна загальна кількість цифр збережуваного числа для цього поля становить $2 + 1 = 3$, а кількість цифр після десяткової коми — 1.

За умовою вправи, потенційними ключами таблиці є:

- поле «test_certificate_no»;
- поле «email»;
- комбінація полів «certificate_series» та «certificate_no».

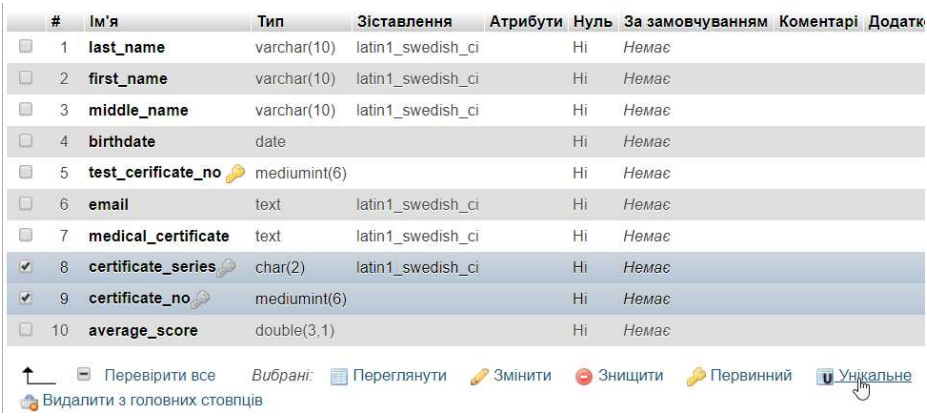
Оберемо серед них первинним ключем, наприклад, поле «test_certificate_no». Тоді в поданні структури таблиці його слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці так, як це показано на рисунку:

	#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Ком
☐	1	last_name	varchar(10)	latin1_swedish_ci		Ні	Немає	
☐	2	first_name	varchar(10)	latin1_swedish_ci		Ні	Немає	
☐	3	middle_name	varchar(10)	latin1_swedish_ci		Ні	Немає	
☐	4	birthdate	date			Ні	Немає	
☑	5	test_cerificate_no	mediumint(6)			Ні	Немає	
☐	6	email	text	latin1_swedish_ci		Ні	Немає	
☐	7	medical_certificate	text	latin1_swedish_ci		Ні	Немає	
☐	8	certificate_series	char(2)	latin1_swedish_ci		Ні	Немає	
☐	9	certificate_no	mediumint(6)			Ні	Немає	
☐	10	average_score	double(3,1)			Ні	Немає	

Перевірити все *Вибрані:* Переглянути Змінити Знищити Первинний
Видалити з головних стовпців

Решту ключів слід почергово налаштувати, виділивши їх поля за допомогою прапорців та скориставшись інструментом *Унікальне* під переліком

полів таблиці для вказання унікального ключа таблиці так, як це показано на рисунку:



	#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатк
☐	1	last_name	varchar(10)	latin1_swedish_ci		Ні	Немає		
☐	2	first_name	varchar(10)	latin1_swedish_ci		Ні	Немає		
☐	3	middle_name	varchar(10)	latin1_swedish_ci		Ні	Немає		
☐	4	birthdate	date			Ні	Немає		
☐	5	test_cerificate_no	mediumint(6)			Ні	Немає		
☐	6	email	text	latin1_swedish_ci		Ні	Немає		
☐	7	medical_certificate	text	latin1_swedish_ci		Ні	Немає		
☑	8	certificate_series	char(2)	latin1_swedish_ci		Ні	Немає		
☑	9	certificate_no	mediumint(6)			Ні	Немає		
☐	10	average_score	double(3,1)			Ні	Немає		

Перевірити все *Вибрані:* Переглянути Змінити Знищити Первинний Унікальне
Видалити з головних стовпців

Після цього в блоці *Індекси* подання структури таблиці буде відображено наступні створені нами індекси:

Дія	Назва ключа	Тип	Унікальне	Запакований	Стовпець
Редагувати Знищити	PRIMARY	BTREE	Так	Ні	test_cerificate_no
Редагувати Знищити	certificate_series	BTREE	Так	Ні	certificate_series
					certificate_no
Редагувати Знищити	email	BTREE	Так	Ні	email

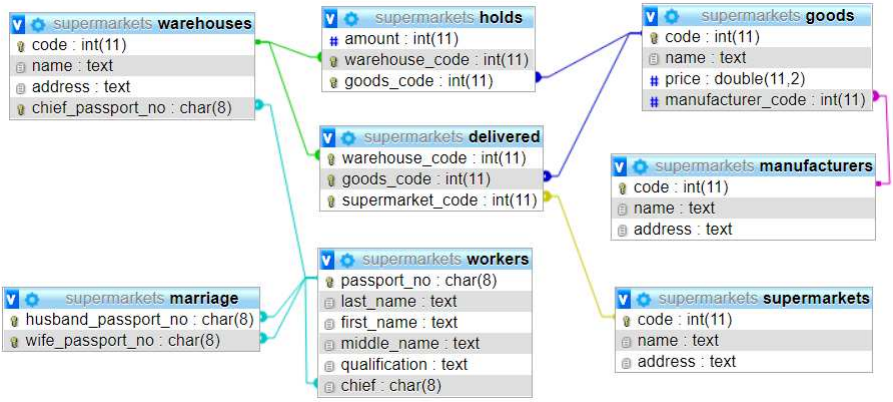
№64. Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
warehouses	chief_passport_no	workers	passport_no
marriage	husband_passport_no	workers	passport_no
marriage	wife_passport_no	workers	passport_no
holds	warehouse_code	warehouses	code
holds	goods_code	goods	code
delivered	warehouse_code	warehouses	code
delivered	goods_code	goods	code
delivered	supermarket_code	supermarkets	code
workers	chief	workers	passport_no
goods	manufacturer_code	manufacturers	code

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№65. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Наступні поля повинні мати тип *TEXT* або *VARCHAR*, оскільки передбачають текстові дані змінної довжини:

- «name», «location» таблиці «power_plants»;
- «type» таблиці «hydroelectric_power_plants»;

- «name» таблиці «rivers»;
- «river_name» таблиці «placed_on»;
- «reactor_type» таблиці «power_units».

Поле «operationality» таблиці «nuclear_power_plants» зберігає логічне значення, а тому повинне мати тип *TINYINT(1)* — він дозволяє зберігати значення 0 (ХИБНІСТЬ) або 1 (ІСТИНА).

Поле «id» таблиці «електростанції» може мати тип *INT* з властивістю *A_I (автоматичний лічильник)* (вона встановлюється прапорцем навпроти поля у відповідному стовпці при створенні таблиці), якщо передбачається надання електростанціям послідовних унікальних значень. Це полегшить процес введення даних в таблиці. З іншого боку це ж поле в таблицях «hydroelectric_power_plants» та «nuclear_power_plants» є зовнішнім ключем і не може мати аналогічний тип значень, оскільки використовує значення ідентифікаторів електростанцій, призначені їм в таблиці «power_plants».

Решта полів зберігають числові дані, а тому їх тип — *INT*.

Первинний ключ кожної з таблиць слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

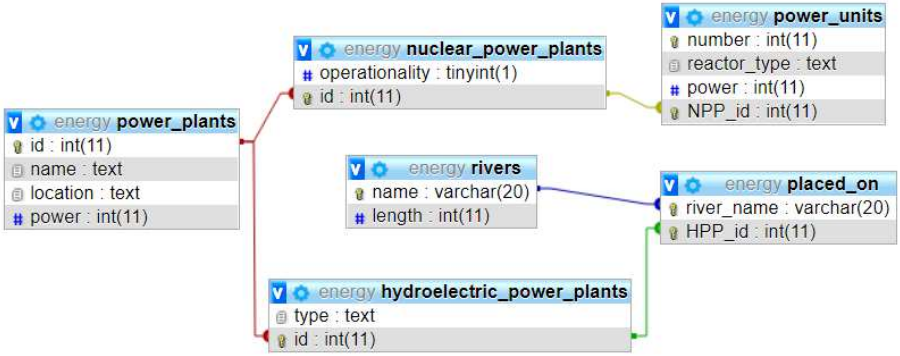
Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
hydroelectric_power_plants	id	power_plants	id
nuclear_power_plants	id	power_plants	id
power_units	NPP_id	nuclear_power_plants	id
placed_on	NPP_id	hydroelectric_power_plants	id
placed_on	river_name	rivers	name

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№66. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Поля «id» (ідентифікатор) всіх таблиць повинні мати тип *INT*, оскільки передбачають числові дані. Решта полів зберігають текстові дані змінної довжини, а тому їх тип — *TEXT* або *VARCHAR*.

Зверніть увагу, що СУБД MySQL не передбачає збереження файлів в полях бази даних. Зазвичай в промислових програмних продуктах в базі даних зберігають лише шлях до відповідного файлу на диску.

Примітка: поле «id» в таблиці «puzzles» може мати властивість A_I (автоматичний лічильник) *для полегшення подальшого вводу даних в таблицю. Таблиці-різновиди використовують ідентифікатор головоломки, визначений в таблиці «головоломки», а тому мати властивість* A_I (автоматичний лічильник) *не можуть.*

Первинний ключ кожної з таблиць, а це поля «id», слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне

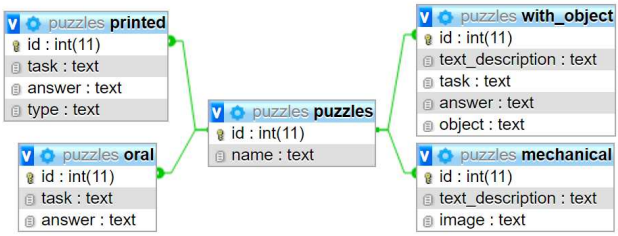
видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
printed	id	puzzles	id
oral	id	puzzles	id
with_object	id	puzzles	id
mechanical	id	puzzles	id

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№67. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Поля «number» (число) всіх таблиць, а також поля «sequence_number» (порядковий номер) повинні мати тип *INT*, оскільки передбачають числові дані. Решта полів зберігають текстові дані змінної довжини, а тому їх тип — *TEXT* або *VARCHAR*.

Первинний ключ кожної з таблиць слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Слід зазначити, що таблиці «fibonacci_numbers» і «lucas_numbers» мають окрім первинного також і потенційний ключ «sequence_number». А то-

му в кожній з цих таблиць в поданні структури слід відмітити це поле прапорцем та натиснути на інструмент *Унікальне* під переліком полів таблиці для вказання унікального ключа таблиці.

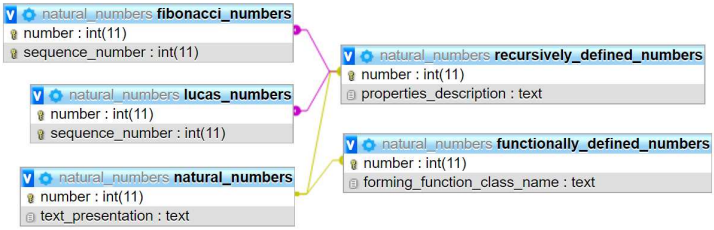
Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
recursively_defined_numbers	number	natural_numbers	number
functionally_defined_numbers	number	natural_numbers	number
fibonacci_numbers	number	recursively_defined_numbers	number
lucas_numbers	number	recursively_defined_numbers	number

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№68. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Наступні поля таблиць повинні мати тип *INT*, оскільки передбачають числові дані:

- «class_room» таблиці «classes»;

- «cabinet» таблиці «class_teachers»;
- «lesson_no» таблиці «schedule».

Примітка: номер особової справи учня зазвичай складається з першої літери прізвища та певного числа, тому не може вважатися числом.

Наступні поля таблиць повинні мати тип *VARCHAR*, оскільки передбачають текстові дані змінної довжини і є частинами ключів або відповідних їм зовнішніх ключів:

- «name» таблиці «classes»;
- «class_name» таблиці «class_teachers»;
- «personal_file_number», «class_name» таблиці «students»;
- «name» таблиці «subjects»;
- «class_name», «subject_name», «weekday» таблиці «schedule».

Наступні поля таблиць повинні мати тип *CHAR(8)*, оскільки передбачають текстові дані фіксованої довжини:

- «passport_no» таблиць «class_teachers» та «teachers»;
- «teachers_passport_no» таблиці «schedule».

Решта полів зберігають текстові дані змінної довжини, а тому їх тип — *TEXT* або *VARCHAR*.

Первинний ключ кожної з таблиць слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Слід зазначити, що таблиця «schedule» матиме (у зв'язку з додатковими умовами на розклад занять) окрім первинного також і додатковий потенційний ключ: комбінацію полів «weekday», «teacher_passport_no», «lesson_no».

А тому в таблиці «schedule» в поданні структури для цього унікального ключа слід відмітити його поля прапорцем та натиснути на інструмент *Унікальне* під переліком полів для вказання унікального ключа таблиці.

При коректному заданні унікального ключа в блоці *Індекси* подання структури таблиці буде відображено наступні створені нами індекси:

Дія	Назва ключа	Тип	Унікальне	Запакований	Стовпець	Кількість елементів	Зіставлення	Нуль	Коментар
Редагувати Знищити	PRIMARY	BTREE	Так	Ні	class_name	115	A	Ні	
					lesson_no	115	A	Ні	
					weekday	115	A	Ні	
Редагувати Знищити	teacher_passport_no	BTREE	Так	Ні	teacher_passport_no	115	A	Ні	
					lesson_no	115	A	Ні	
					weekday	115	A	Ні	

Також, таблиця «class_teachers» містить унікальне поле «class_name», необхідне для підтримання множинності зв'язку між класами та їх керів-

никами. А тому в цій таблиці в поданні структури теж слід відмітити ці поля прапорцем та натиснути на інструмент *Унікальне* під переліком полів таблиці для встановлення унікального ключа таблиці.

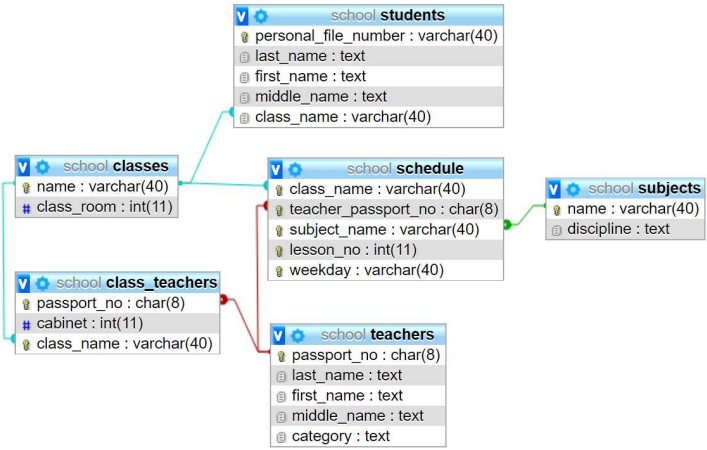
Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
class_teachers	class_name	classes	name
class_teachers	passport_no	teachers	passport_no
students	class_name	classes	name
schedule	class_name	classes	name
schedule	teacher_passport_no	teachers	passport_no
schedule	subject_name	subjects	name

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№69. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

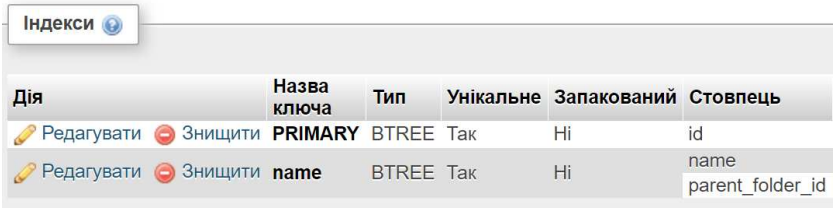
Поля «name» обох таблиць повинні мати тип *VARCHAR*, оскільки вони передбачають текстові дані змінної довжини та входять до первинного та унікального ключів таблиць. Решта полів зберігають числові дані, а тому їх тип — *INT*.

Примітка: поле «id» в таблиці «folders» може мати властивість A_I (автоматичний лічильник) *для полегшення подальшого вводу даних в таблицю.*

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Слід зазначити, що таблиця «folders» має окрім первинного також і додатковий потенційний ключ — комбінацію полів «name» та «parent_folder_id». А тому в цій таблиці в поданні структури слід відмітити це поле прапорцем та натиснути на інструмент *Унікальне* під переліком полів таблиці для вказання унікального ключа таблиці.

При коректному заданні унікального ключа в блоці *Індекси* подання структури таблиці буде відображено наступні створені нами індекси:


Індекси

Дія	Назва ключа	Тип	Унікальне	Запакований	Стовпець
Редагувати Знищити	PRIMARY	BTREE	Так	Ні	id
Редагувати Знищити	name	BTREE	Так	Ні	name
					parent_folder_id

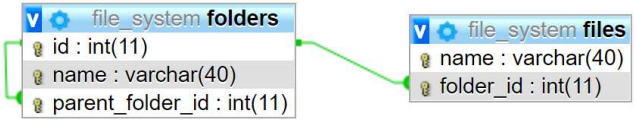
Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
files	folder_id	folders	id
folders	parent_folder_id	folders	id

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№70. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Наступні поля таблиць повинні мати тип *INT*, оскільки передбачають числові дані:

- «song_id» таблиці «countries»;
- «song_id», «points» таблиці «audience_voted»;
- «id» таблиці «jury_members»;
- «id» таблиці «songs»;
- «jury_member», «song» таблиці «votes»;
- «id», «song_id» таблиці «singers».

Поля «name» таблиць «countries» та «events», поля «country», «event» таблиці «audience_voted» та поле «country» таблиці «jury_members» повинні мати тип *VARCHAR*, оскільки вони передбачають текстові дані змінної довжини та входять до первинних ключів таблиць або відповідних їм зовнішніх ключів. Решта полів зберігають текстові дані змінної довжини, а тому їх тип — *TEXT* або *VARCHAR*.

Примітка: поля «id» в таблицях «jury_members», «songs» та «singers» можуть мати властивість A_I (автоматичний лічильник) *для полегшення подальшого вводу даних в таблицю.*

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі phpMyAdmin. Найлегше це

робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Слід зазначити, що таблиця «countries» має окрім первинного також і додатковий потенційний ключ — поле «song_id», яке встановлено унікальним для підтримання множинності зв'язку між країнами та піснями, якими вони представлені. А тому в цій таблиці в поданні структури слід відмітити це поле прапорцем та натиснути на інструмент *Унікальне* під переліком полів таблиці для вказання унікального ключа таблиці.

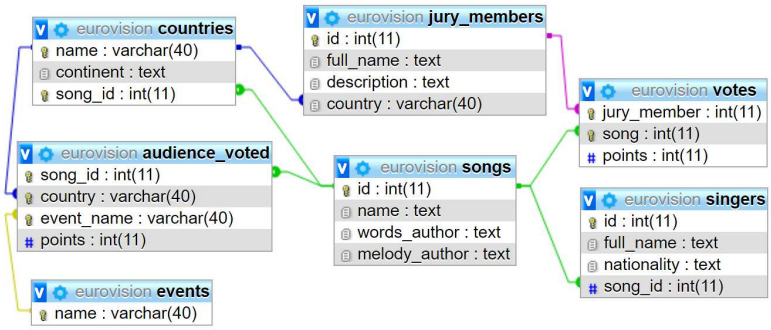
Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
countries	song_id	songs	id
audience_voted	song_id	songs	id
audience_voted	country	countries	name
audience_voted	event_name	events	name
jury_members	country	countries	name
votes	jury_member	jury_members	id
votes	song	songs	id
singers	song_id	songs	id

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№71. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Наступні поля таблиць повинні мати тип *INT*, оскільки передбачають числові дані:

- «water_surface_area» таблиці «seas»;
- «length», «basin_area» таблиці «rivers».

Решта полів мають тип *VARCHAR*, оскільки вони передбачають текстові дані змінної довжини та входять до первинних ключів таблиць або відповідних їм зовнішніх ключів.

Первинний ключ кожної з таблиць, а це поля «name», слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
seas	name	waterbodies	name
rivers	name	waterbodies	name
rivers	falls_into	waterbodies	name
reservoirs	name	seas	name
reservoirs	river_name	rivers	name
flows	river_name	rivers	name

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:

№72. Перша частина завдання полягає у підборі типів полів таблиць з реляційної моделі та створення цих таблиць в середовищі програмного засобу phpMyAdmin.

Наступні поля таблиць повинні мати тип *INT*, оскільки передбачають числові дані:

- «area» таблиці «regions»;
- «population» таблиць «cities» та «countries».

Решта полів мають тип *VARCHAR*, оскільки вони передбачають текстові дані змінної довжини та входять до первинних ключів таблиць або відповідних їм зовнішніх ключів.

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

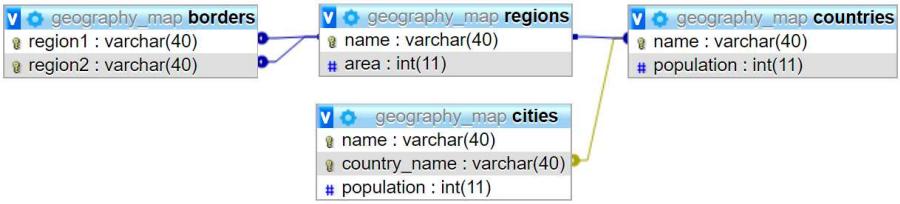
Для реалізації зв'язків з реляційної моделі після створення таблиць слід для кожного зовнішнього ключа у вікні структури таблиці в поданні відносин в блоці *Обмеження зовнішнього ключа* вказати поле іншої таблиці, якому цей зовнішній ключ відповідає. Також можна налаштувати каскадне видалення та оновлення даних, обравши у відповідних розкривних списках *ON DELETE* та *ON UPDATE* опцію *CASCADE*.

В наступній таблиці подано перелік обмежень зовнішніх ключів, які необхідно встановити:

Зв'язана таблиця	*FK*	*Основна таблиця*	*Ключ*
borders	region1	regions	name
borders	region2	regions	name
countries	name	regions	name
cities	country_name	countries	name

Під час створення зв'язків обов'язково переконайтеся, що пов'язувані поля мають однакові або сумісні типи. В протилежному випадку MySQL не створюватиме зв'язок та застосунок phpMyAdmin відображатиме відповідну помилку.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:

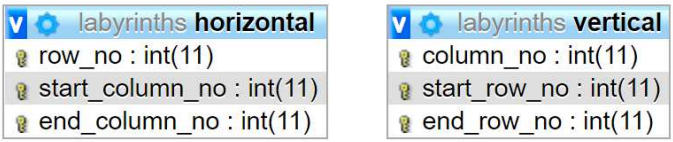


№73. Завдання полягає у підборі типів полів таблиць з реляційної моделі та створенні цих таблиць в середовищі програмного засобу phpMyAdmin.

Всі поля обох таблиць повинні мати тип *INT*, оскільки вони передбачають числові дані.

Первинний ключ кожної з таблиць, підкреслений на реляційній моделі предметної області, слід визначити в середовищі phpMyAdmin. Найлегше це робити вже після створення таблиці з відповідними їй полями. Для цього всі поля таблиці, що входять до первинного ключа, слід помітити прапорцем та натиснути на інструмент *Первинний* під переліком полів таблиці для вказання первинного ключа таблиці.

Якщо базу даних було реалізовано коректно, схема даних повинна виглядати наступним чином:



№74.

1. Можна навести такі приклади аномалій оновлення:

   - До таблиці «гідроелектростанції» додали Кременчуцьку ГЕС, розташовану на Дніпрі, але при додаванні випадково вказали помилкову довжину річки: 2021. Виникає неоднозначність у довжині річки Дніпро.
   - З таблиці «гідроелектростанції» прибрали дані про Дністровську ГАЕС, розташовану на річці Дністер. Разом з електростанцією було прибрано і дані про довжину річки.
   - В таблиці «гідроелектростанції» оновили дані про довжину річки Дніпро: вона стала коротшою на 1 км. Проте при внесенні змін до таблиці випадково змінили довжину Дніпра лише в записі про Канівську ГЕС. Виникає неоднозначність у довжині Дніпра.
   - До таблиці «атомні електростанції» додали новий (третій) реактор Хмельницької АЕС, але при додаванні випадково вказали помилкову потужність АЕС: 2200. Виникає неоднозначність у потужності Хмельницької АЕС.
   - З таблиці «атомні електростанції» прибрали дані про реактори Хмельницької АЕС. Разом з реакторами було прибрано і дані про саму АЕС.

2. З умови та даних таблиць випливає наявність таких повних функціональних залежностей:

   - таблиця «гідроелектростанції»:
     - «ідентифікатор електростанції» → «назва електростанції», «розташування», «потужність», «тип»;
     - «назва річки» → «довжина»;
   - таблиця «атомні електростанції»:

– «ідентифікатор електростанції» → «назва електростанції», «розташування», «потужність», «діючість»;
– «ідентифікатор електростанції», «номер реактора» → «тип реактора», «потужність».

3. Розглянемо почергово кроки нормалізації для кожної з поданих таблиць.

Таблиця «гідроелектростанції» подана в ненормалізованій формі, оскільки містить об'єднання комірок — на перетині рядка і стовпця тут подекуди розташовані два значення. Аби привести її до першої нормальної форми, слід розбити рядки з багатозначними полями та продублювати неповторювані дані (тут — дані про власне АЕС: «ідентифікатор електростанції», «назва електростанції», «розташування», «потужність», «тип»). На виході цієї дії отримаємо таблицю в *1НФ* наступного вигляду:

ідентифікатор електростанції	назва електростанції	розташування	потужність	тип	назва річки	довжина
1	Канівська ГЕС	Канів	444	руслово-греблева	Дніпро	2201
2	Київська ГАЕС	Нові Петрівці	235,5	гідроакумулююча	Дніпро	2201
3	Дністровська ГАЕС	Розкопинці	324	гідроакумулююча	Дністер	1362
4	Теребле-Ріцька ГЕС	Хустський район	27	деривaційна	Тересбля	91
4	Теребле-Ріцька ГЕС	Хустський район	27	деривaційна	Ріка	92

Ключем цієї таблиці є комбінація полів «ідентифікатор електростанції» та «назва річки».

Таблиця в другій нормальній формі не повинна містити неключових полів, що перебувають у повній функціональній залежності від частин ключа, а не від всього ключа. В нашому випадку, поля «назва електростанції», «розташування», «потужність», «тип» залежать лише від поля «ідентифікатор електростанції», а поле «довжина» — лише від поля «назва річки». Тому нам слід винести ці функціональні залежності у окремі таблиці разом з копією їх лівої частини. Отримаємо дві нові таблиці у *2НФ* наступного вигляду («гідроелектростанції» з ключовим полем «ідентифікатор електростанції» та «річки» з ключовим полем «назва річки» відповідно):

ідентифікатор електростанції	назва електростанції	розташування	потужність	тип
1	Канівська ГЕС	Канів	444	руслово-греблева
2	Київська ГАЕС	Нові Петрівці	235,5	гідроакумулююча
3	Дністровська ГАЕС	Розкопинці	324	гідроакумулююча
4	Теребле-Ріцька ГЕС	Хустський район	27	деривaційна

назва річки	довжина
Дніпро	2201
Дністер	1362
Теребля	91
Ріка	92

Початкова таблиця у *2НФ* набуде вигляду («розташування»):

ідентифікатор електростанції	назва річки
1	Дніпро
2	Дніпро
3	Дністер
4	Теребля
4	Ріка

Відповідно, реляційна модель на цьому етапі матиме наступний вигляд:

Неважко переконатися, що в цих таблицях відсутні транзитивні функціональні залежності, а отже вони автоматично перебувають у *третій нормальній формі*.

Таблиця «атомні електростанції» подана в ненормалізованій формі, оскільки містить об'єднання комірок — на перетині рядка і стовпця тут подекуди розташовані два значення. Аби привести її до першої нормальної форми, слід розбити рядки з багатозначними полями та продублювати неповторювані дані (тут — дані про власне АЕС: «ідентифікатор електростанції», «назва електростанції», «розташування», «потужність», «діючість»). На виході цієї дії отримаємо таблицю в *1НФ* наступного вигляду:

ідентифікатор електростанції	назва електростанції	розташування	потужність	діючість	номер реактора	тип реактора	потужність
1	Чорнобильська АЕС	Прип'ять	3800	ні	1	РБМК-1000	1000
1	Чорнобильська АЕС	Прип'ять	3800	ні	2	РБМК-1000	1000
1	Чорнобильська АЕС	Прип'ять	3800	ні	3	РБМК-1000	1000
1	Чорнобильська АЕС	Прип'ять	3800	ні	4	РБМК-1000	1000
2	Хмельницька АЕС	Нетішин	2000	так	1	ВВЕР-1000	1000
2	Хмельницька АЕС	Нетішин	2000	так	2	ВВЕР-1000	1000
3	Рівненська АЕС	Вараш	2835	так	1	ВВЕР-440	440
3	Рівненська АЕС	Вараш	2835	так	2	ВВЕР-440	440
3	Рівненська АЕС	Вараш	2835	так	3	ВВЕР-1000	1000
3	Рівненська АЕС	Вараш	2835	так	4	ВВЕР-1000	1000

Ключем цієї таблиці є комбінація полів «ідентифікатор електростанції» та «номер реактора».

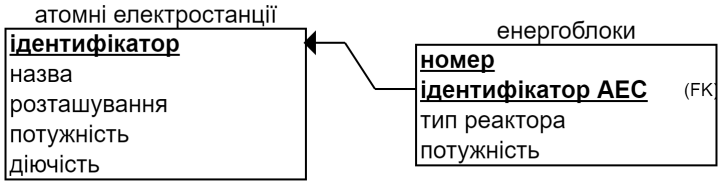
Таблиця в другій нормальній формі не повинна містити неключових полів, що перебувають у повній функціональній залежності від частин ключа, а не від всього ключа. В нашому випадку, поля «назва електростанції», «розташування», «потужність», «діючість» залежать лише від поля «ідентифікатор електростанції». Тому нам слід винести цю функціональну залежність у окрему таблицю разом з копією її лівої частини. Отримаємо нову таблицю у *2НФ* з ключовим полем «ідентифікатор електростанції» наступного вигляду («атомні електростанції»):

ідентифікатор електростанції	назва електростанції	розташування	потужність	діючість
1	Чорнобильська АЕС	Прип'ять	3800	ні
2	Хмельницька АЕС	Нетішин	2000	так
3	Рівненська АЕС	Вараш	2835	так

Початкова таблиця у *2НФ* набуде вигляду («реактори»):

ідентифікатор електростанції	номер реактора	тип реактора	потужність
1	1	РБМК-1000	1000
1	2	РБМК-1000	1000
1	3	РБМК-1000	1000
1	4	РБМК-1000	1000
2	1	ВВЕР-1000	1000
2	2	ВВЕР-1000	1000
3	1	ВВЕР-440	440
3	2	ВВЕР-440	440
3	3	ВВЕР-1000	1000
3	4	ВВЕР-1000	1000

Відповідно, реляційна модель на цьому етапі матиме наступний вигляд:



Неважко переконатися, що в цих таблицях відсутні транзитивні функціональні залежності, а отже вони автоматично перебувають у *третій нормальній формі*.

Аби отримати загальну реляційну модель бази даних предметної області «Енергетика», слід сумістити дві реляційні моделі, подані вище.

4. В цьому випадку в таблиці виникне ще одна повна функціональна залежність: «тип реактора» → «потужність». Якщо до другої нормальної форми включно процес нормалізації цієї таблиці змін не зазнає, то таблиця «реактори», зображена вище, вже не відповідатиме третій нормальній формі через транзитивну залежність поля «потужність» від ключа: «ідентифікатор електростанції», «номер реактора» → «тип реактора» → «потужність».

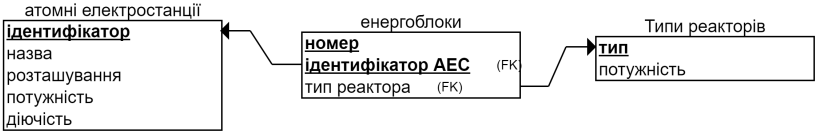
   Для приведення цієї таблиці в третю нормальну форму слід винести поле «потужність» разом з полем, через яке воно транзитивно залежить від ключа («тип реактора») в нову таблицю з ключовим полем «тип реактора» у *3НФ* («типи реакторів»):

тип реактора	потужність
РБМК-1000	1000
ВВЕР-1000	1000
ВВЕР-440	440

   Таблиця «реактори» у *3НФ* набуде вигляду:

ідентифікатор електростанції	номер реактора	тип реактора
1	1	РБМК-1000
1	2	РБМК-1000
1	3	РБМК-1000
1	4	РБМК-1000
2	1	ВВЕР-1000
2	2	ВВЕР-1000
3	1	ВВЕР-440
3	2	ВВЕР-440
3	3	ВВЕР-1000
3	4	ВВЕР-1000

Відповідно, реляційна модель в цьому випадку матиме вигляд:



№75.

1. Можна навести такі приклади аномалій оновлення:
   - До таблиці додали інформацію про гостя Larry BF, який поселився в готелі «Хілтон», але допустили помилку в поштовому індексі цього готелю, увівши 83833 замість 83835. Виникає неоднозначність в поштовому індексі готелю «Хілтон».
   - З таблиці прибрали дані гостя Brad VZ, який поселився в готелі «Хілтон». Разом з ним було прибрано і дані про готель «Хілтон».
   - Нехай в таблиці також зазначено, що гість Larry BF проживає в королівському номері 635 з 1 по 3 липня 2018 року. В таблиці оновили дані про тип кімнати 635: вона стала президентським люксом. Проте при внесенні змін до таблиці випадково змінили тип кімнати лише в записі про гостя Brad VZ. Виникає неоднозначність у типі кімнати 635.
2. З умови та даних таблиць випливає наявність таких повних функціональних залежностей:
   - «номер готелю» → «назва готелю», «поштовий індекс готелю», «місто розташування готелю»;
   - «поштовий індекс готелю» → «місто розташування готелю»;
   - «поштовий індекс гостя» → «місто проживання гостя»;
   - «номер гостя» → «ім'я гостя», «поштовий індекс гостя», «місто проживання гостя»;
   - «номер кімнати», «номер готелю» → «тип кімнати», «ціна кімнати»;
   - «номер гостя», «дата заселення» → «номер готелю», «назва готелю», «поштовий індекс готелю», «місто розташування готелю», «ім'я гостя», «поштовий індекс гостя», «місто проживання гостя», «дата виселення», «номер кімнати», «тип кімнати», «ціна кімнати»;
   - «номер гостя», «дата виселення» → «номер готелю», «назва готелю», «поштовий індекс готелю», «місто розташування готелю»,

«ім'я гостя», «поштовий індекс гостя», «місто проживання гостя», «дата заселення», «номер кімнати», «тип кімнати», «ціна кімнати»;
- «номер готелю», «номер кімнати», «дата заселення» → «назва готелю», «поштовий індекс готелю», «місто розташування готелю», «номер гостя», «ім'я гостя», «поштовий індекс гостя», «місто проживання гостя», «дата виселення», «тип кімнати», «ціна кімнати»;
- «номер готелю», «номер кімнати», «дата виселення» → «назва готелю», «поштовий індекс готелю», «місто розташування готелю», «номер гостя», «ім'я гостя», «поштовий індекс гостя», «місто проживання гостя», «дата заселення», «тип кімнати», «ціна кімнати».

Слід зазначити, що подана в умові таблиця має кілька потенційних ключів:

- «номер гостя», «дата заселення»;
- «номер гостя», «дата виселення»;
- «номер готелю», «номер кімнати», «дата заселення»;
- «номер готелю», «номер кімнати», «дата виселення».

3. Розглянемо почергово кроки нормалізації для цієї таблиці.
Таблиця вже подана в першій нормальній формі, оскільки не містить об'єднання комірок та повторюваних груп полів.
Таблиця в другій нормальній формі не повинна містити неключових полів, що перебувають у повній функціональній залежності від частин принаймні одного з потенційних ключів, а не від всього потенційного ключа. В нашому випадку, три повні функціональні залежності містять в своїй лівій частині підмножину полів деякого потенційного ключа:

- «номер готелю» → «назва готелю», «поштовий індекс готелю», «місто розташування готелю» («номер готелю» — підмножина потенційного ключа «номер готелю», «номер кімнати», «дата заселення»);
- «номер гостя» → «ім'я гостя», «поштовий індекс гостя», «місто проживання гостя» («номер гостя» — підмножина потенційного ключа «номер гостя», «дата заселення»);
- «номер кімнати», «номер готелю» → «тип кімнати», «ціна кімнати» («номер кімнати», «номер готелю» — підмножина потенційного ключа «номер готелю», «номер кімнати», «дата заселення»).

Тому нам слід винести ці функціональні залежності у окремі таблиці

разом з копією їх лівої частини. Отримаємо три нові таблиці у *2НФ* наступного вигляду («готелі» з ключовим полем «номер готелю», «гості» з ключовим полем «номер гостя» та «кімнати» з ключем з двох полів «номер кімнати» та «номер готелю» відповідно):

Номер готелю	Назва готелю	Місто розташування готелю	Поштовий індекс готелю
3	Хілтон	Сан-Дієго	83835

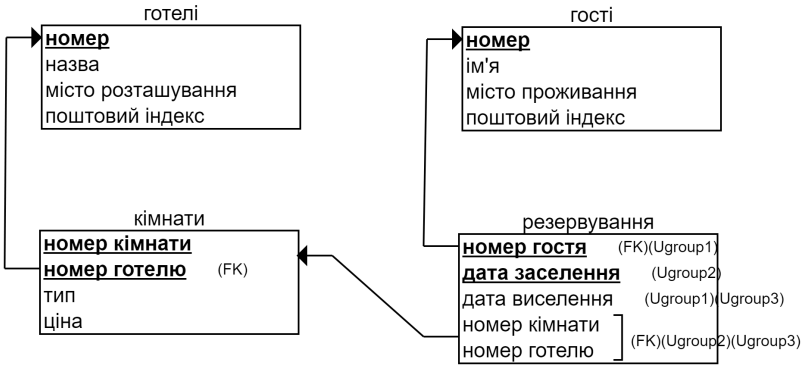
Номер гостя	Ім'я гостя	Місто проживання гостя	Поштовий індекс гостя
232	Brad VZ	Ноксвілл	37996

Номер готелю	Номер кімнати	Тип кімнати	Ціна кімнати
3	635	Королівський	89,99

Початкова таблиця у *2НФ* набуде вигляду:

Номер готелю	Номер гостя	Дата заселення	Дата виселення	Номер кімнати
3	232	22.06.2018	27.06.2018	635

Відповідно, реляційна модель на цьому етапі матиме наступний вигляд:



Таблиця у третій нормальній формі не повинна мати транзитивних залежностей неключових полів від ключа. Неважко переконатися, що таблиця «кімнати» таких залежностей не містить, а тому вже перебуває в третій нормальній формі. Таблиця «резервування» теж відповідає третій нормальній формі, оскільки взагалі не містить неключових полів.

Таблиця «готелі» містить одну транзитивну залежність від ключа: «номер готелю» → «поштовий індекс готелю» → «місто розташування готелю». Для приведення цієї таблиці в третю нормальну форму слід винести поле «місто розташування готелю» разом з полем, через яке воно транзитивно залежить від ключа («поштовий індекс готелю») в нову таблицю з ключовим полем «поштовий індекс» у *3НФ* («поштові індекси готелів»):

Поштовий індекс	Місто
83835	Сан-Дієго

Таблиця «готелі» у *3НФ* набуде вигляду:

Номер готелю	Назва готелю	Поштовий індекс готелю
3	Хілтон	83835

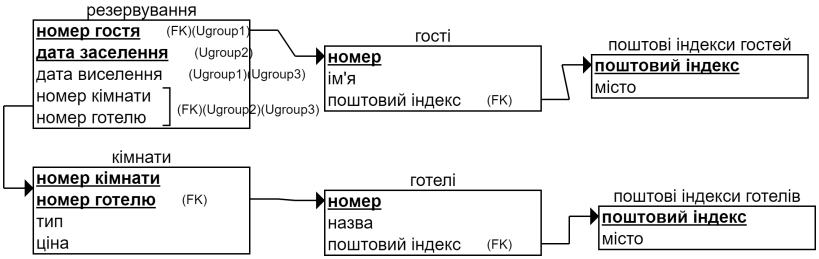
Таблиця «гості» теж містить одну транзитивну залежність від ключа: «номер гостя» → «поштовий індекс гостя» → «місто проживання гостя». Для приведення цієї таблиці в третю нормальну форму слід винести поле «місто проживання гостя» разом з полем, через яке воно транзитивно залежить від ключа («поштовий індекс гостя») в нову таблицю з ключовим полем «поштовий індекс» у *3НФ* («поштові індекси гостей»):

Поштовий індекс	Місто
37996	Ноксвілл

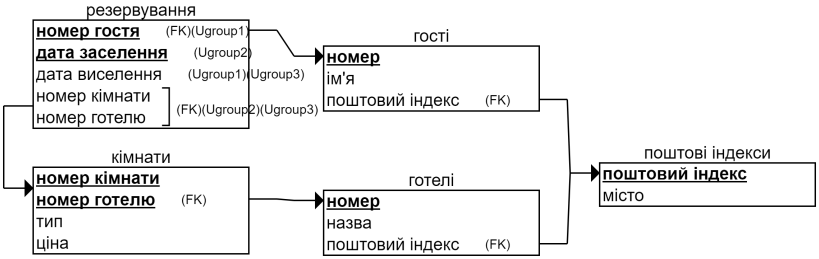
Таблиця «гості» у *3НФ* набуде вигляду:

Номер гостя	Ім'я гостя	Поштовий індекс гостя
232	Brad VZ	37996

Відповідно, реляційна модель на цьому етапі матиме наступний вигляд:



Зверніть увагу, що таблиці «поштові індекси готелів» та «поштові індекси гостей» насправді зберігають одну й ту ж інформацію: який індекс якому місту відповідає. Тому цілком доцільно об'єднати їх в одну таблицю «поштові індекси» з відповідними змінами на логічній схемі бази даних:



№76.

1. Ненормалізована таблиця, в якій зібрано всі дані форми замовлення, має наступний вигляд:

Дата	Номер замовлення	Номер клієнта	Прізвище	Ім'я	Вулиця	Будинок
22.06.2018	1	1	Іванов	Іван	Володимирська	77

Квартира	Місто, Індекс	Домашній телефон	Мобільний телефон	Інші телефони
3	Київ, 11028	+380445468298	+380667864521	+380636548500, +380446785982

Кількість	ID	Назва пончика	Опис	Ціна	Спеціальні побажання
1	1	Звичайний	Звичайний пончик	$1.50	Будь ласка, додайте тарілки та серветки.
5	2	Глазурований	Глазурований пончик	$1.75	
12	3	З корицею	Пончик з корицею	$1.75	
3	4	Шоколадний	Шоколадний пончик	$1.75	
4	5	З присипкою	Пончик з присипкою	$1.75	
5	6	Без глютену	Пончик без глютену	$2.00	

Слід звернути увагу на те, що збереження сум за кожним товаром та загальної суми замовлення призведе до додаткових потенційних неоднозначностей, а тому ці поля не слід включати до таблиці. З іншого боку, маючи кількість та ціну пончиків в замовленні ми можемо легко обчислити як суму для кожного з типів пончиків, так і загальну суму замовлення.

2. Для наведеної вище таблиці можна виділити такі приклади аномалій оновлення:

   - До таблиці додали інформацію про замовлення клієнта 2, який замовив 5 пончиків з корицею, але під час внесення даних допустили помилку в ціні пончика, увівши 1.55 замість 1.75. Виникає неоднозначність в ціні пончика з корицею.
   - З таблиці прибрали замовлення 1. Разом з ним було прибрано і дані про ціни всіх пончиків.
   - Нехай в таблиці також зазначено про замовлення клієнта 2, який замовив 5 пончиків з корицею. В таблиці оновили дані про ціну пончиків з корицею: їх ціна збільшилася на 10 центів. Проте при внесенні змін до таблиці випадково змінили ціну цих пончиків

лише в замовленні 1. Виникає неоднозначність в ціні пончика з корицею.

3. Таблиця «замовлення» подана в ненормалізованій формі, оскільки містить об'єднання комірок — на перетині рядка і стовпця тут подекуди розташовані два значення. Аби привести її до першої нормальної форми, слід розбити рядки з багатозначними полями та продублювати за потреби неповторювані дані (тут — дані про замовлення та клієнта). На виході цієї дії отримаємо таблицю в *1НФ* наступного вигляду:

Дата	Номер замовлення	Номер клієнта	Прізвище	Ім'я	Вулиця	Будинок
22.06.2018	1	1	Іванов	Іван	Володимирська	77
22.06.2018	1	1	Іванов	Іван	Володимирська	77
22.06.2018	1	1	Іванов	Іван	Володимирська	77
22.06.2018	1	1	Іванов	Іван	Володимирська	77
22.06.2018	1	1	Іванов	Іван	Володимирська	77
22.06.2018	1	1	Іванов	Іван	Володимирська	77

Квартира	Місто	Індекс	Мобільний телефон	Кількість	ID
3	Київ	11028	+380667864521	1	1
3	Київ	11028	+380667864521	5	2
3	Київ	11028	+380667864521	12	3
3	Київ	11028	+380667864521	3	4
3	Київ	11028	+380667864521	4	5
3	Київ	11028	+380667864521	5	6

Назва пончика	Опис	Ціна	Спеціальні побажання
Звичайний	Звичайний пончик	$1.50	Будь ласка, додайте тарілки та серветки.
Глазурований	Глазурований пончик	$1.75	Будь ласка, додайте тарілки та серветки.
З корицею	Пончик з корицею	$1.75	Будь ласка, додайте тарілки та серветки.
Шоколадний	Шоколадний пончик	$1.75	Будь ласка, додайте тарілки та серветки.
З присипкою	Пончик з присипкою	$1.75	Будь ласка, додайте тарілки та серветки.
Без глютену	Пончик без глютену	$2.00	Будь ласка, додайте тарілки та серветки.

Ключем цієї таблиці є комбінація полів «номер замовлення» та «ID».

4. З умови та даних таблиць випливає наявність таких повних функціональних залежностей:

   - «номер клієнта» → «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс», «мобільний телефон»;

- «номер замовлення» → «дата», «номер клієнта», «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс», «мобільний телефон», «спеціальні побажання»;
- «ID» → «назва пончика», «опис», «ціна»;
- «номер замовлення», «ID» → «кількість»;
- «мобільний телефон» → «номер клієнта», «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс».

5. Розглянемо почергово наступні кроки нормалізації для побудованої таблиці у першій нормальній формі.
Таблиця в другій нормальній формі не повинна містити неключових полів, що перебувають у повній функціональній залежності від частин принаймні одного з потенційних ключів, а не від всього потенційного ключа. В нашому випадку, дві повні функціональні залежності містять в своїй лівій частині підмножину полів деякого потенційного ключа:

   - «номер замовлення» → «дата», «номер клієнта», «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс», «спеціальні побажання»;
   - «ID» → «назва пончика», «опис», «ціна».

   Тому нам слід винести ці функціональні залежності у окремі таблиці разом з копією їх лівої частини. Отримаємо дві нові таблиці у *2НФ* наступного вигляду («замовлення» з ключовим полем «номер замовлення» та «пончики» з ключовим полем «номер пончика» відповідно):

Дата	Номер замовлення	Номер клієнта	Прізвище	Ім'я	Вулиця	Будинок
22.06.2018	1	1	Іванов	Іван	Володимирська	77

Квартира	Місто	Індекс	Мобільний телефон	Спеціальні побажання
3	Київ	11028	+380667864521	Будь ласка, додайте тарілки та серветки.

ID	Назва пончика	Опис	Ціна
1	Звичайний	Звичайний пончик	$1.50
2	Глазурований	Глазурований пончик	$1.75
3	З корицею	Пончик з корицею	$1.75
4	Шоколадний	Шоколадний пончик	$1.75
5	З присипкою	Пончик з присипкою	$1.75
6	Без глютену	Пончик без глютену	$2.00

Початкова таблиця у *2НФ* набуде вигляду:

Номер замовлення	Кількість	ID
1	1	1
1	5	2
1	12	3
1	3	4
1	4	5
1	5	6

Відповідно, реляційна модель на цьому етапі матиме наступний вигляд:

Таблиця у третій нормальній формі не повинна мати транзитивних залежностей неключових полів від ключа. Неважко переконатися, що таблиці «пончики» та «вміст замовлень» таких залежностей не містять, а тому вже перебувають в третій нормальній формі.
Таблиця «замовлення» містить дві транзитивні залежності від ключа:

- «номер замовлення» → «номер клієнта» → «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс», «мобільний телефон»;
- «номер замовлення» → «мобільний телефон» → «номер клієнта», «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс».

Обидві залежності включають один і той же набір полів, тому для приведення цієї таблиці в третю нормальну форму слід винести поля «прізвище», «ім'я», «вулиця», «будинок», «квартира», «місто», «індекс» разом з полями, через які вони транзитивно залежать від ключа («мобільний телефон» та «номер клієнта») в нову таблицю з двома потенційними ключами «мобільний телефон» та «номер клієнта» у *ЗНФ* («клієнти»):

Номер клієнта	Прізвище	Ім'я	Вулиця	Будинок	Квартира	Місто	Індекс	Мобільний телефон
1	Іванов	Іван	Володимирська	77	3	Київ	11028	+380667864521

Таблиця «замовлення» у *3НФ* набуде вигляду:

Дата	Номер замовлення	Номер клієнта	Спеціальні побажання
22.06.2018	1	1	Будь ласка, додайте тарілки та серветки.

Відповідно, реляційна модель на цьому етапі матиме наступний вигляд:

Розділ 3

Мова структурованих запитів SQL

В попередньому розділі ми ґрунтовно розглянули процес проектування баз даних, продуктом якого є фізично реалізована база даних. Структурування даних за таблицями дозволяє легко та ефективно отримувати дані з цього сховища, а також здійснювати інші маніпуляції над ними. Саме для цього і призначені запити.

Означення 45. *Запит – це команда СУБД, результатом виконання якої є вибірка даних, маніпуляція ними або структурою бази даних чи її складових.*

За призначенням виділяють наступні запити:

1. **Запити на вибірку.** Запити цього класу передбачають відбір даних за певними критеріями та їх повернення у табличному вигляді.
2. **Запити на маніпуляцію з даними.** За допомогою таких запитів можна здійснювати додавання, оновлення та вилучення даних з бази за певними критеріями.
3. **Запити на маніпуляцію структурою бази даних.** Результатом виконання запитів цього класу є зміна структури бази даних: додавання або вилучення таблиць, внесення змін до її полів, ключів, індексів, додавання або вилучення зв'язків між таблицями тощо.

Для складання запиту використовується мова пошукових запитів. Найбільш популярною серед мов запитів для реляційних баз даних є SQL — structured query language (мова структурованих запитів). Саме її вивченню

і присвячений цей розділ книги.

Перші два типи запитів відносяться до так званої мови маніпуляції даними (Data Manipulation Language, DML), третій — до мови опису даних (Data Definition Language, DDL). Ці дві підмови, разом з мовою контролю даних (Data Control Language, DCL) та мовою контролю транзакцій (Transaction Control Language, TCL), які не розглядаються в цій книзі, в сукупності складають мову SQL.

Слід зауважити, що деякі системи управління базами даних надають спеціалізовані засоби, які дещо полегшують створення запитів різного характеру (наприклад, майстер чи конструктор запитів в Access). Проте, мова SQL є базовою для більшості реляційних СУБД і надає можливість будувати більш широкий спектр запитів і деколи значно простіше, ніж подібні спеціалізовані засоби.

3.1 Виконання запитів в середовищі СУБД

Різні системи управління базами даних надають різний набір інструментів взаємодії з цими сховищами даних, проте для більшості реляційних СУБД спільним елементом є підтримка в них мови SQL. Розглянемо можливості щодо написання та виконання запитів у двох з них: Microsoft Access та MySQL.

3.1.1 Запити в СУБД Microsoft Access

Запит в Microsoft Access є одним з п'яти видів об'єктів бази даних, які зберігаються в ній на постійній основі.

Створити новий запит можна в будь-який момент за допомогою інструментів групи *Queries (Запити)* вкладки *Create (Створити)* головного меню програми (Рисунок 3.1).

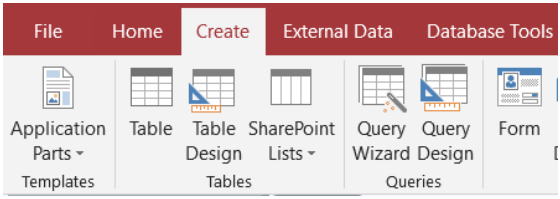


Рис. 3.1: Інструменти для створення запитів

Запит, як і таблиця, має режими подання конструктора та таблиці, проте має ще один додатковий режим — режим введення SQL (Рисунок 3.2), який нас цікавитиме наразі найбільше. Перейти в нього з будь-якого режиму можна за допомогою інструментів в правому нижньому куті екрану.

Рис. 3.2: Подання SQL

В даному вікні ви матимете змогу виконувати всі запити, що пропонуються як приклади або вправи далі в цій книзі. Аби виконати запит достатньо натиснути на інструмент *Run (Виконати)* панелі інструментів, або ж просто перейти в режим таблиці.

3.1.2 Запити в СУБД MySQL

На відміну від Microsoft Access, запити в MySQL безпосередньо в базі даних не зберігаються, а тільки виконуються.

Запустити на виконання новий запит можна в будь-який момент. Для цього слід перейти в необхідну базу даних за допомогою навігації в лівій частині екрану, а потім перейти на вкладку *SQL* головного меню вікна (Рисунок 3.3).

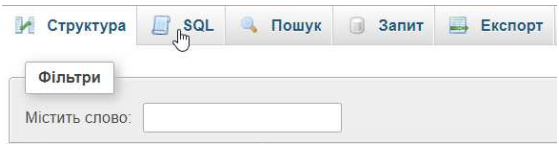


Рис. 3.3: Інструменти для створення запитів

У режимі введення SQL (Рисунок 3.4) ви матимете змогу виконувати всі запити, що пропонуються як приклади або вправи далі в цій книзі. Аби виконати запит достатньо натиснути на кнопку *Виконати* в нижній частині даного вікна.

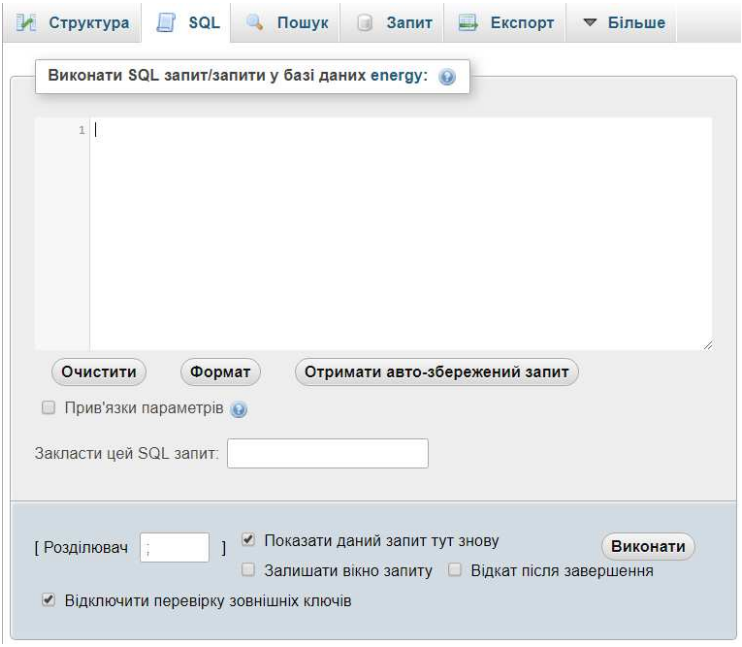


Рис. 3.4: Вкладка SQL

3.2 Вибірка даних

Запити на вибірку є основою більшості ресурсів, які ми бачимо в мережі Інтернет: від невеликих Інтернет-магазинів до гігантів на кшталт Facebook. Інформація, збережена в табличній базі даних, не завжди потрібна саме в тому вигляді, в якому зберігається. Часто для відповіді на запит необхідно виконувати складні дії над записами багатьох таблиць.

Цей підрозділ присвячений розгляду засобів побудови запитів на вибірку мовою SQL. В його рамках в якості прикладів, а також для закріплення вивченого матеріалу будуть представлені вправи різної складності, які читач може виконувати в середовищі системи *sql:itolymp*, доступної за адресою

http://sql.itolymp.com/. Для входу в кімнату даного курсу скористайтесь кодом *isdbbook2018*.

В цій системі ви матимете змогу виконувати написані власноруч запити, а також надсилати їх для автоматичної перевірки, що значно полегшить процес опанування цієї теми. Для кращої навігації, заняття кімнати відповідають темам цього підрозділу книги. В межах занять пропонується опрацювати як приклади, розв'язані в цій книзі, так і вправи, що пропонуються до самостійного виконання.

В межах цього та кількох наступних підрозділів для пояснення синтаксису команд мови SQL ми використовуватимемо наступні умовні позначення:

- літери верхнього регістру (наприклад, SELECT) означатимуть зарезервовані слова мови;
- літери нижнього регістру (наприклад, warehouses) означатимуть користувацькі слова;
- кутові дужки позначають елемент або сукупність елементів, зміст якої подана в цих дужках (наприклад, <назва таблиці>);
- квадратні дужки позначають необов'язковий елемент команди (наприклад, $[a]$).

3.2.1 Структура запиту на вибірку

В своїй найпростішій формі запит на вибірку має наступний вигляд:

SELECT <перелік полів або виразів>
FROM <табличний вираз>;

Запит на вибірку завжди починається з інструкції *SELECT* (*з англійської — вибрати*). Після цієї інструкції слідує перелік полів через кому, які потрібно повернути у відповідь на запит. Далі, після інструкції *FROM*, слідує табличний вираз — по суті, таблиця, одержана за допомогою операцій над наявними в базі даних таблицями.

Кожне з полів у переліку зазвичай подається у вигляді:

<назва таблиці>.<назва поля в таблиці>

При цьому в блоці *SELECT* можуть бути присутні тільки поля з тих таблиць, які в тому чи іншому вигляді фігурують в блоці *FROM*.

Наприклад, поле *last_name* таблиці *workers* побудованої нами бази даних мережі супермаркетів можна відобразити, вказавши його в SQL-коді як *workers.last_name*.

Якщо в запиті вибираються дані тільки з однієї таблиці або ім'я поля є унікальним серед всіх використаних в запиті таблиць, його можна включати

в запит і без назви таблиці.

Рекомендуємо на етапі вивчення мови SQL завжди вказувати назву таблиці для будь-якого поля в запиті.

Код з Лістингу 3.1 відобразить прізвища та імена всіх працівників мережі супермаркетів так, як це показано на Рисунку 3.5.

```
SELECT workers.last_name, workers.first_name
FROM workers;
```

Лістинг 3.1: Прізвища та імена працівників

Звертаємо вашу увагу на те, що SQL-запити зазвичай досить легко читаються. Наприклад, вище поданий запит можна прочитати так: «вибрати прізвище працівника та ім'я працівника з таблиці працівників».

last_name	first_name
Іванова	Наталія
Андрієнко	Роман
Боголюбов	Борис
Бобров	Семен
Петренко	Віктор
Титаренко	Віктор

Рис. 3.5: Прізвища та імена працівників

Вправа 77

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть інформацію про назви та місця розташування всіх електростанцій.
2. Відобразіть всю доступну інформацію про річки.
3. Відобразіть всю доступну інформацію з таблиці *power_units*.

У випадках, коли необхідно відобразити дані усіх полів таблиці, можна скористатися синтаксичним скороченням: замість назв полів після назви таблиці можна вказати символ *, що позначає довільне поле. Наприклад, щоб відобразити всю доступну інформацію про працівників мережі супермаркетів, достатньо виконати запит, код якого подано в Лістингу 3.2.

```
SELECT workers.*
FROM workers;
```

Лістинг 3.2: Всі дані про працівників

Результат виконання цього запиту подано на Рисунку 3.6.

passport_no	last_name	first_name	middle_name	qualification	chief
AA123456	Іванова	Наталія	Сергіївна	вища	*NULL*
AH564646	Андрієнко	Роман	Григорович	вища	AA123456
BP556465	Боголюбов	Борис	Арсенович	вища	AH564646
HH564127	Бобров	Семен	Іванович	1 кат.	AH564646
OH654643	Петренко	Віктор	Борисович	вища	CP486823
CP486823	Титаренко	Віктор	Леонідович	3 кат.	AA123456

Рис. 3.6: Всі дані про працівників

3.2.2 Використання обчислюваних полів

Не завжди для забезпечення потреб інформаційної системи достатньо відобразити наявні в базі дані. Досить часто виникає потреба здійснити попередні обчислення над даними таблиці та відобразити інформацію, яка є результатом такої обробки.

Одним з найпростішних методів такої обробки даних, що надається мовою SQL, є можливість відображення обчислюваних полів.

Розглянемо приклад. В таблиці *goods* в полі *price* зберігається ціна товару. Припустимо, що на кожну десяту одиницю товару мережа супермаркетів надає знижку 50%. Тоді вартість десяти одиниць кожного з товарів можна визначити за допомогою запиту, поданого в Лістингу 3.3.

```
SELECT goods.name, goods.price*9.5 AS total_price
FROM goods;
```

Лістинг 3.3: Вартість 10 одиниць товару

Результат виконання відповідного запиту подано на Рисунку 3.7.

Оскільки дев'ять одиниць з десяти матимуть звичайну ціну, а лише один — половину від звичайної, то загальна сума може бути легко отримана за допомогою множення ціни на число 9,5.

name	total_price
Батон білий	135.85
Молоко українське	268.85
Сир голандський	2195.45
Хліб білий	121.60
Сирок плавлений	125.40
Хліб житній	97.85

Рис. 3.7: Вартість 10 одиниць товару

Обчислюване поле, наведене в Лістингу 3.3, використовує операцію множення. В SQL аналогічним чином можна скористатися й іншими арифметичними операціями, а також великою кількістю вбудованих функцій. Оскільки набір функцій залежить від системи управління базами даних, в якій здійснюється запит, ми розглядатимемо їх пізніше окремо для кожної з двох СУБД, яким присвячена ця книга.

Також, для забезпечення коректної обробки результату запиту або його коректного відображення, обчислюване поле рекомендовано називати мнемонічною (змістовною) назвою. Для цього в мові SQL передбачено інструкцію для встановлення псевдоніму *AS*. Її синтаксис є наступним:

<поле або вираз> **AS** <псевдонім>

Слід зазначити, що за допомогою цієї інструкції можна також надавати інший псевдонім полям таблиці в межах запиту. Цим власне і зумовлена назва.

Вправа 78

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. В таблиці *rivers* в полі *length* подано інформацію про довжини річок в кілометрах. Відобразіть інформацію про назви та довжини річок в мегаметрах (1 мегаметр = 1000 кілометрів). Назвіть обчислюване поле *length*.
2. В таблиці *power_plants* в полі *power* подано інформацію про потужність електростанцій в мегаватах. Відобразіть інформацію про назви та потужності електростанцій в мегакалоріях на годину (1 мегават = 859,85 мегакалорій на годину). Назвіть обчислюване поле *megacalories*.

3.2.3 Впорядкування записів

За промовчанням записи результату запиту впорядковуються за первинним ключем таблиць, що для нього використовуються. Проте досить часто такий порядок не є достатньо інформативним. Особливо, коли сам первинний ключ не відображається в кінцевій таблиці-результаті. Саме тому в SQL існує спеціальна інструкція *ORDER BY*, яка дозволяє вирішити цю проблему.

Синтаксис запиту зі впорядкуванням записів матиме наступний вигляд:

```
SELECT <перелік полів або виразів>
FROM <табличний вираз>
[ORDER BY <перелік полів або виразів>];
```

Інструкція *ORDER BY* розташовується наприкінці коду і після неї вказується перелік полів, за якими здійснюватиметься впорядкування.

Найпростішим є випадок, коли необхідно впорядкувати записи за лише одним полем. Наприклад, якщо потрібно відобразити перелік товарів мережі супермаркетів у алфавітному порядку їх назв, достатньо виконати запит, код якого подано у Лістингу 3.4.

```
SELECT goods.*
FROM goods
ORDER BY goods.name;
```

Лістинг 3.4: Перелік товарів за алфавітним порядком їх назв

Після виконання такого запиту система управління базами даних поверне таблицю, зображену на Рисунку 3.8, в якій записи розташовані за назвою товару.

code	name ▲ 1	price	manufacturer_code
245558543	Батон білий	14.30	3
654646878	Молоко українське	28.30	2
654687269	Сир голандський	231.10	2
863543436	Сирок плавлений	13.20	2
821556286	Хліб білий	12.80	1
989651445	Хліб житній	10.30	1

Рис. 3.8: Перелік товарів за алфавітним порядком їх назв

Читається цей запит так: «вибрати всю інформацію про товари з таблиці товарів і впорядкувати за назвою товару».

Якщо ж потрібно впорядкувати товари у порядку, зворотному до алфавітного, достатньо після відповідного поля в блоці ORDER BY додати ключове слово DESC (від англійського словосполучення descending order — порядок зменшення). Відповідний код подано у Лістингу 3.5.

```
SELECT goods.*
FROM goods
ORDER BY goods.name DESC;
```

Лістинг 3.5: Перелік товарів у порядку, зворотному до алфавітного порядку їх назв

Вправа 79

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть всі дані про електростанції у порядку спадання їх потужності.
2. Відобразіть назви річок у порядку зростання їх довжини.

Варто зауважити, що доволі часто впорядкування записів за одним полем недостатньо. Наприклад, коли потрібно впорядкувати працівників за комбінацією прізвища, імені та по батькові.

Розглянемо запит, поданий в Лістингу 3.6.

```
SELECT workers.*
FROM workers
ORDER BY workers.qualification,
    workers.passport_no DESC;
```

Лістинг 3.6: Перелік працівників у алфавітному порядку категорій та зворотному порядку номерів паспортів

Інструкція *ORDER BY* дозволяє вказувати кілька полів для здійснення сортування. Спочатку записи впорядковуються за першим полем, потім записи з однаковим значенням першого поля — за другим, і т. д.

Вищенаведений запит призведе до отримання результату, зображеного на Рисунку 3.9.

В прикладі, що розглядається, спочатку було здійснено сортування за першим з вказаних полів — *workers.qualification*. Оскільки є чотири записи

passport_no	last_name	first_name	middle_name	qualification ▲ 1	chief
НН564127	Бобров	Семен	Іванович	1 кат.	АН564646
СР486823	Титаренко	Віктор	Леонідович	3 кат.	АА123456
ОН654643	Петренко	Віктор	Борисович	вища	СР486823
ВР556465	Боголюбов	Борис	Арсенович	вища	АН564646
АН564646	Андрієнко	Роман	Григорович	вища	АА123456
АА123456	Іванова	Наталія	Сергіївна	вища	*NULL*

Рис. 3.9: Перелік працівників

з однаковим значенням «вища» для цього поля, поміж них було здійснено сортування за другим з вказаних полів — *workers.passport_no* у зворотному напрямку (оскільки було вказано ключове слово *DESC*).

Зверніть увагу! Ключове слово *DESC* необхідно вказувати після кожного з полів, за яким порядок сортування має бути зворотнім до звичайного.

Вправа 80

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть код електростанції та номер реактора для кожного реактора. Впорядкуйте записи у зворотному до алфавітного порядку типів реакторів, а у разі однакового типу реактора — за зростанням номеру реактора. Якщо ж номер реактора також однаковий, впорядкуйте такі записи за спаданням коду електростанції.
2. Відобразіть коди атомних електростанцій. Розмістіть спочатку недіючі електростанції (значення поля *operationality* рівне 0), а потім діючі. У разі однакового статусу, розмістіть коди в порядку спадання.

3.2.4 Вибірка записів

Вище ми опанували засоби, за допомогою яких ми можемо повертати дані з однієї таблиці, обробляти та впорядковувати їх. Проте мова SQL надає

значно ширші можливості. До них відноситься зокрема й вибірка (фільтрація) записів, що проводиться за умовою, вміщеною в логічному виразі після інструкції *WHERE*.

Синтаксис запиту з вибіркою записів матиме наступний вигляд:

```
SELECT <перелік полів або виразів>
FROM <табличний вираз>
WHERE <вираз логічного типу>
ORDER BY <перелік полів або виразів>;
```

Для побудови умови зокрема можна використовувати оператори звичайних арифметичних порівнянь (=, >=, <=, <>). До результату запиту будуть включені лише ті рядки, для яких умова є істинною. Наприклад, відібрати товари, що коштують до 20 гривень включно можна за допомогою запиту, поданого в Лістингу 3.7.

```
SELECT goods.*
FROM goods
WHERE goods.price <= 20;
```

Лістинг 3.7: Товари ціною до 20 гривень включно

Результат виконання цього запиту зображено на Рисунку 3.10. Як можна переконатися, в результаті відображено лише товари, ціна яких не перевищує 20 гривень.

code	name	price	manufacturer_code
245558543	Батон білий	14.30	3
821556286	Хліб білий	12.80	1
863543436	Сирок плавлений	13.20	2
989651445	Хліб житній	10.30	1

Рис. 3.10: Товари ціною до 20 гривень включно

Цей запит можна прочитати так: «вибрати всю інформацію про товари з таблиці товарів, для яких ціна товару менша або рівна 20 гривень».

Також в якості умови може бути використане й порівняння з рядком (звісно, якщо тип відповідного поля — текстовий). Тоді рядок-порівняння вказується в лапках (наприклад, "вища") так, як це звично в популярних мовах програмування.

Відібрати працівників з вищою категорією можна за допомогою запиту, код якого вміщено в Лістингу 3.8.

```
SELECT workers.*
FROM workers
WHERE workers.qualification = "вища";
```

Лістинг 3.8: Працівники з вищою категорією

Результат виконання цього запиту зображено на Рисунку 3.11.

passport_no	last_name	first_name	middle_name	qualification	chief
AA123456	Іванова	Наталія	Сергіївна	вища	*NULL*
AH564646	Андрієнко	Роман	Григорович	вища	AA123456
BP556465	Боголюбов	Борис	Арсенович	вища	AH564646
OH654643	Петренко	Віктор	Борисович	вища	CP486823

Рис. 3.11: Працівники з вищою категорією

Але в житті часто таких простих умов недостатньо.

Вправа 81

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть назви та розташування електростанцій, потужність яких перевищує 1000 МВт.
2. Відобразіть коди гідроелектростанцій, що розташовані на річці Дніпро.
3. Відобразіть коди діючих атомних електростанцій.

3.2.5 Складені умовні вирази

Логічні оператори

Мова SQL надає можливість працювати з логічними операторами, звичними для багатьох мов програмування: *AND* (логічне «і»), *OR* (логічне «або») та *NOT* (логічне «ні»). Ці логічні зв'язки дозволяють будувати складені умовні вирази.

Синтаксис цих операторів є наступним:

<вираз логічного типу> **AND** <вираз логічного типу>
<вираз логічного типу> **OR** <вираз логічного типу>
NOT <вираз логічного типу>

Аби більш детально познайомитися з властивостями цих операторів та принципами побудови складених умовних виразів, розглянемо наступний приклад.

Приклад 4

Визначте істиннісне значення наступних логічних виразів:

1. (1 + 2 = 3) AND (6 + 3 = 9);
2. ("квітка"= "Квітка") AND (6 + 3 = 9);
3. ("квітка"= "Квітка") OR (6 + 3 = 9);
4. (1 + 2 = 1) OR (6 + 3 > 9);
5. NOT (5 = 1 + 4);
6. NOT ("квітка"= "Квітка").

Відповіді:

1. ІСТИНА; обидва логічних вирази зліва та справа від оператора AND є істинними, а тому і складений вираз є істинним;
2. ХИБНІСТЬ; вираз по правий бік оператора AND є істинним, проте рядки "квітка" та "Квітка" не є однаковими, тому правий вираз хибний і складений вираз також є хибним;
3. ІСТИНА; аргументи оператора OR ідентичні попередньому пункті, а тому складений вираз істинний;
4. ХИБНІСТЬ; обидва логічних вирази є хибними, а тому складений вираз теж є хибним;
5. ХИБНІСТЬ; логічний вираз, що є аргументом оператора NOT, є істинним, а тому складений вираз є хибним;
6. ІСТИНА; логічний вираз, що є аргументом оператора NOT, є хибним, а тому складений вираз є істинним.

Отже, щоб відобразити товари ціною від 10 до 20 гривень, достатньо виконати запит, код якого подано в Лістингу 3.9.

```
SELECT goods.*
FROM goods
```

```
WHERE goods.price >= 10 AND goods.price <= 20;
```

Лістинг 3.9: Товари ціною від 10 до 20 гривень

Результат виконання цього запиту зображено на Рисунку 3.12.

code	name	price	manufacturer_code
245558543	Батон білий	14.30	3
821556286	Хліб білий	12.80	1
863543436	Сирок плавлений	13.20	2
989651445	Хліб житній	10.30	1

Рис. 3.12: Товари ціною від 10 до 20 гривень

Оператор BETWEEN ... AND

Умову

```
goods.price >= 10 AND goods.price <= 20,
```

використану вище у запиті, можна записати коротше за допомогою оператора *BETWEEN ... AND*:

```
goods.price BETWEEN 10 AND 20,
```

Синтаксис цього оператора в загальному є наступним:

```
<вираз1> BETWEEN <вираз2> AND <вираз3>
```

За змістом така конструкція ідентична наступній:

```
<вираз1> >= <вираз2> AND <вираз1> <= <вираз3>
```

Зрозуміло, що тип виразів по правий бік від ключового слова *BETWEEN* повинні відповідати типу виразу по лівий бік.

Комбінування операторів у складених умовах

Якщо умова є більш складною та потребує кількаразового використання логічних операторів, рекомендуємо використовувати дужки, аби бути впевненим у тому, що написана умова відповідає змісту, який ви хотіли в неї закласти. Наприклад, умову «товари ціною більше 20 гривень виробника з кодом 2 або ціною менше 20 виробника з кодом 3» можна записати мовою SQL так:

```
(goods.price > 20 AND goods.manufacturer_code = 2)
    OR (goods.price < 20 AND
        goods.manufacturer_code = 3)
```

Вправа 82

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть назви та розташування електростанцій, потужність яких міститься в межах від 2000 МВт до 3000 МВт включно.
2. Відобразіть коди гідроелектростанцій, що розташовані на річках Дніпро або Дністер.
3. Відобразіть назви та розташування електростанцій, у яких потужність міститься в межах від 2000 МВт до 3000 МВт включно, або розташованих в Розкопинцях.

3.2.6 Особливості логіки в мові SQL

Розглянемо наступний приклад. Нехай необхідно визначити керівника мережі супермаркетів серед усіх працівників. Оскільки працівник є керівником мережі, в нього немає начальника і значення в полі *chief* таблиці *workers* відсутнє. Ми вже зазначали раніше, що за відсутнє або невизначене значення відповідає значення *NULL*.

Розглянемо код в лістингу 3.10.

```
SELECT workers.*
FROM workers
WHERE workers.chief = NULL;
```

Лістинг 3.10: Керівник мережі супермаркетів

Як не дивно, виконання такого коду не призведе до відображення жодного рядка результату. Насправді, в реляційній теорії і в SQL зокрема використовується не звична нам двійкова, а трійкова логіка, в якій окрім значень ІСТИНА (*TRUE*) та ХИБНІСТЬ (*FALSE*) є ще третє значення — НЕВИЗНАЧЕНІСТЬ (позначатимемо його *NULL*, оскільки значення *NULL* в тому числі відповідає й невизначності).

В розглянутому вище прикладі в умові

```
workers.chief = NULL
```

по праву сторону порівняння стоїть значення *NULL*, що відповідає невизначеному (невідомому) значенню. Чи може ця невизначеність бути рівна, наприклад, рядку "НН564127"? Так, але може й не бути рівною. Оскільки конкретне значення, що стоїть за цією невизначеністю, невідоме, результат порівняння невизначеності з будь-чим буде невизначеним. Отже, вищевказана умова завжди прийматиме значення *NULL* і жоден рядок відображений не буде.

Слід зауважити, що за двома невизначеностями також не обов'язково стоїть одне й теж значення, а тому умова

NULL = **NULL**

теж приймає невизначене значення *NULL*.

Більшість операторів, зокрема арифметичні та логічні, у разі невизначності будь-якого з аргументів повертають *NULL*. Проте існує оператор, який дозволяє визначити значення *NULL*. Його синтаксис є наступним:

<поле або вираз> IS **NULL**

Цей оператор повертає значення ІСТИНА (*TRUE*), якщо відповідний вираз є *NULL*, і ХИБНІСТЬ (*FALSE*) — в протилежному випадку.

Двоїстий до нього оператор *IS NOT NULL* повертає значення ХИБНІСТЬ (*FALSE*), якщо відповідний вираз є *NULL*, і ІСТИНА (*TRUE*) — в протилежному випадку.

Отже, запит для визначення керівника мережі супермаркетів слід привести до вигляду, поданого в Лістингу 3.11.

```
SELECT workers.*
FROM workers
WHERE workers.chief IS NULL;
```

Лістинг 3.11: Керівник мережі супермаркетів

Результат виконання цього запиту зображено на Рисунку 3.13.

passport_no	last_name	first_name	middle_name	qualification	chief
AA123456	Іванова	Наталія	Сергіївна	вища	*NULL*

Рис. 3.13: Керівник мережі супермаркетів

Вправа 83

Відомо, що для двійкової логіки вираз

X OR NOT X,

де X — довільний логічний вираз, завжди є істинним. Чи справедлива така властивість для трійкової логіки? Якщо так, доведіть це твердження. Якщо ні — наведіть приклад коли ця умова не є істиною та подібну властивість для цієї логіки і доведіть її.

Вправа 84

За допомогою запиту до бази даних «Енергетика», написаного мовою SQL, відобразіть коди гідроелектростанцій, тип яких є невідомим.

3.2.7 Вибірка записів за допомогою шаблонів

Вище ми розглянули деякі прості засоби, які надає мова SQL для вибірки певних записів таблиці. Проте досить часто умова, якій повинні відповідати записи результату, є досить складною і обійтися під час її втілення тільки операціями порівняння та логічними операторами не вдається. Саме тому в SQL зокрема підтримується використання в умовах шаблонів.

Означення 46. *Шаблон — це рядок (можливо, зі спеціальними символами), що описує певну множину рядків відповідно до набору спеціальних синтаксичних правил.*

Шаблони є більш простою версією регулярних виразів — потужного апарату, що широко використовується для опису цілого класу мов. Порівняння рядка з шаблоном значно розширює можливості звичайного посимвольного порівняння рядків, дозволяючи опускати під час порівняння несуттєві для умови частини рядка.

В цьому розділі ми детально зупинимося на шаблонах, притаманних системі управління базами даних MySQL. Відмінності у роботі з шаблонами в СУБД Access будуть розглянуті далі в цій книзі.

Найбільшу цінність в SQL-подібних шаблонах, звісна річ, складають їх спеціальні символи:

- символ % замінює будь-яку кількість довільних символів (наприклад, «а», «Р», «454», «лікар», «Школа», «»);
- символ _ замінює рівно один довільний символ («А», «р», «_», «5»).

Аби більш ґрунтовно познайомитися з принципами побудови шаблонів, розглянемо приклад.

Приклад 5

Побудуйте шаблони для множин рядків, описаних нижче:

1. *будь-який рядок, що починається з літери а;*
2. *рядок, що містить слово «літак»;*
3. *рядок з п'яти довільних символів;*
4. *рядок з п'яти або більше довільних символів.*

Відповіді:

1. *а%*;
2. *%літак%*;
3. _____;
4. _____% або %_____.

Вправа 85

Побудуйте регулярні вирази для множин рядків, описаних нижче:

1. рядок, що містить фразу «у воді»;
2. рядок, що містить принаймні одну літеру «а»;
3. рядок, що містить принаймні три символи;
4. будь-який рядок.

Для порівняння рядка з шаблоном в SQL передбачено спеціальний оператор *LIKE* з наступним синтаксисом:

<поле або вираз> **LIKE** <рядок-шаблон>

Цей регістронезалежний оператор повертає значення ІСТИНА (*TRUE*), якщо поле або вираз по лівий бік від оператора відповідає шаблону по правий бік від нього, і повертає ХИБНІСТЬ (*FALSE*) в протилежному випадку.

Приклад 6

Яке значення приймають наступні логічні вирази, записані мовою SQL:

1. *"Данило"LIKE "дАнило";*
2. *"яблуко"LIKE "%у%";*
3. *"%у%"LIKE "яблуко";*
4. *"ас"LIKE "___".*

Відповіді:

1. ІСТИНА, оскільки оператор є регістронезалежним;
2. ІСТИНА, оскільки рядок зліва дійсно містить літеру «у»;
3. ХИБНІСТЬ, оскільки рядок зліва не відповідає шаблону «яблуко»;
4. ХИБНІСТЬ, оскільки рядок зліва містить два символи, а не три, як того вимагає шаблон.

Оскільки оператор *LIKE* повертає логічне значення, вираз з ним можна використовувати в умові запиту безпосередньо, або ж комбінуючи його з іншими логічними виразами за допомогою логічних операторів.

Наприклад, аби знайти інформацію про всі типи хлібів, які продаються в нашій мережі супермаркетів, слід виконати запит, код якого поданого в Лістингу 3.12.

```
SELECT goods.*
FROM goods
WHERE goods.name LIKE "%хліб%";
```

Лістинг 3.12: Типи хлібів

Результат виконання цього запиту зображено на Рисунку 3.14.

code	name	price	manufacturer_code
821556286	Хліб білий	12.80	1
989651445	Хліб житній	10.30	1

Рис. 3.14: Типи хлібів

Двоїстим до оператора *LIKE* є оператор *NOT LIKE* з наступним синтаксисом:

<поле або вираз> **NOT LIKE** <рядок-шаблон>

За своїм змістом вище наведений вираз ідентичний наступному:

NOT (<поле або вираз> **LIKE** <рядок-шаблон>)

Вправа 86

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть назви та місця розташування гідроакумулятивних електростанцій (ГАЕС) в алфавітному порядку назв.
2. Відобразіть номер АЕС, номер реактора та потужність для кожного реактора класу «ВВЕР». Впорядкуйте записи за зростанням номерів АЕС та реактора — в межах однієї АЕС.
3. Відобразіть назви електростанцій, потужність яких є непарним числом.
4. Відобразіть назви електростанцій, назва місця розташування яких містить принаймні одну літеру «а».
5. Відобразіть назви електростанцій, назва місця розташування яких не містить жодної літери «а».

3.2.8 Особливості порівняння з шаблоном в СУБД Access

Загальні принципи порівняння з шаблоном в СУБД Access та MySQL збігаються, проте набір спеціальних символів в Microsoft Access дещо інший:

- символ * замінює будь-яку кількість довільних символів (наприклад, «а», «Р», «454», «лікар», «Школа», «»);
- символ ? замінює рівно один довільний символ («А», «р», «_», «5»);
- символи *[...]* замінюють рівно один символ з переліку, поданого замість трикрапки:
  - *[а-я]* – довільна літера від «а» до «я»;
  - *[аеоуиі]* – одна з літер «а», «е», «о», «у», «и», «і»;
- символи *[!...]* замінюють рівно один будь-який символ, окрім символів з переліку, поданого замість трикрапки:
  - *[!а-я]* – довільний символ окрім літер від «а» до «я»;

– *[!аеоуиі]* – довільний символ окрім літер «а», «е», «о», «у», «и», «і».

Аби більш ґрунтовно познайомитися з принципами побудови шаблонів в Access, розглянемо приклад.

Приклад 7

Побудуйте шаблони з використанням спеціальних символів Access для множин рядків, описаних нижче:

1. *будь-який рядок, що починається з літери а;*
2. *рядок, що містить слово «літак»;*
3. *рядок з п'яти довільних символів;*
4. *слово з п'яти літер кирилиці.*

Відповіді:

1. *а**;
2. **літак**;
3. *?????*;
4. *[а-яА-Я][а-яА-Я][а-яА-Я][а-яА-Я][а-яА-Я].*

Вправа 87

Побудуйте регулярні вирази для множин рядків, описаних нижче:

1. рядок, що містить фразу «у воді»;
2. діапазон з двох двоцифрових чисел (наприклад, 12–19);
3. рядок, що містить принаймні одну голосну літеру кирилиці;
4. рядок, що містить принаймні три символи;
5. рядок, що починається з кириличної літери;
6. рядок, що містить принаймні один символ, відмінний від голосних літер.

Вправа 88

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання в середовищі Microsoft Access:

1. Відобразіть назви та місця розташування гідроакумулятивних елетростанцій (ГАЕС) в алфавітному порядку назв.
2. Відобразіть номер АЕС, номер реактора та потужність для кожного реактора класу «ВВЕР». Впорядкуйте записи за зростанням номерів АЕС та реактора — в межах однієї АЕС.
3. Відобразіть назви електростанцій, потужність яких є непарним числом.
4. Відобразіть назви електростанцій, назва місця розташування яких не містить жодної літери «а».

3.2.9 Оператори над таблицями. Багатотабличні запити

Всі запити, які ми будували досі, базувалися лише на одній певній таблиці. Проте, часто необхідно отримувати інформацію з кількох таблиць одночасно, послуговуючись зв'язами, які між цими таблицями побудовані.

Нагадаємо поточну структуру SQL-запиту на основі вивчених нами тем:

SELECT <перелік полів або виразів>
FROM <табличний вираз>
[**WHERE** <вираз логічного типу>]
[**ORDER BY** <перелік полів або виразів>];

Як і було зазначено на початку розгляду мови SQL, після інструкції FROM може слідувати не просто таблиця, а й табличний вираз. Нижче ми розглянемо як саме такі вирази формувати.

Перехресне об'єднання (CROSS JOIN)

Найпростішою операцією над двома таблицями є операція *декартового добутку, або перехресне об'єднання (CROSS JOIN)*. Синтаксис цієї операції доволі простий:

<табличний вираз>, <табличний вираз>

Тобто два табличні вирази записуються одне за одним через кому.

Результатом цієї операції, застосованої до двох таблиць, є таблиця, стовпці якої включають стовпці обох цих таблиць, а записи є всіма можливими комбінаціями записів цих таблиць.

Розглянемо дві таблиці: таблицю товарів *goods* (Рисунок 3.15) та таблицю виробників *manufacturers* (Рисунок 3.16).

code	name	price	manufacturer_code
245558543	Батон білий	14.30	3
654646878	Молоко українське	28.30	2
654687269	Сир голандський	231.10	2
821556286	Хліб білий	12.80	1
863543436	Сирок плавлений	13.20	2
989651445	Хліб житній	10.30	1

Рис. 3.15: Таблиця *goods*

code	name	address
1	Київхліб	м. Київ, вул. Межигірська, 83
2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.16: Таблиця *manufacturers*

Отже, декартів добуток цих двох таблиць матиме поля *goods.code*, *goods.name*, *goods.price*, *goods.manufacturer_code*, *manufacturers.code*, *manufacturers.name*, *manufacturers.address*. Утворену таким чином таблицю можна переглянути, виконавши запит, код якого подано в Лістингу 3.13.

```
SELECT *
FROM goods, manufacturers;
```

Лістинг 3.13: Декартів добуток таблиць *goods* та *manufacturers*

Результат виконання цього запиту, а отже і результат операції перехресного об'єднання, подано на Рисунку 3.17.

Вправа 89

Як за допомогою цієї таблиці отримати для кожного товару всі дані про його виробника? Запишіть відповідний SQL-запит.

Теорема 5. Якщо таблицях A та B m та n записів відповідно, то в таблиці A, B — $m * n$ записів.

Вправа 90

Доведіть вищенаведене твердження.

code	name	price	manufacturer_code	code	name	address
245558543	Батон білий	14.30	3	1	Київхліб	м. Київ, вул. Межигірська, 83
245558543	Батон білий	14.30	3	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
245558543	Батон білий	14.30	3	3	Кулиничі	м. Харків, вул. Грищенка, 17
654646878	Молоко українське	28.30	2	1	Київхліб	м. Київ, вул. Межигірська, 83
654646878	Молоко українське	28.30	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
654646878	Молоко українське	28.30	2	3	Кулиничі	м. Харків, вул. Грищенка, 17
654687269	Сир голандський	231.10	2	1	Київхліб	м. Київ, вул. Межигірська, 83
654687269	Сир голандський	231.10	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
654687269	Сир голандський	231.10	2	3	Кулиничі	м. Харків, вул. Грищенка, 17
821556286	Хліб білий	12.80	1	1	Київхліб	м. Київ, вул. Межигірська, 83
821556286	Хліб білий	12.80	1	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
821556286	Хліб білий	12.80	1	3	Кулиничі	м. Харків, вул. Грищенка, 17
863543436	Сирок плавлений	13.20	2	1	Київхліб	м. Київ, вул. Межигірська, 83
863543436	Сирок плавлений	13.20	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
863543436	Сирок плавлений	13.20	2	3	Кулиничі	м. Харків, вул. Грищенка, 17
989651445	Хліб житній	10.30	1	1	Київхліб	м. Київ, вул. Межигірська, 83
989651445	Хліб житній	10.30	1	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
989651445	Хліб житній	10.30	1	3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.17: Декартів добуток таблиць *goods* та *manufacturers*

Вправа 91

За допомогою запиту до бази даних «Енергетика», написаного мовою SQL, побудуйте перехресний добуток таблиць *nuclear_power_plants* та *power_plants*.

Приклад 8

Відобразіть пари колег — працівників, що мають одного й того ж начальника.

Аби отримати пари колег, необхідно спочатку отримати всі пари працівників мережі супермаркетів. Це можна зробити за допомогою операції декартового добутку таблиці працівників самої з собою. Але щойно ми виконаємо код, поданий в Лістингу 3.14, ми одержимо помилку *#1066 — Неунікальна таблиця/псевдонім: 'workers'*.

```
SELECT *
```

```
FROM workers, workers;
```

Лістинг 3.14: Колеги

Дійсно, в межах одного запиту не може бути двох таблиць з однією й тією ж назвою. Проте раніше ми вже розглядали оператор призначення синонімів *AS*, за допомогою перейменовували поля запиту. За допомогою цього ж оператора можна надавати синоніми й таблицям. Розглянемо код в Лістингу 3.15.

```
SELECT *
FROM workers, workers AS workers_1;
```

Лістинг 3.15: Колеги

На Рисунку 3.18 подано кілька перших рядків результату виконання цього коду. Тут відображено дані тільки деяких полів, оскільки таблиця має великі розміри. Аби переглянути повноформатну таблицю-результат, виконайте цей запит в середовищі бази даних.

passport_no	last_name	chief	passport_no	last_name	chief
AA123456	Іванова	*NULL*	AA123456	Іванова	*NULL*
AH564646	Андрієнко	AA123456	AA123456	Іванова	*NULL*
BP556465	Боголюбов	AH564646	AA123456	Іванова	*NULL*
HH564127	Бобров	AH564646	AA123456	Іванова	*NULL*
OH654643	Петренко	CP486823	AA123456	Іванова	*NULL*
CP486823	Титаренко	AA123456	AA123456	Іванова	*NULL*
AA123456	Іванова	*NULL*	AH564646	Андрієнко	AA123456
AH564646	Андрієнко	AA123456	AH564646	Андрієнко	AA123456
BP556465	Боголюбов	AH564646	AH564646	Андрієнко	AA123456
HH564127	Бобров	AH564646	AH564646	Андрієнко	AA123456
OH654643	Петренко	CP486823	AH564646	Андрієнко	AA123456
CP486823	Титаренко	AA123456	AH564646	Андрієнко	AA123456
AA123456	Іванова	*NULL*	BP556465	Боголюбов	AH564646
AH564646	Андрієнко	AA123456	BP556465	Боголюбов	AH564646
BP556465	Боголюбов	AH564646	BP556465	Боголюбов	AH564646
HH564127	Бобров	AH564646	BP556465	Боголюбов	AH564646
OH654643	Петренко	CP486823	BP556465	Боголюбов	AH564646

Рис. 3.18: Колеги

Маючи всі можливі пари працівників отримати пари колег, що мають спільного начальника, можна додавши просту умову (Лістинг 3.16).

```
SELECT *
```

```
FROM workers, workers AS workers_1
WHERE workers.chief = workers_1.chief;
```

Лістинг 3.16: Колеги

Зверніть увагу! Після надання таблиці псевдоніму до неї в межах запиту слід звертатися саме за цим псевдонімом.
На Рисунку 3.19 подано дані деяких полів результату виконання цього коду.

passport_no	last_name	chief	passport_no	last_name	chief
AH564646	Андрієнко	AA123456	AH564646	Андрієнко	AA123456
CP486823	Титаренко	AA123456	AH564646	Андрієнко	AA123456
BP556465	Боголюбов	AH564646	BP556465	Боголюбов	AH564646
HH564127	Бобров	AH564646	BP556465	Боголюбов	AH564646
BP556465	Боголюбов	AH564646	HH564127	Бобров	AH564646
HH564127	Бобров	AH564646	HH564127	Бобров	AH564646
OH654643	Петренко	CP486823	OH654643	Петренко	CP486823
AH564646	Андрієнко	AA123456	CP486823	Титаренко	AA123456
CP486823	Титаренко	AA123456	CP486823	Титаренко	AA123456

Рис. 3.19: Колеги

На цьому рисунку можна помітити, що в результаті даного запиту кожна пара відображається двічі (змінюються лише позиції працівників в парі). Також до відповіді включено пари, в яких працівник розташований на обох позиціях (адже він має спільного начальника сам з собою). Спробуємо позбутися цих недоліків.
Оскільки номер паспорта для працівника є унікальним, ми можемо взяти за аксіому той факт, що номер паспорта першого працівника має бути меншим за номер паспорта другого працівника (Лістинг 3.17).

```
SELECT *
FROM workers, workers AS workers_1
WHERE workers.chief = workers_1.chief AND
      workers.passport_no < workers_1.passport_no;
```

Лістинг 3.17: Колеги

Дійсно, якщо перший з працівників пари виду (A, B) має номер паспорта менший за номер паспорта другого з працівників, то пара виду (B, A) вже не може потрапити до результату. Аналогічно, оскільки $A = A$, то пара виду (A, A) теж не може потрапити до результату.
Результат виконання цього запиту подано на Рисунку 3.20.

passport_no	last_name	chief	passport_no	last_name	chief
АН564646	Андрієнко	АА123456	СР486823	Титаренко	АА123456
ВР556465	Боголюбов	АН564646	НН564127	Бобров	АН564646

Рис. 3.20: Колеги

Вправа 92

За допомогою запиту до бази даних «Енергетика», написаного мовою SQL, відобразіть пари кодів електростанцій, що розміщені на одній річці. Відобразіть спочатку менший з кодів з назвою поля *HPP_1*, а потім більший — з назвою *HPP_2*.

Повернімося до декартового добутку таблиць *goods* та *manufacturers*. В ході розгляду логічного проектування баз даних ми зазначали, що власне зв'язок між записами різних таблиць забезпечується за допомогою зовнішніх ключів. В нашому випадку, зовнішнім ключем є поле *manufacturer_code*, а відповідним йому ключем таблиці *manufacturers* — поле *code*.

Записи вважаються зв'язаними тоді, коли зовнішній ключ рівний відповідному основному ключу. Тобто отримати інформацію про виробника кожного з товарів можна, доповнивши декартів добуток відповідною умовою (Лістинг 3.18).

```
SELECT *
FROM goods, manufacturers
WHERE goods.manufacturer_code = manufacturers.code;
```

Лістинг 3.18: Дані про виробника для кожного з товарів

Результатом виконання цього запиту буде таблиця, подана на Рисунку 3.21.

code	name	price	manufacturer_code	code	name	address
245558543	Батон білий	14.30	3	3	Кулиничі	м. Харків, вул. Грищенка, 17
654646878	Молоко українське	28.30	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
654687269	Сир голандський	231.10	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
821556286	Хліб білий	12.80	1	1	Київхліб	м. Київ, вул. Межигірська, 83
863543436	Сирок плавлений	13.20	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
989651445	Хліб житній	10.30	1	1	Київхліб	м. Київ, вул. Межигірська, 83

Рис. 3.21: Дані про виробника для кожного з товарів

Вправа 93

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Для кожної АЕС відобразіть інформацію про її назву, розташування, потужність та діючість.
2. Відобразіть такі пари з назви гідроелектростанції (назвіть поле *HPP*) та назви річки (назвіть поле *river_name*), що ГЕС з кожної пари розташована на річці з цієї ж пари. Впорядкуйте пари за назвами ГЕС та річки.
3. Відобразіть назви гідроакумулюючих ГЕС в алфавітному порядку.

Внутрішнє об'єднання (INNER JOIN)

Наступною операцією над таблицями мови SQL є операція *внутрішнього об'єднання (INNER JOIN)*. Синтаксис цієї операції наступний:

<табличний вираз 1> **INNER JOIN** <табличний вираз 2>
ON <вираз логічного типу (умова)>

Внутрішнє об'єднання за двома таблицями та умовою утворює найбільшу підмножину декартового добутку цих таблиць, що задовільняє заданій умові. За своїм змістом ця операція є аналогічною до перехресного об'єднання з накладеною на нього умовою:

FROM <табличний вираз 1>, <табличний вираз 2>
WHERE <вираз логічного типу (умова)>

Отримати інформацію про виробника кожного з товарів за допомогою внутрішнього об'єднання можна за допомогою запиту, поданого в Лістингу 3.19.

```
SELECT *
FROM goods INNER JOIN manufacturers
    ON goods.manufacturer_code =
        manufacturers.code;
```

Лістинг 3.19: Дані про виробника для кожного з товарів

Результатом виконання цього запиту буде таблиця, подана на Рисунку 3.22.

code	name	price	manufacturer_code	code	name	address
245558543	Батон білий	14.30	3	3	Кулиничі	м. Харків, вул. Грищенка, 17
654646878	Молоко українське	28.30	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
654687269	Сир голандський	231.10	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
821556286	Хліб білий	12.80	1	1	Київхліб	м. Київ, вул. Межигірська, 83
863543436	Сирок плавлений	13.20	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
989651445	Хліб житній	10.30	1	1	Київхліб	м. Київ, вул. Межигірська, 83

Рис. 3.22: Дані про виробника для кожного з товарів

Зверніть увагу! Хоча внутрішнє об'єднання може бути повністю замінене перехресним з відповідною умовою, рекомендується для поєднання таблиць використовувати саме INNER JOIN, оскільки це покращує читабельність написаного коду та логіку запиту.

Вправа 94

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL з використанням оператора *INNER JOIN*, виконайте наступні завдання:

1. Для кожної АЕС відобразіть інформацію про її назву, розташування, потужність та діючість.
2. Відобразіть такі пари з назви гідроелектростанції (назвіть поле *HPP*) та назви річки (назвіть поле *river_name*), що ГЕС з кожної пари розташована на річці з цієї ж пари. Впорядкуйте пари за назвами ГЕС та річки.
3. Відобразіть назви гідроакумулюючих ГЕС в алфавітному порядку.

Вправа 95

За допомогою запиту до бази даних «Енергетика», написаного мовою SQL з використанням оператора *INNER JOIN*, відобразіть пари кодів електростанцій, що розміщені на одній річці. Відобразіть спочатку менший з кодів з назвою поля *HPP_1*, а потім більший — з назвою *HPP_2*.

Оскільки операції об'єднання повертають також таблицю, це дає змогу пов'язувати в запиті дані більш, ніж двох таблиць. Це може знадобитися, наприклад, якщо ми хочемо відобразити прізвища та імена подружніх пар

серед працівників мережі супермаркетів.

В таблиці marriage зберігається інформація про номери паспортів подружніх пар, а тому слід з'єднати її в запиті з двома екземплярами таблиці workers, аби за паспортом можна було дізнатися й іншу інформацію (Лістинг 3.20).

```
SELECT husband.last_name AS husband_LN,
       husband.first_name AS husband_FN,
       wife.last_name AS wife_LN,
       wife.first_name AS wife_FN
FROM (
          marriage INNER JOIN workers AS husband
          ON marriage.husband_passport_no =
               husband.passport_no
       ) INNER JOIN workers AS wife
       ON marriage.wife_passport_no =
            wife.passport_no;
```

Лістинг 3.20: Подружні пари

Вправа 96

За допомогою запиту до бази даних «Енергетика», написаного мовою SQL з використанням оператора *INNER JOIN*, відобразіть всю інформацію про енергоблоки діючих АЕС, а також назву, розташування та потужність відповідної АЕС, на якій кожен з енергоблоків розташований.

Вправа 97

За допомогою запиту до бази даних «Енергетика», написаного мовою SQL з використанням оператора *INNER JOIN*, відобразіть пари назв електростанцій, що розміщені на одній річці. Відобразіть спочатку електростанцію з меншим кодом з назвою поля *HPP_1*, а потім з більшим — з назвою *HPP_2*.

Зовнішні об'єднання (OUTER JOIN)

Логіка назви операції внутрішнього об'єднання, яку ми розглянули вище, підказує нам про існування *зовнішнього об'єднання*. Загалом існує три операції зовнішнього об'єднання: *ліве (LEFT JOIN), праве (RIGHT JOIN) та повне (FULL JOIN)*. Більшість систем управління базами даних реалізують лише дві перші операції, в тому числі MySQL та Access.

Розглянемо запит, що визначає які товари на яких складах зберігаються (Лістинг 3.21).

```
SELECT goods.*, holds.warehouse_code, holds.amount
FROM goods INNER JOIN holds
    ON goods.code = holds.goods_code;
```

Лістинг 3.21: Товари на складах

Результатом виконання цього запиту буде таблиця, подана на Рисунку 3.23.

code	name	price	manufacturer_code	warehouse_code	amount
245558543	Батон білий	14.30	3	1	11
654646878	Молоко українське	28.30	2	1	12
245558543	Батон білий	14.30	3	2	17
654687269	Сир голандський	231.10	2	2	22

Рис. 3.23: Дані про виробника для кожного з товарів

Як можна помітити на рисунку, далеко не всі товари наявні наразі на складах. Але в запиті жодної інформації про них немає, адже для жодного з таких товарів не буде виконана умова *goods.code = holds.goods_code*. Яким чином тоді додатково відобразити товари, яких на складах наразі немає? Для цього якраз і потрібні зовнішні об'єднання.

В чому ж різниця між трьома типами зовнішніх об'єднань?

Ліве зовнішнє об'єднання (*LEFT JOIN*) включає до свого результату всі записи результату операції *INNER JOIN* над тими ж таблицями, а також записи лівого аргумента, для яких жодного разу не була виконана умова об'єднання.

Нехай задані дві таблиці (Рисунок 3.24). Тоді результат виконання запиту з Лістингу 3.22 буде мати вигляд, поданий на Рисунку 3.25.

```
SELECT goods.*, holds.warehouse_code, holds.amount
FROM goods LEFT JOIN holds
    ON goods.code = holds.goods_code;
```

Лістинг 3.22: Товари на складах

code	name	price	manufacturer_code
245558543	Батон білий	14.30	3
654646878	Молоко українське	28.30	2
654687269	Сир голандський	231.10	2
821556286	Хліб білий	12.80	1
863543436	Сирок плавлений	13.20	2
989651445	Хліб житній	10.30	1

а)

amount	warehouse_code	goods_code
11	1	245558543
12	1	654646878
17	2	245558543
22	2	654687269

б)

Рис. 3.24: Таблиці (а) *goods* та (б) *holds*

code	name	price	manufacturer_code	warehouse_code	amount
245558543	Батон білий	14.30	3	1	11
654646878	Молоко українське	28.30	2	1	12
245558543	Батон білий	14.30	3	2	17
654687269	Сир голандський	231.10	2	2	22
821556286	Хліб білий	12.80	1	*NULL*	*NULL*
863543436	Сирок плавлений	13.20	2	*NULL*	*NULL*
989651445	Хліб житній	10.30	1	*NULL*	*NULL*

Рис. 3.25: Дані про виробника для кожного з товарів з лівим об'єднанням

Дійсно, зображена на рисунку таблиця містить всі рядки результату операції *INNER JOIN* (Рисунок 3.23), а також три товари: *Хліб білий*, *Сирок плавлений* та *Хліб житній*, — які не зберігаються на жодному зі складів, а отже для них не могла бути виконана умова $goods.code = holds.goods\_code$. Причому, оскільки для цих трьох товарів немає зв'язаного запису в таблиці *holds*, то значення полів *warehouse_ code* та *amount* з цієї таблиці в результаті запиту для цих рядків є невизначеним (*NULL*).

Вправа 98

Як за допомогою цієї таблиці отримати всі дані про товари, які не зберігаються на складах мережі супермаркетів? Запишіть відповідний SQL-запит.

Праве зовнішнє об'єднання (*RIGHT JOIN*) включає до свого результату всі записи результату операції *INNER JOIN* над тими ж таблицями, а також ті записи правого аргумента, для яких жодного разу не була виконана умова об'єднання. Власне зміст цього зовнішнього об'єднання аналогічний до *LEFT JOIN* з точністю до порядку аргументів.

Повне зовнішнє об'єднання (*FULL JOIN*) включає до свого результату всі записи результату операції *INNER JOIN* над тими ж таблицями, а також ті записи обох аргументів, для яких жодного разу не була виконана умова об'єднання. Цей тип за змістом є поєднанням *LEFT JOIN* та *RIGHT JOIN*.

Зверніть увагу! FULL JOIN відсутній в СУБД MySQL та Access!

Приклад 9

Відобразіть назви складів, що не зберігають жодного товару. Аби отримати склади, що не зберігають жодного товару, скористаємось спочатку лівим зовнішнім об'єднанням для отримання інформації про всі склади та пов'язані з ними товари (Лістинг 3.23).

```
SELECT warehouses.*, holds.goods_code, holds.amount
FROM warehouses LEFT JOIN holds
    ON warehouses.code = holds.warehouse_code;
```

Лістинг 3.23: Склади та збережувані в них товари

На Рисунку 3.26 подано результат виконання цього коду.

code	name	address	chief_passport_no	goods_code	amount
1	Васильківський	м. Київ, вул. Васильківська, 36	АН564646	245558543	11
1	Васильківський	м. Київ, вул. Васильківська, 36	АН564646	654646878	12
2	Голосіївський	м. Київ, вул. Ломоносова, 89	СР486823	245558543	17
2	Голосіївський	м. Київ, вул. Ломоносова, 89	СР486823	654687269	22
3	Ужгородський	м. Ужгород, вул. Володимирська, 3	АА123456	*NULL*	*NULL*

Рис. 3.26: Склади та збережувані в них товари

З рисунку помітно, що для всіх складах, які не містять жодного товару, значення полів *goods_code* та *amount* будуть невизначені (*NULL*). Пригадуючи, що перевірку поля на значення *NULL* можна здійснити за допомогою оператора *IS NULL*, модифікуємо код запиту так, як вказано в Лістингу 3.24.

```
SELECT warehouses.*
FROM warehouses LEFT JOIN holds
    ON warehouses.code = holds.warehouse_code
WHERE holds.amount IS NULL;
```

Лістинг 3.24: Порожні склади

Зверніть увагу, що невизначені поля було прибрано з результату запиту, оскільки для порожніх складів вони беззмістовні.
На Рисунку 3.27 подано результат виконання цього коду.

code	name	address	chief_passport_no
3	Ужгородський	м. Ужгород, вул. Володимирська, 3	AA123456

Рис. 3.27: Порожні склади

Вправа 99

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть всю наявну інформацію про електростанції. Для АЕС також вкажіть їх діючість, а для ГЕС — тип.
2. Відобразіть всі дані про неатомні електростанції.
3. Відобразіть всі дані про електростанції, що не є ані АЕС, ані ГЕС.

3.2.10 Групування та агрегатні функції

Ми вже познайомилися з великої кількістю засобів, які надає мова SQL для вибірки даних з таблиць бази даних. З їх допомогою можна здійснювати пошук даних, об'єднувати дані кількох таблиць, впорядковувати їх тощо. Проте, розглянуті вище інструменти не дозволяють формувати звіти в тій формі, до якої ми звикли.

Нехай необхідно обчислити загальну кількість одиниць кожного товару на всіх складах мережі супермаркетів. Якби ми визначали цю інформацію вручну, ми б виписали перелік всіх наявних товарів та для кожного складу, де цей товар зберігається, додавали б до загальної кількості кількість одиниць цього товару на цьому складі. Мовою SQL ж можна надати такі інструкції системі управління базами даних, аби такі обчислення вона виконала самотужки. Для цього в цій мові передбачено блок групування даних.

Операція групування

Означення 47. *Групуванням записів таблиці за набором полів називатимемо співставлення кожному значенню цього набору полів сукупності значень решти полів тих рядків, що йому відповідають.*

Нехай маємо таблицю *holds*, зображену на Рисунку 3.28.

amount	warehouse_ code	goods_code
11	1	245558543
12	1	654646878
17	2	245558543
22	2	654687269

Рис. 3.28: Вміст таблиці *holds*

Тоді результатом операції групування записів цієї таблиці за полем *goods_ code* буде таблиця, подана на Рисунку 3.29, в якій кожному коду товару поставлено у відповідність сукупність значень двох інших полів: номеру складу та кількості цього товару на цьому складі.

goods_code	amount	warehouse_code
245558543	11	1
	17	2
654646878	12	1
654687269	22	2

Рис. 3.29: Результат групування записів таблиці *holds* за полем *goods_ code*

Якщо згадати, що таблиця — це n-арне відношення, то результат операції можна подати у вигляді наступної множини: $\{(245558543, \{11, 17\}, \{1, 2\}), (654646878, \{12\}, \{1\}), (654687269, \{22\}, \{2\})\}$.

Після виконання цієї операції ми маємо безпосередній доступ до полів, за якими здійснювалося групування. В даному випадку нам безпосередньо доступні значення стовпця *goods_ code*.

Як можна помітити з поданого вище відношення, за назвами решти полів (тут — *amount* та *warehouse_ code*) ми матимемо відтепер доступ не до окремих скалярних значень, а до сукупності значень, що відповідають певному коду товару.

Вправа 100

1. Побудуйте результат групування таблиці *placed_ on* (Рисунок 3.30a) за полем *river_ name*.
2. Побудуйте результат групування таблиці *power_ units* (Рисунок

3.30б) за полем *NPP_id*.

3. Побудуйте результат групування таблиці *power_units* (Рисунок 3.30б) за полями *NPP_id* та *reactor_type*.

river_name	HPP_id
Дніпро	1
Дніпро	2
Дністер	3
Ріка	4
Теребля	4

а)

number	reactor_type	power	NPP_id
1	ВВЕР-1000	1000	6
2	ВВЕР-1000	1000	6
3	ВВЕР-1000	1000	7
4	ВВЕР-1000	1000	7
1	ВВЕР-440	440	7
2	ВВЕР-440	440	7
1	РБМК-1000	1000	5
2	РБМК-1000	1000	5
3	РБМК-1000	1000	5
4	РБМК-1000	1000	5

б)

Рис. 3.30: Таблиці (а) *placed_on* та (б) *power_units*

Групування записів в SQL

Для здійснення операції групування в мові SQL надається спеціальна інструкція *GROUP BY*.

Синтаксис запиту з групуванням записів матиме наступний вигляд:

SELECT <перелік полів або виразів>
FROM <табличний вираз>
[**WHERE** <вираз логічного типу>]
[**GROUP BY** <перелік полів або виразів>]
[**ORDER BY** <перелік полів або виразів>];

Як і для решти розглянутих нами інструкцій, всі поля з переліку після ключових слів *GROUP BY* повинні належати до результату обчислення табличного виразу в блоці *FROM*, а вирази — базуватися на його полях.

Оскільки операція групування передбачає для деяких полів значення у вигляді сукупності, які не можуть бути безпосередньо виведені в результат запиту, у разі використання операції групування в переліках полів та

виразів інструкцій *SELECT* та *ORDER BY* безпосередньо можуть використовуватися тільки ті поля, за якими проводиться групування. Решта полів можуть бути включені в ці блоки як аргументи агрегатних функцій.

Агрегатні функції

Означення 48. *Агрегатна (групова) функція – це функція, що в якості параметра приймає множину значень і повертає єдине агреговане (узагальнююче) значення.*

Основні агрегатні функції, що є спільними для більшості систем управління базами даних, включені до наступного переліку:

- *COUNT()* — повертає кількість непорожніх (не *NULL*) значень в сукупності;
- *SUM()* — повертає суму непорожніх (не *NULL*) значень в сукупності;
- *MIN()* — повертає найменше серед непорожніх (не *NULL*) значень в сукупності;
- *MAX()* — повертає найбільше серед непорожніх (не *NULL*) значень в сукупності;
- *AVG()* — повертає середнє арифметичне серед непорожніх (не *NULL*) значень в сукупності.

Таким чином, обчислити загальну кількість одиниць кожного товару на всіх складах мережі супермаркетів можна за допомогою запиту, поданого в Лістингу 3.25.

```
SELECT holds.goods_code,
    SUM(holds.amount) AS total_amount
FROM holds
GROUP BY holds.goods_code;
```

Лістинг 3.25: Кількість товарів на складі

Результат виконання цього запиту зображено на Рисунку 3.31.

goods_code	total_amount
245558543	28
654646878	12
654687269	22

Рис. 3.31: Кількість товарів на складі

Код цього запиту природною мовою можна прочитати так: «вибрати коди товарів та їх загальну кількість з таблиці *holds*, групуючи записи за кодом товару».

Слід звернути увагу, що СУБД MySQL підтримує додаткову приховану агрегатну функцію, яка дозволяє спростити код запиту і використовувати під час виводу результату також ті поля, за якими групування не відбувається.

Розглянемо запит, код якого подано в Лістингу 3.26.

```
SELECT MIN(goods.name) AS name,
    SUM(holds.amount) AS total_amount
FROM holds INNER JOIN goods
    ON holds.goods_code=goods.code
GROUP BY holds.goods_code;
```

Лістинг 3.26: Кількість товарів на складі

Поданий вище запит відображає в побудованій нами раніше таблиці замість коду товару його назву, що дозволяє демонструвати інформацію більш наочно.

В даному випадку для відображення назви товару було використано агрегатну функцію *MIN*, яка на сукупності назв товару з усіма однаковими елементами повертатиме єдине можливе значення елемент. І якщо тут використання цієї функції не надто засмічує код, то при виводі всіх даних про товар ситуація дещо погіршується (Лістинг 3.27).

```
SELECT goods.code, MIN(goods.name) AS name,
    MIN(goods.price) AS price,
    MIN(goods.manufacturer_code)
        AS manufacturer_code,
    SUM(holds.amount) AS total_amount
FROM holds INNER JOIN goods
    ON holds.goods_code=goods.code
GROUP BY holds.goods_code;
```

Лістинг 3.27: Кількість товарів на складі

Оскільки випадки, коли записи групуються за ключем певної таблиці, не є рідкістю, MySQL в якості покращення коду дозволяє включати поля, що не входять до групування, в результат запиту. При цьому до таких полів зазвичай застосовується прихована агрегатна функція *FIRST()*, що повертає перше за порядком слідування значення в сукупності. Звісна річ, для сукупності з усіма однаковими елементами вона так само повертатиме єдине можливе значення.

Не рекомендуємо використовувати такий варіант запису, якщо ви намагаєтесь вивести поля, значення яких не є унікальним для кожного значення полів групування, адже результат в цьому випадку може бути непередбачуваним.

В такому разі код запиту буде значно простішим (Лістинг 3.28).

```
SELECT goods.code, goods.name,
    goods.price, goods.manufacturer_code,
    SUM(holds.amount) AS total_amount
FROM holds INNER JOIN goods
    ON holds.goods_code=goods.code
GROUP BY holds.goods_code;
```

Лістинг 3.28: Кількість товарів на складі

Результат виконання цього запиту зображено на Рисунку 3.32.

code	name	price	manufacturer_code	total_amount
245558543	Батон білий	14.30	3	28
654646878	Молоко українське	28.30	2	12
654687269	Сир голандський	231.10	2	22

Рис. 3.32: Кількість товарів на складі

Оскільки в даному випадку відображаються всі поля таблиці *goods*, запит можна ще більше спростити (Лістинг 3.29).

```
SELECT goods.*,
    SUM(holds.amount) AS total_amount
FROM holds INNER JOIN goods
    ON holds.goods_code=goods.code
GROUP BY holds.goods_code;
```

Лістинг 3.29: Кількість товарів на складі

Вправа 101

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Для кожної річки відобразіть її назву та кількість ГЕС, що на ній розташовані (назвіть поле *plants_amount*).

2. Для кожної атомної електростанції відобразіть її назву, загальну (назвіть поле *total_power*) та середню потужність її енергоблоків (назвіть поле *average_power*).
3. Для кожної атомної електростанції та кожного типу реактора відобразіть назву станції, тип реактора, загальну (назвіть поле *total_power*) та середню потужність її енергоблоків цієї станції, що мають такий тип реактора (назвіть поле *average_power*).
4. Для кожної атомної електростанції відобразіть її назву та коефіцієнт корисної дії — відношення її власної потужності до загальної потужності її енергоблоків (назвіть поле *ECE*). Розташуйте АЕС у порядку спадання цього коефіцієнту.

Агрегатні функції також можна використовувати і без групування. Тоді вони повертають відповідне узагальнене значення для всіх значень відповідного стовпця.

Наприклад, аби визначити середню ціну збережених в базі даних товарів, достатньо виконати код з Лістингу 3.30.

```
SELECT AVG(goods.price) AS average_price
FROM goods;
```

Лістинг 3.30: Середня ціна товарів

Результат виконання цього запиту зображено на Рисунку 3.33.

average_price
51.666667

Рис. 3.33: Середня ціна товарів

Вправа 102

За допомогою запиту до бази даних «Мережа супермаркетів», написаного мовою SQL, відобразіть загальну вартість всіх товарів, що зберігаються на складі (назвіть поле *total_cost*).

Вправа 103

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть загальну потужність наявних електростанцій (назвіть поле *total_power*).
2. Відобразіть частку активних атомних електростанцій (назвіть поле *active_part*).

Передумова запиту на групування

Нехай необхідно для кожного виробника відобразити кількість товарів ціною менше 20 гривень, які цей виробник виготовляє.

Слід зазначити, що інструкція *WHERE* в SQL-запиті недарма передує інструкції *GROUP BY*. Насправді, система управління базами даних спочатку відфільтрує записи результуючої таблиці відповідно до умови, поданої в блоці *WHERE*, а вже потім здійснюватиме групування та обчислення значень агрегатних функцій.

Отже, запит, що дає відповідь на поставлене запитання, матиме вигляд, поданий в Лістингу 3.31.

```
SELECT manufacturers.name,
    COUNT(goods.code) AS goods_amount
FROM manufacturers INNER JOIN goods
    ON manufacturers.code = goods.manufacturer_code
WHERE goods.price < 20
GROUP BY manufacturers.code;
```

Лістинг 3.31: Кількість товарів ціною менше 20 гривень

Розглянемо детальніше процес отримання результату виконання даного запиту.

code	name	address
1	Київхліб	м. Київ, вул. Межигірська, 83
2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.34: Таблиця *manufacturers*

code	name	price	manufacturer_ code
245558543	Батон білий	14,3	3
654646878	Молоко українське	28,3	2
654687269	Сир голандський	231,1	2
821556286	Хліб білий	12,8	1
863543436	Сирок плавлений	13,2	2
989651445	Хліб житній	10,3	1

Рис. 3.35: Таблиця *goods*

Отже, нехай ми маємо таблиці *manufacturers* та *goods*, зображені на Рисунках 3.34–3.35.

Їх внутрішнє об'єднання за умовою $manufacturers.code = goods.manufacturer\_code$ матиме вигляд, поданий на Рисунку 3.36.

goods. code	goods.name	price	manufacturer_ code	manufacturers. code	manufacturers.name	address
245558543	Батон білий	14,3	3	3	Кулиничі	м. Харків, вул. Грищенка, 17
654646878	Молоко українське	28,3	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
654687269	Сир голандський	231,1	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
821556286	Хліб білий	12,8	1	1	Київхліб	м. Київ, вул. Межигірська, 83
863543436	Сирок плавлений	13,2	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
989651445	Хліб житній	10,3	1	1	Київхліб	м. Київ, вул. Межигірська, 83

Рис. 3.36: Внутрішнє об'єднання таблиць *manufacturers* та *goods*

Після цього СУБД здійснює фільтрацію отриманої таблиці та залишає тільки ті записи, для яких умова $goods.price < 20$ є істинною (Рисунок 3.37).

goods. code	goods.name	price	manufacturer_ code	manufacturers. code	manufacturers.name	address
245558543	Батон білий	14,3	3	3	Кулиничі	м. Харків, вул. Грищенка, 17
821556286	Хліб білий	12,8	1	1	Київхліб	м. Київ, вул. Межигірська, 83
863543436	Сирок плавлений	13,2	2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
989651445	Хліб житній	10,3	1	1	Київхліб	м. Київ, вул. Межигірська, 83

Рис. 3.37: Товари ціною менше 20 гривень

Отримана вище таблиця далі підлягає групуванню, що призводить до отримання таблиці, поданої на Рисунку 3.38. Таким чином до груп вже потрапляють тільки товари, ціна яких нижча за 20 гривень.

Вже після цього здійснюється підрахунок кількості товарів для кожного виробника, який призводить до отримання таблиці, поданої на Рисунку 3.39.

manufacturers.code	goods.code	goods.name	price	manufacturer_code	manufacturers.name	address
1	821556286	Хліб білий	12,8	1	Київхліб	м. Київ, вул. Межигірська, 83
	989651445	Хліб житній	10,3	1	Київхліб	м. Київ, вул. Межигірська, 83
2	863543436	Сирок плавлений	13,2	2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
3	245558543	Батон білий	14,3	3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.38: Виробники та їх товари ціною менше 20 гривень

name	goods_amount
Київхліб	2
1-й Київський молокозавод	1
Кулиничі	1

Рис. 3.39: Виробники та кількість товарів ціною менше 20 гривень

Вправа 104

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Для кожної річки відобразіть її назву та кількість гідроакумулюючих ГЕС, що на ній розташовані (назвіть поле *plants_amount*). Річки, на яких такі станції відсутні, не відображайте.
2. Для кожної атомної електростанції визначте загальну (назвіть поле *total_power*) та середню потужність її енергоблоків з реакторами типу «РБМК» (назвіть поле *average_power*). Електростанції без таких енергоблоків не відображайте.
3. Для кожної електростанції визначте загальну (назвіть поле *total_power*) та середню потужність її енергоблоків з реакторами типу «РБМК» (назвіть поле *average_power*). Для неатомних електростанцій, або АЕС без таких енергоблоків в цьому полі відобразіть *NULL*.

Післяумова запиту на групування

Нехай необхідно відобразити виробників, що виготовляють більше одного товару ціною менше 20 гривень.

Звісна річ, для вирішення цього завдання слід спочатку для кожного виробника обчислити кількість товарів ціною менше 20 гривень, які він виготовляє. Запит для цього ми побудували вище.

Яким чином тепер встановити умову, що обчислена кількість повинна бути більшою за один? Адже перевірка умови, поданої в блоці *WHERE*, здійснюється до групування, а кількість стане відомою тільки після нього. Для цього в SQL передбачена ще одна інструкція *HAVING*, яка власне і надає можливість відбирати записи, отримані в результаті групування та агрегатних обчислень.

Синтаксис запиту з групуванням записів та післяумовою матиме наступний вигляд:

```
SELECT <перелік полів або виразів>
FROM <табличний вираз>
[WHERE <вираз логічного типу>]
[GROUP BY <перелік полів або виразів>]
[HAVING <вираз логічного типу>]
[ORDER BY <перелік полів або виразів>];
```

HAVING-умова може містити усі ті ж вирази, що й перелік полів та виразів інструкції *SELECT*, включаючи агрегатні функції. Винятком є лише поля, за якими не проводиться групування, якщо вони не включені до агрегатних функцій.

Для вище згаданого прикладу запит матиме вигляд, код якого подано в Лістингу 3.32.

```
SELECT manufacturers.name
FROM manufacturers INNER JOIN goods
    ON manufacturers.code = goods.manufacturer_code
WHERE goods.price < 20
GROUP BY manufacturers.code
HAVING COUNT(goods.code) > 1;
```

Лістинг 3.32: Виробники з більш, ніж одним дешевим товаром

Результат виконання цього запиту зображено на Рисунку 3.40.

name
Київхліб

Рис. 3.40: Виробники з більш, ніж одним дешевим товаром

Код цього запиту можна прочитати так: «вибрати назви виробників з таблиці *manufacturers*, пов'язаної з *goods* за умовою *manufacturers.code* = *goods.manufacturer_ code*, де ціна товару менша за 20 гривень, групуючи записи за кодом товару, що мають кількість товарів більшу за одиницю».

Вправа 105

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Визначте назви електростанцій, що розташовані на двох річках одночасно.
2. Визначте назви атомних електростанцій, загальна потужність енергоблоків яких більша за 2000 МВт.
3. Визначте назви річок, середня потужність ГЕС на яких менша за 100 МВт.

3.2.11 Особливості запитів на групування в СУБД Access

Запити на групування в Microsoft Access в більшості деталей є цілком традиційними, тому всі вищезазначені запити (окрім тих, що стосувалися особливостей MySQL) успішно виконуються в середовищі Access.

В порівнянні з традиційним базовим набором агрегатних функцій, основні групові функції Access доповнюються двома додатковими:

- *FIRST()* — повертає перше з значень в множині в порядку слідування;
- *LAST()* — повертає останнє з значень в множині в порядку слідування.

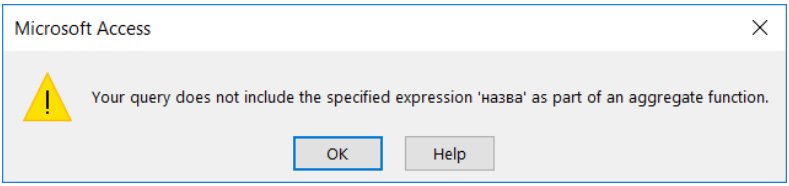


Рис. 3.41: Запит не містить вираз як частину агрегатної функції

На відміну від MySQL, ці агрегатні функції не є прихованими. Це означає, що *в запитах з групуванням безпосереднє використання в інструкціях*

SELECT, ORDER BY та HAVING полів, що не входять до полів групування, заборонено і призводить до помилки (Рисунок 3.41).

А тому, аби відобразити усі дані товару поряд з загальною кількістю його одиниць на складах, доведеться використовувати для більшості полів таблиці *goods* агрегатні функції (Лістинг 3.33).

```
SELECT goods.code, FIRST(goods.name) AS name,
    FIRST(goods.price) AS price,
    FIRST(goods.manufacturer_code)
        AS manufacturer_code,
    SUM(holds.amount) AS total_amount
FROM holds INNER JOIN goods
    ON holds.goods_code=goods.code
GROUP BY holds.goods_code;
```

Лістинг 3.33: Кількість товарів на складі

Тут замість функції *FIRST* можуть бути використані також функції *LAST*, *MIN* та *MAX*, оскільки всі вони на сукупності з усіх однакових значень повертають єдиний можливий результат.

Також одним із рішень даної проблеми може бути додавання в блок групування тих полів, які однозначно визначаються рештою полів групування. Це не вплине на результат, проте дозволить розвантажити блок *SELECT* (Лістинг 3.34).

```
SELECT goods.code, goods.name, goods.price,
    goods.manufacturer_code,
    SUM(holds.amount) AS total_amount
FROM holds INNER JOIN goods
    ON holds.goods_code=goods.code
GROUP BY holds.goods_code, goods.name, goods.price,
    goods.manufacturer_code;
```

Лістинг 3.34: Кількість товарів на складі

Проте цей варіант розв'язання часто призводить до змістових помилок у запитах, оскільки доволі легко порушити таким чином результат операції виконання. Тому *не рекомендуємо додавати до блоку GROUP BY зайві поля заради їх виводу.*

Вправа 106

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання в середовищі Microsoft Access:

1. Відобразіть частку активних атомних електростанцій (назвіть поле *active_part*).
2. Для кожної атомної електростанції та кожного типу реактора загальну (назвіть поле *total_power*) та середню потужність її енергоблоків цієї станції, що мають такий тип реактора (назвіть поле *average_power*).
3. Визначте назви електростанцій, що розташовані на двох річках одночасно.

3.2.12 Прибирання дублікатів в запитах

Припустимо, що нам необхідно визначити виробників, які виготовляють товари ціною менше 20 гривень. Отримати відповідь на це запитання доволі легко за допомогою звичайного запиту з умовою (Лістинг 3.35).

```
SELECT manufacturers.*
FROM manufacturers INNER JOIN goods
    ON manufacturers.code=goods.manufacturer_code
WHERE goods.price < 20
ORDER BY manufacturers.name;
```

Лістинг 3.35: Виробники дешевих товарів

В результаті цього запиту ми отримаємо по одному рядку на кожен товар з ціною менше 20 гривень із зазначеним виробником. Вище ми приховали інформацію про товари, адже вона не вимагається в постановці завдання.

Результат виконання цього запиту зображено на Рисунку 3.42.

code	name ▲ 1	address
2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
1	Київхліб	м. Київ, вул. Межигірська, 83
1	Київхліб	м. Київ, вул. Межигірська, 83
3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.42: Виробники, які виготовляють товари ціною менше 20 гривень

Справді, оскільки *Київхліб* виготовляє одразу два товари ціною менше 20 гривень, в результаті даного запиту від трапляється двічі. Як в такому разі залишити лише одне згадування про кожного виробника?

Перший варіант прибирання дублікатів — використати групування записів. Щойно ми згрупуємо наші записи за кодом виробника, ми отримаємо рівно по одному рядку на кожне значення цього коду — власне, той ефект, який нам і був потрібен (Лістинг 3.36).

```
SELECT manufacturers.*
FROM manufacturers INNER JOIN goods
    ON manufacturers.code=goods.manufacturer_code
WHERE goods.price < 20
GROUP BY manufacturers.code
ORDER BY manufacturers.name;
```

Лістинг 3.36: Виробники дешевих товарів

Результат виконання цього запиту зображено на Рисунку 3.43.

code	name ▲ 1	address
2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
1	Київхліб	м. Київ, вул. Межигірська, 83
3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.43: Виробники, які виготовляють товари ціною менше 20 гривень

Проте, вдаватися до таких хитрощів зовсім не обов'язково, адже мова SQL надає спеціальну інструкцію *DISTINCT*, яка прибирає дублікати рядків, що виводяться як результат (подані в блоці *SELECT*). Цю інструкцію слід вміщувати в коді одразу після ключового слова *SELECT*:

```
SELECT [DISTINCT] <перелік полів або виразів>
FROM <табличний вираз>
[WHERE <вираз логічного типу>]
[GROUP BY <перелік полів або виразів>]
[HAVING <вираз логічного типу>]
[ORDER BY <перелік полів або виразів>];
```

Код запиту для визначення виробників, які виготовляють товари ціною менше 20 гривень, подано в Лістингу 3.37.

```
SELECT DISTINCT manufacturers.*
FROM manufacturers INNER JOIN goods
```

```
    ON manufacturers.code=goods.manufacturer_code
WHERE goods.price < 20
ORDER BY manufacturers.name;
```

Лістинг 3.37: Виробники дешевих товарів

Вправа 107

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть в алфавітному порядку назви гідроелектростанцій, що розташовані на річках Дністер, Ріка та Теребля.
2. Відобразіть в алфавітному порядку назви атомних електростанцій, що мають енергоблоки з типом реактора «ВВЕР».

3.2.13 Обмеження розміру результату

За промовчанням більшість реляційних систем управління базами даних відображає усі дані, для яких виконуються умови, вміщені до вихідного коду запиту. Проте деколи потрібно отримати перші п'ять, десять записів результату чи рівно його половину — що ж робити в такому разі?

Мова SQL в цьому випадку пропонує спеціальну інструкцію. Оскільки її синтаксис повністю відрізняється в MySQL та Access, розглянемо ці СУБД окремо.

В якості прикладу розглядатимемо визначення трьох найдорожчих товарів мережі супермаркетів.

Обмеження кількості записів в MySQL

В системі управління базами даних MySQL за обмеження кількості записів в результаті запиту відповідає інструкція *LIMIT*. Синтаксис запиту з обмеженням кількості рядків є наступним:

```
SELECT [DISTINCT] <перелік полів або виразів>
FROM <табличний вираз>
[WHERE <вираз логічного типу>]
[GROUP BY <перелік полів або виразів>]
[HAVING <вираз логічного типу>]
[ORDER BY <перелік полів або виразів>]
[LIMIT [<зсув> ,] <кількість записів>];
```

Це ключове слово недарма міститься після інструкції *ORDER BY*: система управління базами даних спочатку впорядковує записи, а потім повертає в якості результату кількість, подану в блоці *LIMIT*.

Отже, аби визначити три найдорожчі товари нам необхідно впорядкувати всі товари за спаданням ціни та обмежити результат запиту до трьох рядків (Лістинг 3.38).

```
SELECT *
FROM goods
ORDER BY goods.price DESC
LIMIT 3;
```

Лістинг 3.38: Трійка найдорожчих товарів

Результат виконання цього запиту зображено на Рисунку 3.44.

code	name	price 1	manufacturer_code
654687269	Сир голандський	231.10	2
654646878	Молоко українське	28.30	2
245558543	Батон білий	14.30	3

Рис. 3.44: Трійка найдорожчих товарів

Слід зауважити, що якщо, наприклад, третій та четвертий товари мали б однакову ціну, MySQL все одно б відобразив три товари відповідно до того, як вони будуть відображені в результаті (зазвичай, якщо комбінації полів з блоку *ORDER BY* не вистачає для сортування рядків, впорядкування в межах однакових значень цієї комбінації відбувається за ключами об'єднаних в запиті таблиць).

За допомогою цього ж ключового слова в MySQL можна відображати не тільки перші N записів, а й, наприклад, три записи, починаючи з другого.

Щоб, наприклад, визначити три найдорожчі товари, починаючи з другого, нам необхідно впорядкувати всі товари за спаданням ціни та обмежити результат запиту до трьох рядків зі зсувом 1 (Лістинг 3.39).

```
SELECT *
FROM goods
ORDER BY goods.price DESC
LIMIT 1, 3;
```

Лістинг 3.39: Трійка найдорожчих товарів, починаючи з другого

Результат виконання цього запиту зображено на Рисунку 3.45.

code	name	price ▼ 1	manufacturer_code
654646878	Молоко українське	28.30	2
245558543	Батон білий	14.30	3
863543436	Сирок плавлений	13.20	2

Рис. 3.45: Трійка найдорожчих товарів, починаючи з другого

Обмеження кількості записів у відсотковому вигляді в MySQL за допомогою штатних засобів здійснити не вдасться.

Вправа 108

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть дані про три найбільш потужні електростанції.
2. Визначте назву річки (назвіть поле *river_name*), на якій розташовано найбільше гідроелектростанцій (використайте запит, побудований у Вправі 101).
3. Визначте назву та ККД атомної електростанції з найменшим коефіцієнтом корисної дії (ККД) — відношенням її власної потужності до загальної потужності її енергоблоків (назвіть поле *ECE*) (використайте запит, побудований у Вправі 101).

Обмеження кількості записів в Microsoft Access

В системі управління базами даних Microsoft Access за обмеження кількості записів в результаті запиту відповідає інструкція *TOP*. Синтаксис запиту з обмеженням кількості рядків є наступним:

SELECT [**DISTINCT**] [TOP <кількість записів> [PERCENT]] <перелік полів або виразів>
FROM <табличний вираз>
[**WHERE** <вираз логічного типу>]
[**GROUP BY** <перелік полів або виразів>]
[**HAVING** <вираз логічного типу>]
[**ORDER BY** <перелік полів або виразів>];

Аналогічно до MySQL, Access спочатку впорядковує записи, а потім повертає в якості результату кількість, подану в блоці *TOP*.

Отже, визначення трьох найдорожчих товарів зводиться до наступного запиту (Лістинг 3.40).

```
SELECT TOP 3 *
FROM goods
ORDER BY goods.price DESC;
```

Лістинг 3.40: Трійка найдорожчих товарів

Результат виконання цього запиту зображено на Рисунку 3.46.

	code	name	price	manufacture
	654687269	Сир голандськ	231,10 ₴	2
	654646878	Молоко україн	28,30 ₴	2
	245558543	Батон білий	14,30 ₴	3
*	0		0,00 ₴	0

Рис. 3.46: Трійка найдорожчих товарів

Слід зауважити, що якщо, наприклад, третій та четвертий товари мали б однакову ціну, Access відобразив би чотири товари. Загалом, при обмеженні кількості запитів в результаті в цій СУБД записи з однаковим значенням комбінації полів сортування (в блоці *ORDER BY*) не розрізняються. Якщо один з таких записів потрапляє в обмежений діапазон рядків, туди ж включаються всі інші записи з тим же значенням комбінації полів сортування.

За допомогою цього ж ключового слова в Access можна відображати певну частину записів у відсотковому вигляді.

Щоб, наприклад, визначити чверть найдорожчих товарів, нам необхідно впорядкувати всі товари за спаданням ціни та обмежити результат запиту до 25 відсотків за допомогою додавання ключового слова PERCENT після числа 25 (Лістинг 3.41).

```
SELECT TOP 25 PERCENT *
FROM goods
ORDER BY goods.price DESC;
```

Лістинг 3.41: Чверть найдорожчих товарів

Результат виконання цього запиту зображено на Рисунку 3.47.

Обмеження кількості записів зі зміщенням, як це властиво MySQL, в Microsoft Access штатними засобами не підтримується.

	code	name	price	manufacture
	654687269	Сир голандськ	231,10 ₴	2
	654646878	Молоко україн	28,30 ₴	2
*	0		0,00 ₴	0

Рис. 3.47: Чверть найдорожчих товарів

Вправа 109

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть всі дані про три найбільш потужні електростанції.
2. Визначте назву річки, на якій розташовано найбільше гідроелектростанцій (використайте запит, побудований у Вправі 101).
3. Визначте назву та ККД 30% атомних електростанцій з найменшим коефіцієнтом корисної дії (ККД) — відношенням її власної потужності до загальної потужності її енергоблоків (назвіть поле *ECE*) (використайте запит, побудований у Вправі 101).

3.2.14 Підзапити та їх використання

Вище ми розглянули велику частину апарату мови SQL для здійснення запитів на вибірку. Проте, як в програмуванні деколи вирішення підзадачі слід оформити у вигляді підпрограми, в багатьох випадках для коректної побудови запиту необхідно оформити деяку підзадачу у вигляді підзапиту, аби скористатися ним при отриманні кінцевого результату.

Означення 49. *Підзапит – це запит до бази даних, що вирішує певну підзадачу, необхідну для досягнення кінцевого результату.*

Підзапити в коді запису виділяють дужками.

Залежно від типу, підзапити можуть використовуватися в блоках *SELECT*, *FROM*, *WHERE*, *HAVING*, *ORDER BY*.

За характером результату виконання розрізняють наступні типи підзапитів:

- *підзапит-значення* (підзапит з одним полем та одним записом);

- *підзапит-стовпець* (підзапит з одним полем та нуль або більше записами);
- *підзапит-таблиця* (підзапит з одним або більше полями).

Слід зазначити, що кожен попередній тип включається в наступний, тобто будь-який підзапит-значення є підзапитом-стовпцем і тобто будь-який підзапит-стовпець є підзапитом-таблицею. Далі ми звертатимемо на увагу лише на ті властивості, які відрізняють представників цих типів між собою.

Підзапити-значення та їх використання

Підзапити, що повертають скалярне значення, можуть використовуватися в умовах, виразах, при сортуванні тощо — всюди, де можуть використовуватися інші вирази, як звичайні скалярні значення.

Наприклад, запит для визначення товару з максимальною ціною може мати вигляд, поданий у Лістингу 3.42.

```
SELECT *
FROM goods
WHERE goods.price = (
    SELECT MAX(goods.price)
    FROM goods
);
```

Лістинг 3.42: Товар з максимальною ціною

В цьому запиті підзапит обчислює значення максимальної ціни, а потім основний запит відображає товари, ціна яких рівна обчисленій.

Результат виконання запиту зображено на Рисунку 3.48.

code	name	price	manufacturer_code
654687269	Сир голандський	231.10	2

Рис. 3.48: Товар з максимальною ціною

Зокрема підзапити-значення можуть бути використані й безпосередньо для виводу значень в блоці *SELECT*.

Наприклад, якщо потрібно відобразити відхилення ціни кожного товару від середньої, достатньо запустити запит, код якого подано в Лістингу 3.43.

```
SELECT *, goods.price - (
        SELECT AVG(goods.price)
```

```
        FROM goods
    ) AS deviation
FROM goods;
```

Лістинг 3.43: Відхилення від середнього

Результат виконання запиту зображено на Рисунку 3.49.

code	name	price	manufacturer_code	deviation
245558543	Батон білий	14.30	3	-37.366667
654646878	Молоко українське	28.30	2	-23.366667
654687269	Сир голандський	231.10	2	179.433333
821556286	Хліб білий	12.80	1	-38.866667
863543436	Сирок плавлений	13.20	2	-38.466667
989651445	Хліб житній	10.30	1	-41.366667

Рис. 3.49: Відхилення від середнього

Вправа 110

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть всі дані про електростанції, потужність яких вища середньої.
2. Для кожної електростанції відобразіть її назву та частку її потужності в загальній потужності всіх електростанцій (назвіть поле *power_share*). Впорядкуйте станції за спаданням обчисленої частки.

Області видимості в SQL

Якщо раніше у нас не виникало питань в якій ділянці коду видно таблицю, а в якій — ні, адже ми мали справу лише з однією ділянкою, то зараз, при введенні підзапитів, постає запитання де які таблиці буде видно.

Означення 50. *Область видимості — це ділянка запиту, де існує окрема множина імен, пов'язаних із певними даними (таблицями та їх полями, синонімами тощо).*

Нехай для кожного товару нам потрібно дізнатися скільки товарів мають меншу за його ціну. Один із варіантів рішення цього завдання вимагає по-

будову підзапиту в блоці *SELECT*, який підраховуватиме кількість товарів з ціною, меншою за задану.

Розглянемо код, поданий в Лістингу 3.44.

```
SELECT *, (
        SELECT COUNT(goods.code)
        FROM goods
        WHERE goods.price < <значення>
    )
FROM goods;
```

Лістинг 3.44: Кількість товарів з ціною, меншою за задану

Якщо у підзапит ми могли б підставити ціну товару з основного запиту, то завдання було б цілком вирішене: для кожного товару здійснювався б підрахунок кількості товарів з ціною, меншою за ціну цього товару. Виявляється, така можливість є.

Кожне ім'я таблиці (сама таблиця чи синонім) в SQL-запиті має свою область видимості: до нього можна звернутися в будь-якому підзапиті запиту, в якому його було оголошено в блоці *FROM*. В тому числі це ім'я видно й у підзапитах підзапитів і т.д. Єдиний виняток складає ситуація затінення змінної: якщо в основному запиті і в підзапиті одночасно оголошено деяке ім'я, то при зверненні до цього імені в підзапиті звернення відбуватиметься до більш локального імені.

В наведеному вище коді саме через затінення імені ми не можемо написати умову *goods.price* < *goods.price*: тут в обох випадках відбуватиметься звернення до таблиці *goods* з підзапиту і умова буде завжди хибною. То як вийти з цієї ситуації? Виявляється, дуже просто: слід надати якійсь із таблиць синонім — тоді вона втратить своє початкове ім'я goods і в підзапиті ці дві таблиці будуть видимі під різними іменами.

Рекомендується завжди надавати синонім таблиці, що розташована в підзапиті.

Якщо надати таблиці *goods* з підзапиту синонім *goods_1*, то код запиту для визначення кількості товарів з меншою ціною набуде вигляду, поданого в Лістингу 3.45.

```
SELECT goods.*, (
        SELECT COUNT(goods_1.code)
        FROM goods AS goods_1
        WHERE goods_1.price < goods.price
    ) AS position
```

```
FROM goods;
```

Лістинг 3.45: Кількість товарів з ціною, меншою за задану

Результат виконання запиту зображено на Рисунку 3.50.

code	name	price	manufacturer_code	position
245558543	Батон білий	14.30	3	3
654646878	Молоко українське	28.30	2	4
654687269	Сир голандський	231.10	2	5
821556286	Хліб білий	12.80	1	1
863543436	Сирок плавлений	13.20	2	2
989651445	Хліб житній	10.30	1	0

Рис. 3.50: Кількість товарів з ціною, меншою за задану

Якщо впорядкувати товари за обчисленим значенням та додати до нього одиницю, то отримаємо нумерацію товарів за зростанням їх ціни (Лістинг 3.46).

```
SELECT (
        SELECT COUNT(goods_1.code)
        FROM goods AS goods_1
        WHERE goods_1.price < goods.price
    ) + 1 AS position, goods.*
FROM goods
ORDER BY position;
```

Лістинг 3.46: Нумерований перелік товарів

Дійсно, номер товару у списку за зростанням ціни — це кількість товарів з меншою ціною плюс 1. Результат виконання запиту зображено на Рисунку 3.51.

position ▲ 1	code	name	price	manufacturer_code
1	989651445	Хліб житній	10.30	1
2	821556286	Хліб білий	12.80	1
3	863543436	Сирок плавлений	13.20	2
4	245558543	Батон білий	14.30	3
5	654646878	Молоко українське	28.30	2
6	654687269	Сир голандський	231.10	2

Рис. 3.51: Нумерований перелік товарів

Вправа 111

Створіть ідентичний нумерований перелік товарів, використовуючи групування та перехресне об'єднання.

Вправа 112

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть нумерований перелік електростанцій (поле нумерації назвіть N) у порядку спадання їх потужності.
2. Для кожної гідроелектростанції відобразіть її назву, тип та частку, яку становить її потужність в загальній потужності ГЕС того ж типу (поле назвіть *power_share*).
3. Відобразіть нумерований перелік енергоблоків АЕС (поле нумерації назвіть N) у порядку спадання їх потужності. Зверніть увагу, що енергоблоки з однаковою потужністю повинні мати різні номери відповідно до їх появи в переліку.

Підзапити-стовпці та їх використання

Підзапити, що повертають сукупність скалярних значень, використовують зазвичай в умовах блоків *WHERE* та *HAVING*. Безпосереднє їх використання, як підзапитів-значень, звісно ж неможливе, проте їх можна використовувати в умовах в комбінації з операторами порівняння (=, >, <, >=, <=, <>) та ключовими словами *ALL*, *ANY*, *SOME* або *IN*. Розглянемо кожен з даних модифікаторів детальніше.

Логічний вираз з використанням модифікатора *ALL* має наступний синтаксис:

<вираз> <оператор порівняння> **ALL** <підзапит-стовпець>

Вищевказаний логічний вираз прийматиме значення ІСТИНА (*TRUE*) тільки тоді, коли для кожного значення у підзапиті-стовпці виконуватиметься умова:

<вираз> <оператор порівняння> <значення>

Наприклад, вираз

```
50 >= ALL (SELECT goods.price FROM goods)
```

буде істинним, якщо число 50 більше або рівне за ціну будь-якого товару.

price
23
17
48
3
10

а)

price
51
28
36
23
6

б)

price
59
73
66
103
92

в)

Рис. 3.52: Приклади результатів підзапиту-стовпця

Для прикладів результатів підзапиту-стовпця, поданих на Рисунку 3.52, для стовпця *а* вказаний вище вираз матиме значення ІСТИНА, а для стовпців *б* та *в* — ХИБНІСТЬ, оскільки вони містять значення, більші за 50.

Нехай необхідно відобразити інформацію про товар з найвищою ціною. Це рівносильно тому, що ціна такого товару є більшою або рівною ціні будь-якого іншого товару, а тому один із варіантів розв'язання матиме вигляд, поданий в Лістингу 3.47.

```
SELECT goods.*
FROM goods
WHERE goods.price >= ALL (
    SELECT goods.price
    FROM goods
);
```

Лістинг 3.47: Найдорожчий товар

Результат виконання цього запиту зображено на Рисунку 3.53.

code	name	price	manufacturer_code
654687269	Сир голандський	231.10	2

Рис. 3.53: Найдорожчий товар

Логічний вираз з використанням модифікатора *ANY* має наступний синтаксис:

<вираз> <оператор порівняння> **ANY** <підзапит-стовпець>

Вищевказаний логічний вираз прийматиме значення ІСТИНА (*TRUE*) тільки тоді, коли принаймні для одного значення у підзапиті-стовпці виконуватиметься умова:

<вираз> <оператор порівняння> <значення>

Наприклад, вираз

```
50 >= ANY (SELECT goods.price FROM goods)
```

буде істинним, якщо число 50 більше або рівне за ціну принаймні деякого товару.

Для прикладів результатів підзапиту-стовпця, поданих на Рисунку 3.52, для стовпців *а* та *б* вказаний вище вираз матиме значення ІСТИНА, а для стовпця *в* — ХИБНІСТЬ, оскільки він містить лише значення, більші за 50.

Нехай необхідно відобразити інформацію про товари з не найнижчою ціною. Це рівносильно тому, що ціна такого товару є більшою за ціною хоча б одного з товарів, а тому один із варіантів розв'язання матиме вигляд, поданий в Лістингу 3.48.

```
SELECT goods.*
FROM goods
WHERE goods.price > ANY (
    SELECT goods.price
    FROM goods
);
```

Лістинг 3.48: Не найдешевші товари

Результат виконання цього запиту зображено на Рисунку 3.54.

code	name	price	manufacturer_code
245558543	Батон білий	14.30	3
654646878	Молоко українське	28.30	2
654687269	Сир голандський	231.10	2
821556286	Хліб білий	12.80	1
863543436	Сирок плавлений	13.20	2

Рис. 3.54: Не найдешевші товари

Модифікатор *SOME* є синонімом модифікатора *ANY*, а тому його поведінка ідентична наведеній вище.

Логічний вираз з використанням оператора *IN* має наступний синтаксис:

<вираз> **IN** <підзапит-стовпець>

Вищевказаний логічний вираз прийматиме значення ІСТИНА (*TRUE*) тільки тоді, коли для деякого значення у підзапиті-стовпці виконуватиметься умова:

<вираз> = <значення>

Іншими словами, це вираз є істинним тоді і тільки тоді, коли значення відповідного виразу міститься в сукупності значень підзапиту-стовпця.

Наприклад, вираз

```
23 IN (SELECT goods.price FROM goods)
```

буде істинним, якщо число 23 є ціною хоча б одного з товарів.

Для прикладів результатів підзапиту-стовпця, поданих на Рисунку 3.52, для стовпців *а* та *б* вказаний вище вираз матиме значення ІСТИНА, оскільки вони містять значення 23, а для стовпця *в* — ХИБНІСТЬ, оскільки він цього значення не містить.

Нехай необхідно відобразити інформацію про виробників товарів, ціна яких менша за 20 гривень. Це рівносильно тому, що код виробника є серед кодів виробників товарів ціною менше 20 гривень, а тому один із варіантів розв'язання матиме вигляд, поданий в Лістингу 3.49.

```
SELECT manufacturers.*
FROM manufacturers
WHERE manufacturers.code IN (
    SELECT goods.manufacturer_code
    FROM goods
    WHERE goods.price < 20
);
```

Лістинг 3.49: Виробники дешевих товарів

Результат виконання цього запиту зображено на Рисунку 3.55.

code	name	address
1	Київхліб	м. Київ, вул. Межигірська, 83
2	1-й Київський молокозавод	м. Київ, вул. Жилянська, 47
3	Кулиничі	м. Харків, вул. Грищенка, 17

Рис. 3.55: Виробники дешевих товарів

Двоїстий оператор *NOT IN* має аналогічний синтаксис:

<вираз> **NOT IN** <підзапит-стовпець>

Протилежно до оператора *IN*, вищевказаний логічний вираз прийматиме значення ІСТИНА (*TRUE*) тільки тоді, коли для всіх значень у підзапиті-стовпці виконуватиметься умова:

<вираз> <> <значення>

Іншими словами, це вираз є істинним тоді і тільки тоді, коли значення відповідного виразу відсутнє в сукупності значень підзапиту-стовпця.

Наприклад, вираз

```
23 NOT IN (SELECT goods.price FROM goods)
```

буде істинним, якщо число 23 не є ціною жодного з товарів.

Для прикладів результатів підзапиту-стовпця, поданих на Рисунку 3.52, для стовпців *а* та *б* вказаний вище вираз матиме значення ХИБНІСТЬ, оскільки вони містять значення 23, а для стовпця *в* — ІСТИНА, оскільки він цього значення не містить.

Нехай необхідно відобразити інформацію про склади, що не зберігають жодного товару. Це рівносильно тому, що код такого складу відсутній серед кодів складів в таблиці *holds*, а тому один із варіантів розв'язання матиме вигляд, поданий в Лістингу 3.50.

```
SELECT warehouses.*
FROM warehouses
WHERE warehouses.code NOT IN (
    SELECT holds.warehouse_code
    FROM holds
);
```

Лістинг 3.50: Порожні склади

Результат виконання цього запиту зображено на Рисунку 3.56.

code	name	address	chief_passport_no
3	Ужгородський	м. Ужгород, вул. Володимирська, 3	AA123456

Рис. 3.56: Порожні склади

Оператор *NOT IN* є синонімом комбінації операторів <> *ALL*.

Вправа 113

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть назви атомних електростанцій, всі енергоблоки яких містять реактори типу «ВВЕР-1000».
2. Відобразіть назви та потужності електростанцій, які не є ані найбільш, ані найменш потужними.
3. Відобразіть назви електростанцій, що розташовані на річках довжиною менше 100 км.
4. Відобразіть назви електростанцій, що не є ані атомними, ані гідроелектростанціями.
5. Відобразіть назви атомних електростанцій, всі енергоблоки яких містять реактори одного типу.

Підзапити-таблиці та їх використання

Підзапити, що повертають таблицю, використовують у виразах в блоці *FROM*, а також в умовах блоків *WHERE* та *HAVING* у супроводі оператора *EXISTS*.

Зокрема, підзапити-таблиці можуть брати участь в об'єднаннях так само, як і звичайні таблиці. Наприклад, аби визначити товари, що виготовляються київськими виробниками, можна скористатися запитом, код якого подано в Лістингу 3.51.

```
SELECT goods.*
FROM goods INNER JOIN (
        SELECT manufacturers.*
        FROM manufacturers
        WHERE manufacturers.address LIKE "%Київ%"
    ) AS kyiv_manufacturers
    ON goods.manufacturer_code =
        kyiv_manufacturers.code;
```

Лістинг 3.51: Товари київських виробників

В цьому запиті спочатку формується таблиця київських виробників, яка отримує псевдонім *kyiv_manufacturers*, а вже потім здійснюється об'єднання з цією таблицею таблиці *goods*. Оскільки використано внутрішнє об'єднання,

в результат включаються тільки ті товари, що зв'язані з київськими виробниками.

Результат виконання цього запиту зображено на Рисунку 3.57.

code	name	price	manufacturer_code
654646878	Молоко українське	28.30	2
654687269	Сир голандський	231.10	2
821556286	Хліб білий	12.80	1
863543436	Сирок плавлений	13.20	2
989651445	Хліб житній	10.30	1

Рис. 3.57: Товари київських виробників

Слід зазначити, що надання псевдоніму підзапиту в блоці FROM є обов'язковим.

Також табличні підзапити можуть бути аргументами операції *UNION*, результатом якої є об'єднання рядків без повторень двох таблиць з однаковою кількістю полів сумісних типів. Результат об'єднання таблиць теж є таблицею, а тому також може використовуватися в місцях, де використовуються підзапити-таблиці.

Нехай нам необхідно побудувати запит, який би відображав всі дані про працівників, але у випадку, коли працівник не має начальника, у відповідному полі замість *NULL* відображався б текст «відсутній». Для цього ми можемо створити два окремих запити, в одному з яких відібрати працівників з начальниками, а в іншому — без начальників з відповідними змінами у полі *chief*. Код такого запиту матиме вигляд, поданий в Лістингу 3.52.

```
(
    SELECT workers.*
    FROM workers
    WHERE workers.chief IS NOT NULL
) UNION (
    SELECT workers.passport_no,
           workers.last_name,
           workers.first_name,
           workers.middle_name,
           workers.qualification,
           "відсутній" AS chief
    FROM workers
    WHERE workers.chief IS NULL
);
```

Лістинг 3.52: Працівники

Як можна помітити в коді цього запиту, *об'єднання двох підзапитів-таблиць можна подавати без блоків SELECT та FROM.* Проте, якщо нам необхідно після цього впорядкувати працівників, наприклад, за їх категорією та номером паспорта, доведеться вкласти цей результат в блок *FROM* та дати йому псевдонім, після чого вже відповідним чином оформити блок *ORDER BY* (Лістинг 3.53).

```
SELECT all_workers.*
FROM (
    (
        SELECT workers.*
        FROM workers
        WHERE workers.chief IS NOT NULL
    ) UNION (
        SELECT workers.passport_no,
            workers.last_name,
            workers.first_name,
            workers.middle_name,
            workers.qualification,
            "відсутній" AS chief
        FROM workers
        WHERE workers.chief IS NULL
    )
) AS all_workers
ORDER BY all_workers.qualification,
    all_workers.passport_no;
```

Лістинг 3.53: Працівники

Результат виконання цього запиту зображено на Рисунку 3.58.

Якщо оператор *UNION* повертає об'єднання записів двох підзапитів без повторень, тобто запис, який трапляється в обох підзапитах, буде відображений в результаті лише один раз, то оператора *UNION ALL* дозволяє відобразити безпосереднє об'єднання підзапитів без прибирання повторів.

Як ми вже зазначали вище, табличні підзапити можна також використовувати під час побудови логічних виразів в якості параметра оператора *EXISTS*, що має наступний синтаксис:

EXISTS <підзапит-таблиця>

Змістовне наповнення цього оператора доволі просте: він повертає значення ІСТИНА (*TRUE*), якщо результат підзапиту містить принаймні один запис, і ХИБНІСТЬ (*FALSE*) — якщо результат підзапиту порожній.

passport_no	last_name	first_name	middle_name	qualification ▲ 1	chief
НН564127	Бобров	Семен	Іванович	1 кат.	АН564646
СР486823	Титаренко	Віктор	Леонідович	3 кат.	АА123456
АА123456	Іванова	Наталія	Сергіївна	вища	відсутній
АН564646	Андрієнко	Роман	Григорович	вища	АА123456
ВР556465	Боголюбов	Борис	Арсенович	вища	АН564646
ОН654643	Петренко	Віктор	Борисович	вища	СР486823

Рис. 3.58: Атомні електростанції

Наприклад, аби визначити виробників, які виготовляють товари ціною менше 20 гривень, достатньо виконати запит, код якого подано в Лістингу 3.54.

```
SELECT manufacturers.*
FROM manufacturers
WHERE EXISTS (
    SELECT goods.code
    FROM goods
    WHERE goods.manufacturer_code =
            manufacturers.code
        AND goods.price < 20
);
```

Лістинг 3.54: Виробники дешевих товарів

Як і деякі оператори, розглянуті нами вище, оператор *EXISTS* має двоїстий — *NOT EXISTS*, який повертає значення ІСТИНА (*TRUE*), якщо результат підзапиту порожній, і ХИБНІСТЬ (*FALSE*) — якщо результат підзапиту містить принаймні один запис.

Оператори *NOT IN* та *NOT EXISTS* є основою розв'язання задач на множинні порівняння.

Приклад 10

Відобразіть назви складів, що постачали товари принаймні в ті ж супермаркети, що й склад «Голосіївський».

З математичної точки зору нам необхідно визначити множину супер-

маркетів, в які постачались товари з кожного зі складів, зокрема й зі складу «Голосіївський». Після цього слід залишити ті склади, для яких множина супермаркетів складу «Голосіївський» є підмножиною їх множини супермаркетів. Проблема задач множинного порівняння полягає в тому, що SQL не надає можливості перевірки підмножин, а тому її доводиться реалізовувати доступними засобами самостійно. Переформулюємо факт «множина супермаркетів складу «Голосіївський» є підмножиною множини супермаркетів поточного складу» так: «не існує такого супермаркету, в який постачаються товари зі складу «Голосіївський», що в нього не постачаються товари з поточного складу». Таке формулювання дозволяє нам побудувати запит на основі розглянутих вище операторів (Лістинг 3.55).

```
SELECT warehouses.name
FROM warehouses
WHERE NOT EXISTS (
    SELECT supermarkets.code
    FROM (supermarkets INNER JOIN delivered
            ON supermarkets.code =
                delivered.supermarket_code)
        INNER JOIN warehouses AS W
            ON delivered.warehouse_code =
                W.code
    WHERE W.name = "Голосіївський" AND
        supermarkets.code NOT IN (
            SELECT delivered.supermarket_code
            FROM delivered
            WHERE delivered.warehouse_code =
                warehouses.code
        )
)
```

Лістинг 3.55: Множинне порівняння

Результат виконання даного запиту подано на Рисунку 3.59.

name
Голосіївський
Ужгородський

Рис. 3.59: Множинне порівняння

Вправа 114

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Відобразіть всі дані про атомні електростанції, в тому числі й їх діючість у вигляді тексту («діюча» або «недіюча»; назвіть поле *operationality*). Впорядкуйте електростанції за ідентифікатором.
2. Відобразіть назви атомних електростанцій, що містять тільки реактори того ж типу, що й Рівненська АЕС.
3. Відобразіть назви атомних електростанцій, що містять тільки реактори тих типів, які не містить Рівненська АЕС.

3.2.15 Додаткові вправи на вибірку даних

В цьому підрозділі подано додаткові вправи для відпрацювання навичок написання запитів на вибірку. Структуру таблиць відповідних баз даних ви можете переглянути в системі *sql:itolymp*.

Вправа 115

II етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас
За допомогою запитів до бази даних «Головоломки», написаних мовою SQL, виконайте наступні завдання:

1. Визначте ідентифікатори та назви неусних головоломок.
2. Визначте пари ідентифікаторів друкованих головоломок з однаковим типом. Розташуйте в парі спочатку головоломку з меншим ідентифікатором. Назвіть поля *printed1* та *printed2*.

Вправа 116

IV етап Інтернет-олімпіади з інформаційних технологій «IOIT-2017», 10-11 клас
За допомогою запитів до бази даних «Натуральні числа», написаних

мовою SQL, виконайте наступні завдання:

1. Визначте числа, що є одночасно і числами Фібоначчі, і числами Люка.
2. Визначте суму дільників числа 480. Назвіть поле *dividers*.
3. Для кожного з чисел обчисліть суму його дільників (назвіть поле *dividers*).
4. Визначте чи є число 496 досконалим, надлишковим або недостатнім. В результаті запиту зазначте єдине слово в полі *type* — тип числа.
   Досконале число — натуральне число, яке дорівнює сумі всіх своїх дільників крім самого числа. Наприклад, числа 6 та 28.
   Надлишкове число — натуральне число, менше за суму всіх своїх дільників крім самого числа. Наприклад, числа 12 та 48.
   Недостатнє число — натуральне число, більше за суму всіх своїх дільників крім самого числа. Наприклад, числа 11 та 22.
5. Визначте всі досконалі числа серед поданих.

Вправа 117

Тренувальний тур I етапу I Всеукраїнської учнівської Інтернет-олімпіади з інформаційних технологій
За допомогою запитів до бази даних «Школа», написаних мовою SQL, виконайте наступні завдання:

1. Визначте для кожного класу кількість предметів, які в ньому викладаються більш, ніж 2 години на тиждень.
2. Для кожного класу визначте середній бал по класу (назвіть поле *average_score*) та прізвище, ім'я та по батькові учня з найвищим середнім балом (одним полем, наприклад: *Іванов Іван Іванович*; назвіть поле *best_student*). Якщо таких учнів декілька – вивести першого за номером особової справи.
3. Визначте прізвище, ім'я та по батькові вчителів, які не є класними керівниками.
4. Вивести прізвища, імена та по батькові вчителів, які викладають алгебру на четвертому уроці або інформатику на п'ятому.

5. Вивести прізвища, імена та по батькові вчителів, які викладають алгебру на четвертому уроці і не викладають інформатику на п'ятому.
6. Визначте у яких класів є "вікна"та в який день і на якому уроці. Вікно — це вільний від заняття урок, після якого ще є навчальні заняття. Вважайте, що в розкладі не може бути підряд два вільних від заняття уроки посеред навчального дня.
7. Вивести прізвища, імена та по батькові вчителів, які викладають в усіх класах.
8. Вивести прізвища, імена та по батькові вчителів, які викладають лише предмети, які викладає *Леоненко Ксенія Ігорівна*.

Вправа 118

І етап Інтернет-олімпіади з інформаційних технологій «ІОІТ-2016», 10-11 клас

За допомогою запитів до бази даних «Файлова система», написаних мовою SQL, виконайте наступні завдання:

1. Визначте каталоги та файли кореневого каталогу (в таблиці він є своїм нащадком). Відобразіть назву відповідного об'єкту та його тип (*файл* або *каталог*; назвіть поле *type*). У списку спочатку розмістіть каталоги у алфавітному порядку, потім – файли у алфавітному порядку. Зверніть увагу, що сам каталог *root* до себе не належить, хоч так і вказано у відповідній таблиці.
2. Визначте назви документів Microsoft Word, збережених у файловій системі. Документи Microsoft Word зазвичай мають розширення *doc*, *docx*, *docm*.
3. Визначте всю інформацію про порожні каталоги.
4. Відобразіть назви файлів, що в дереві каталогів розташовані не більш, ніж на другому рівні вкладеності (тобто, файл може бути в кореневому каталозі або в каталозі, що є його нащадком).

Вправа 119

IV етап I Всеукраїнської учнівської олімпіади з інформаційних технологій
За допомогою запитів до бази даних «Водні ресурси», написаних мовою SQL, виконайте наступні завдання:

1. Визначте назви та довжини річок, які протікають територією України.
2. Визначте назви річок, на яких немає водосховищ.
3. Визначте назви річок, що протікають принаймні тими ж країнами, що й Десна.

Вправа 120

IV етап III Всеукраїнської учнівської олімпіади з інформаційних технологій
За допомогою запитів до бази даних «Географічна карта», написаних мовою SQL, виконайте наступні завдання:

1. Визначте міста, до яких можна дістатися з *Києва* по суші, перетнувши кордон не більше одного разу.
2. Визначити, які держави і моря межують з *Україною*.
3. Визначте країни, до яких можна дістатися з *України* по суші, перетнувши кордон рівно два рази, а також число *ways* — скількома способами (через скільки різних країн) це можна зробити.
4. Відобразіть нумерований перелік країн за їх площею (не включаючи до переліку *Росію*, *Азербайджан* та *Туреччину* як переважно азійські країни). Назвіть поле нумерації *N*, відобразіть також назву країни та її площу.
5. Визначте позицію міста Києва серед решти міст за населенням. Назвіть поле *position*.
6. Відобразіть три країни, що межують з найбільшою кількістю інших країн.

Вправа 121

IV етап IV Всеукраїнської учнівської олімпіади з інформаційних технологій

За допомогою запитів до бази даних «Лабіринти», написаних мовою SQL, виконайте наступні завдання:

1. Визначити координати клітинок, що розміщені на перетині тунелів. Кутові чи Т-подібні сполучення тунелів також вважаються перетинами. Назвіть стовпці *column_no* та *row_no*. Впорядкуйте клітинки за зростанням стовпця та рядка.
2. Визначити, скільки виходів із лабіринту видно з клітинки у четвертому стовпці та третьому рядку. Виходом називається клітинка тунелю, розташована на межі квадрата. Вважається, що вихід видно із клітинки, якщо вона належить тунелю, який має вихід. Вважайте, що тунель не може йти повністю по межі квадрата 10x10, тобто по його першому або останньому рядку чи стовпцю.
3. Відобразити всю інформацію, яка є в базі стосовно тунелів, що мають тупики. Тупиком називається кінець тунелю, який не є виходом і з якого не можна завернути в інший тунель. Для кожного тунелю додатково вкажіть його тип у полі *type*: *горизонтальний* або *вертикальний*.

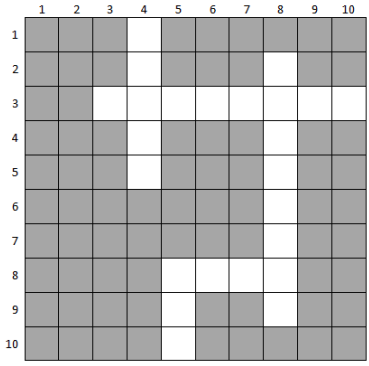


Рис. 3.60: До Вправи 121

3.3 Маніпуляція даними

Маніпуляція даними в таблицях передбачає всього три операції — додавання запису, його модифікацію та видалення. Розглянемо кожну з них окремо.

3.3.1 Додавання записів

Запит на додавання запису має наступний вигляд:

```
INSERT INTO <назва таблиця>
    [(<перелік стовпців через кому>)]
VALUES (<перелік відповідних значень через кому>);
```

Наприклад, аби додати товар *Булочка з маком* виробника *Київхліб* ціною 8,90 грн (штрих-код 182135600) в таблицю *goods* бази даних мережі супермаркетів, слід виконати запит, код якого подано в Лістингу 3.56.

```
INSERT INTO goods
    (code, name, price, manufacturer_code)
VALUES (182135600, "Булочка з маком", 8.90, 1);
```

Лістинг 3.56: Додавання нового товару

Перелік полів в цій команді вказувати не обов'язково, якщо вказувати значення полів відповідно до порядку полів таблиці в базі даних (Лістинг 3.57).

```
INSERT INTO goods
VALUES (182135600, "Булочка з маком", 8.90, 1);
```

Лістинг 3.57: Додавання нового товару

Окрім додавання рядків по одному, в SQL надається можливість також додавати в таблицю результат запиту на вибірку:

```
INSERT INTO <назва таблиця>
    (<перелік стовпців через кому>)
<запит на вибірку>;
```

Наприклад, якщо ми хочемо наповнити таблицю *cheap_goods* тієї ж структури, що й таблиця *goods*, товарами, дешевшими за 20 гривень, слід виконати запит, код якого подано в Лістингу 3.58.

```
INSERT INTO cheap_goods
    (code, name, price, manufacturer_code)
```

```
SELECT goods.*
FROM goods
WHERE goods.price < 20;
```

Лістинг 3.58: Додавання товарів з запиту

Зверніть увагу, що цей запит не створює таблицю, а лише наповнює вже створену таблицю даними.

3.3.2 Модифікація записів

Запит на додавання запису має наступний вигляд:

```
UPDATE <табличний вираз>
SET
      <назва поля> = <значення> [,
      <назва поля> = <значення> ...]
[WHERE <логічний вираз>];
```

В результаті виконання запиту в записах пов'язаних таблиць, для яких виконується логічний вираз, для полів, вказаних після ключового слова *SET*, встановлюються відповідні їм вказані значення. Якщо блок WHERE відсутній, значення встановлюються для всіх записів таблиці.

В якості значень тут можуть виступати константи, поточні значення полів або вирази над ними, підзапити-значення.

Наприклад, аби подвоїти ціни для товарів, ціна яких менша 100 гривень, достатньо виконати запит, код якого подано в Лістингу 3.59.

```
UPDATE goods
SET goods.price = goods.price * 2
WHERE goods.price < 100;
```

Лістинг 3.59: Оновлення цін товарів

Примітка. СУБД Access вимагає після інструкції UPDATE перелік полів, а тому до назв таблиць тут слід додати символи .*.

3.3.3 Видалення записів

Запит на видалення запису має наступний вигляд:

```
DELETE <перелік таблиць>
FROM <табличний вираз>
[WHERE <логічний вираз>];
```

В результаті виконання запиту будуть видалені записи з таблиць в переліку, для яких виконується логічний вираз. Якщо в переліку вказані не всі таблиці, присутні в табличному виразі, то видалення записів при виконанні умови здійснюється лише в тих таблицях, які вказані в переліку. Перелік можна не вказувати, якщо табличний вираз представлений однією таблицею.

Наприклад, аби видалити товари, ціна яких менша 100 гривень, достатньо виконати запит, код якого подано в Лістингу 3.60.

```
DELETE FROM goods
WHERE goods.price < 100;
```

Лістинг 3.60: Видалення товарів

Якщо потрібно видалити товари виробника *Київхліб*, ціна яких менша 100 гривень, достатньо виконати запит, код якого подано в Лістингу 3.61.

```
DELETE goods
FROM goods INNER JOIN manufacturers
    ON goods.manufacturer_code =
        manufacturers.code
WHERE goods.price < 100;
```

Лістинг 3.61: Видалення товарів

Вправа 122

За допомогою запитів до бази даних «Енергетика», написаних мовою SQL, виконайте наступні завдання:

1. Створіть таблицю *other_power_plants* з тією ж структурою даних, що й таблиця *power_plants* (за допомогою конструктора таблиць Access або phpMyAdmin). Додайте до неї електростанції, що не є ані атомними, ані гідроелектростанціями.
2. Додайте до таблиці *nuclear_power_plants* поле *potential_power* (за допомогою конструктора таблиць Access або phpMyAdmin). Внесіть в нього загальну потужність енергоблоків кожної з АЕС.
3. Видаліть з таблиці *power_plants* записи, що містяться в таблиці *other_power_plants*.

3.4 Маніпуляція структурою бази даних

Маніпуляція структурою бази даних включає в себе наступні основні можливості:

1. створення, редагування, видалення баз даних (*CREATE/ALTER/DROP DATABASE*);
2. створення, редагування структури, видалення таблиць (*CREATE/ALTER/DROP TABLE*);
3. очищення даних таблиць (*TRUNCATE TABLE*);
4. створення, редагування та видалення індексів (*CREATE/ALTER/DROP INDEX*).

Далі подано лише огляд основних операцій цієї частини мови SQL в якості короткої довідки. З рештою можливостей мови DDL ви можете познайомитися більш детально в [2].

3.4.1 Операції над базами даних

Створення баз даних в більшості реляційних систем управління базами даних здійснюється за допомогою наступного запиту:

CREATE DATABASE <назва бази даних>;

Для подальшого виконання запитів в межах створеної бази даних, слід виконати запит:

USE <назва бази даних>;

Після цього всі наступні запити стосуватимуться вказаної в якості параметра бази даних.

Видалення бази даних можна здійснити за допомогою запиту:

DROP DATABASE <назва бази даних>;

Зверніть увагу, що вказані тут команди не діятимуть в СУБД Microsoft Access, оскільки в ній кожна база даних представляється окремим файлом.

3.4.2 Операції над таблицями

Створення таблиць в більшості реляційних систем управління базами даних здійснюється за допомогою запиту наступного вигляду:

CREATE TABLE <назва таблиці> (
<назва поля> <тип поля> [<параметри поля>],

```
    <назва поля> <тип поля> [<параметри поля>],
    ...,
    [PRIMARY KEY <перелік полів ключа> ,]
    [FOREIGN KEY (<назва поля зовнішнього ключа>)
        REFERENCES <назва основної таблиці> (<назва поля>)]
);
```

Тут інструкція *PRIMARY KEY* визначає первинний ключ таблиці, а інструкція *FOREIGN KEY* — зовнішній ключ. Інструкція *PRIMARY KEY* може бути присутня не більше одного разу, а інструкція *FOREIGN KEY* — довільну кількість разів (відповідно до кількості зовнішніх ключів).

Наприклад, для створення таблиці товарів в СУБД MySQL слід виконати наступний запит (за умови наявності таблиці *manufacturers*):

```
CREATE TABLE goods (
    code int(11),
    name text,
    price double(11,2),
    manufacturer_code int(11),
    PRIMARY KEY (code),
    FOREIGN KEY (manufacturer_code)
        REFERENCES manufacturers (code)
);
```

В Microsoft Access аналогічний запит матиме трохи інший вигляд:

```
CREATE TABLE goods (
    code int,
    name text,
    price double,
    manufacturer_code int,
    PRIMARY KEY (code),
    FOREIGN KEY (manufacturer_code)
        REFERENCES manufacturers (code)
);
```

Видалення таблиці можна здійснити за допомогою запиту:

```
DROP TABLE <назва таблиці>;
```

Очищення даних таблиці можна здійснити за допомогою запиту:

```
TRUNCATE TABLE <назва таблиці>;
```

Слід зазначити, що ключове слово *TRUNCATE* в СУБД Microsoft Access недоступне.

3.4.3 Операції над індексами

Створення індексів в більшості реляційних систем управління базами даних здійснюється за допомогою запиту наступного вигляду:

```
CREATE [UNIQUE] INDEX <назва індексу>
    ON <назва таблиці> (
        <назва поля> [DESC] ,
        <назва поля> [DESC] ,
        ...
);
```

Наприклад, для створення унікального індексу з поля *code* таблиці *goods* з природнім порядком сортування в СУБД MySQL чи Microsoft Access слід виконати наступний запит:

```
CREATE UNIQUE INDEX goods_code
    ON goods (code);
```

Видалення індексу можна здійснити за допомогою запиту:

```
DROP INDEX <назва індексу>
    ON <назва таблиці>;
```

3.5 Відповіді та вказівки до розв'язання вправ

№77.

1. ```
   SELECT power_plants.name, power_plants.location
   FROM power_plants;
   ```
2. ```
   SELECT rivers.name, rivers.length
   FROM rivers;
   ```
3. ```
   SELECT power_units.number,
       power_units.reactor_type,
       power_units.power, power_units.NPP_id
   FROM power_units;
   ```

№78.

1. ```
   SELECT rivers.name,
       rivers.length/1000 AS length
   FROM rivers;
   ```
2. ```
   SELECT power_plants.name,
       power_plants.power*859.85 AS megacalories
   FROM power_plants;
   ```

№79.

1. ```
   SELECT power_plants.*
   FROM power_plants
   ORDER BY power_plants.power DESC;
   ```
2. ```
   SELECT rivers.name
   FROM rivers
   ORDER BY rivers.length;
   ```

№80.

1. ```
   SELECT power_units.NPP_id, power_units.number
   FROM power_units
   ORDER BY power_units.reactor_type DESC,
       power_units.number, power_units.NPP_id DESC;
   ```
2. ```
   SELECT nuclear_power_plants.id
   FROM nuclear_power_plants
   ORDER BY nuclear_power_plants.operationality,
       nuclear_power_plants.id DESC;
   ```
   ```

№81.

```
1. SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE power_plants.power > 1000;
2. SELECT placed_on.HPP_id
   FROM placed_on
   WHERE placed_on.river_name = "Дніпро";
3. SELECT nuclear_power_plants.id
   FROM nuclear_power_plants
   WHERE nuclear_power_plants.operationality = 1;
```

№82.

```
1. SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE power_plants.power >= 2000 AND
         power_plants.power <= 3000;
   SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE power_plants.power BETWEEN 2000 AND 3000;
2. SELECT placed_on.HPP_id
   FROM placed_on
   WHERE placed_on.river_name = "Дніпро" OR
         placed_on.river_name = "Дністер";
3. SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE (power_plants.power >= 2000 AND
         power_plants.power <= 3000) OR
         power_plants.location = "Розкопинці";
   SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE (power_plants.power BETWEEN 2000 AND 3000)
        OR power_plants.location = "Розкопинці";
```

№83. У зв'язку з присутністю значення НЕВИЗНАЧЕНІСТЬ (*NULL*) в трійковій логіці, ця властивість не буде тотожно істинною. Якщо X прийматиме значення *NULL*, порівняння також даватиме результат *NULL*.

З іншого боку, трійкова логіка теж має подібні тотожно істинні вирази:

```
X IS NULL OR X IS NOT NULL
X OR NOT X OR X IS NULL
```

Дійсно, якщо X прийматиме значення *NULL*, вираз X *IS NULL* буде істинним, в протилежному випадку — вираз X *OR NOT* X.

№84.

```
SELECT hydroelectric_power_plants.id
FROM hydroelectric_power_plants
WHERE hydroelectric_power_plants.type IS NULL;
```

№85.

1. *%у воді%*;
2. *%а%*;
3. *___%* або *%___*;
4. *%*.

№86.

1. ```
   SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE power_plants.name LIKE "%ГАЕС%"
   ORDER BY power_plants.name;
   ```
2. ```
   SELECT power_units.NPP_id, power_units.number,
       power_units.power
   FROM power_units
   WHERE power_units.reactor_type LIKE "%ВВЕР%"
   ORDER BY power_units.NPP_id, power_units.number;
   ```
3. ```
   SELECT power_plants.name
   FROM power_plants
   WHERE power_plants.power LIKE "%1" OR
       power_plants.power LIKE "%3" OR
       power_plants.power LIKE "%5" OR
       power_plants.power LIKE "%7" OR
       power_plants.power LIKE "%9";
   ```
4. ```
   SELECT power_plants.name
   FROM power_plants
   WHERE power_plants.location LIKE "%а%";
   ```
5. В цьому випадку обійтися лише оператором порівняння з шаблоном не вдасться, адже він не передбачає перевірку того, що який символ відсутній в рядку. Цей оператор передбачає тільки перевірку наявності символів.

З іншого боку, оператор *NOT LIKE*, який повертає ХИБНІСТЬ у разі невідповідності рядка шаблону, цілком здатен вирішити поставлене завдання. Адже відсутність літер «а» в слові — факт протилежний факту присутності принаймні однієї літери «а» в цьому слові.
Запит матиме наступний вигляд:

```
SELECT power_plants.name
FROM `power_plants`
WHERE power_plants.location NOT LIKE "%а%";
```

№87.

1. **у воді**;
2. *[1-9][0-9]–[1-9][0-9]*;
3. **[аеоуиіїєюя]**;
4. ***** або *****;
5. *[а-я]**;
6. **[!аеоуиіїєюя]**.

№88.

1. ```
   SELECT power_plants.name, power_plants.location
   FROM power_plants
   WHERE power_plants.name LIKE "*ГАЕС*"
   ORDER BY power_plants.name;
   ```
2. ```
   SELECT power_units.NPP_id, power_units.number,
          power_units.power
   FROM power_units
   WHERE power_units.reactor_type LIKE "*ВВЕР*"
   ORDER BY power_units.NPP_id, power_units.number;
   ```
3. ```
   SELECT power_plants.name
   FROM power_plants
   WHERE power_plants.power LIKE "*[13579]";
   ```
4. В цьому випадку обійтися лише оператором порівняння з шаблоном не вдасться, адже він не передбачає перевірку того, що який символ відсутній в рядку. Цей оператор передбачає тільки перевірку наявності символів.
   З іншого боку, оператор *NOT LIKE*, який повертає ХИБНІСТЬ у разі невідповідності рядка шаблону, цілком здатен вирішити поставлене завдання. Адже відсутність літер «а» в слові — факт протилежний факту присутності принаймні однієї літери «а» в цьому слові.
   ```

Запит матиме наступний вигляд:

```
SELECT power_plants.name
FROM `power_plants`
WHERE power_plants.location NOT LIKE "*a*";
```

№89.

```
SELECT *
FROM goods, manufacturers
WHERE goods.manufacturer_code = manufacturers.code;
```

№90. Твердження є наслідком теореми про потужність декартового добутку двох множин.

№91.

```
SELECT *
FROM nuclear_power_plants, power_plants;
```

№92.

```
SELECT placed_on.HPP_id AS HPP_1,
    placed_on_1.HPP_id AS HPP_2
FROM placed_on, placed_on AS placed_on_1
WHERE placed_on.river_name = placed_on_1.river_name AND
    placed_on.HPP_id < placed_on_1.HPP_id;
```

№93.

1. ```
   SELECT power_plants.name, power_plants.location,
       power_plants.power,
       nuclear_power_plants.operationality
   FROM power_plants, nuclear_power_plants
   WHERE power_plants.id = nuclear_power_plants.id;
   ```
2. ```
   SELECT power_plants.name AS HPP, placed_on.river_name
   FROM power_plants, placed_on
   WHERE power_plants.id = placed_on.HPP_id
   ORDER BY power_plants.name, placed_on.river_name;
   ```
3. ```
   SELECT power_plants.name
   FROM power_plants, hydroelectric_power_plants
   WHERE power_plants.id =
       hydroelectric_power_plants.id AND
       hydroelectric_power_plants.type =
           "гідроакумулююча"
   ORDER BY power_plants.name;
   ```
   ```

№94.

```
1. SELECT power_plants.name, power_plants.location,
       power_plants.power,
       nuclear_power_plants.operationality
   FROM power_plants INNER JOIN nuclear_power_plants
       ON power_plants.id = nuclear_power_plants.id;
2. SELECT power_plants.name AS HPP, placed_on.river_name
   FROM power_plants INNER JOIN placed_on
       ON power_plants.id = placed_on.HPP_id
   ORDER BY power_plants.name, placed_on.river_name;
3. SELECT power_plants.name
   FROM power_plants INNER JOIN
           hydroelectric_power_plants
       ON power_plants.id =
           hydroelectric_power_plants.id
   WHERE hydroelectric_power_plants.type =
           "гідроакумулююча"
   ORDER BY power_plants.name;
```

№95.

```
SELECT placed_on.HPP_id AS HPP_1,
    placed_on_1.HPP_id AS HPP_2
FROM placed_on INNER JOIN placed_on AS placed_on_1
    ON placed_on.river_name = placed_on_1.river_name
WHERE placed_on.HPP_id < placed_on_1.HPP_id;
```

№96.

```
SELECT power_units.*, power_plants.name,
    power_plants.location, power_plants.power
FROM (
        power_plants INNER JOIN nuclear_power_plants
        ON power_plants.id = nuclear_power_plants.id
    ) INNER JOIN power_units
    ON power_units.NPP_id = nuclear_power_plants.id
WHERE nuclear_power_plants.operationality = 1;
```

№97.

```
SELECT power_plants.name AS HPP_1,
    power_plants_1.name AS HPP_2
```

```
FROM (
        placed_on INNER JOIN power_plants
        ON placed_on.HPP_id = power_plants.id
    ) INNER JOIN (
        placed_on AS placed_on_1 INNER JOIN
            power_plants AS power_plants_1
        ON placed_on_1.HPP_id = power_plants_1.id
    ) ON placed_on.river_name = placed_on_1.river_name
WHERE placed_on.HPP_id < placed_on_1.HPP_id;
```

№98.

```
SELECT goods.*
FROM goods LEFT JOIN holds
    ON goods.code = holds.goods_code
WHERE holds.amount IS NULL;
```

№99.

1. ```
   SELECT power_plants.*,
       nuclear_power_plants.operationality,
       hydroelectric_power_plants.type
   FROM (
           power_plants LEFT JOIN
               hydroelectric_power_plants
           ON power_plants.id =
               hydroelectric_power_plants.id
       ) LEFT JOIN nuclear_power_plants
       ON power_plants.id = nuclear_power_plants.id;
   ```
2. ```
   SELECT power_plants.*
   FROM power_plants LEFT JOIN nuclear_power_plants
       ON power_plants.id = nuclear_power_plants.id
   WHERE nuclear_power_plants.id IS NULL;
   ```
3. ```
   SELECT power_plants.*
   FROM (
           power_plants LEFT JOIN
               hydroelectric_power_plants
           ON power_plants.id =
               hydroelectric_power_plants.id
       ) LEFT JOIN nuclear_power_plants
       ON power_plants.id = nuclear_power_plants.id
   WHERE nuclear_power_plants.id IS NULL AND
   ```
   ```

```
hydroelectric_power_plants.id IS NULL;
```

№100.

1.

river_name	HPP_id
Дніпро	1
	2
Дністер	3
Ріка	4
Теребля	4

2.

NPP_id	number	reactor_type	power
6	1	ВВЕР-1000	1000
	2	ВВЕР-1000	1000
7	3	ВВЕР-1000	1000
	4	ВВЕР-1000	1000
	1	ВВЕР-440	440
	2	ВВЕР-440	440
5	1	РБМК-1000	1000
	2	РБМК-1000	1000
	3	РБМК-1000	1000
	4	РБМК-1000	1000

3.

NPP_id	reactor_type	number	power
6	ВВЕР-1000	1	1000
		2	1000
7	ВВЕР-1000	3	1000
		4	1000
7	ВВЕР-440	1	440
		2	440
5	РБМК-1000	1	1000
		2	1000
		3	1000
		4	1000

№101.

1.
```
SELECT placed_on.river_name,
    COUNT(placed_on.HPP_id) AS plants_amount
FROM placed_on
GROUP BY placed_on.river_name;
```

2. Оскільки назва не є унікальним полем таблиці *power_plants*, групувати за цим полем не можна (ніхто не гарантує в межах бази даних, що назви станцій не співпадуть).

```
SELECT power_plants.name,
    SUM(power_units.power) AS total_power,
    AVG(power_units.power) AS average_power
FROM power_units INNER JOIN power_plants
    ON power_units.NPP_id = power_plants.id
```

```
GROUP BY power_plants.id;
```

3. В цьому випадку групування слід здійснювати одразу за двома полями: *power_plants.id* та *power_units.reactor_type*. Знов-таки, оскільки назва не є унікальним полем таблиці *power_plants*, групувати за цим полем не можна.

```
SELECT power_plants.name, power_units.reactor_type,
    SUM(power_units.power) AS total_power,
    AVG(power_units.power) AS average_power
FROM power_units INNER JOIN power_plants
    ON power_units.NPP_id = power_plants.id
GROUP BY power_plants.id, power_units.reactor_type;
```

4. Знов-таки, оскільки назва не є унікальним полем таблиці *power_plants*, групувати за цим полем не можна. Результат агрегатних функцій є звичайним значенням, а тому його можна використовувати в арифметичних виразах чи інших функціях.

```
SELECT power_plants.name,
    power_plants.power /
        SUM(power_units.power) AS ECE
FROM power_units INNER JOIN power_plants
    ON power_units.NPP_id = power_plants.id
GROUP BY power_plants.id
ORDER BY ECE DESC;
```

№102. Спочатку об'єднаємо таблицю товарів та їх зберігання та обчислимо суму для кожного запису (це добуток кількості з таблиці *holds* та ціни з таблиці *goods*). Після цього обчислюємо суму цього значення в усіх рядках.

```
SELECT SUM(holds.amount*goods.price)
    AS total_cost
FROM holds INNER JOIN goods
    ON holds.goods_code = goods.code;
```

№103.

1.
```
SELECT SUM(power_plants.power) AS total_power
FROM power_plants;
```

2. Частка — це кількість активних атомних електростанцій поділити на загальну кількість АЕС: *Active/Total*. Кількість активних електростанцій — це сума за полем діючості (адже воно рівне 1, якщо станція

активна, і 0 — в протилежному випадку). Отже, частка активних АЕС — це середнє арифметичне за полем діючості.

```
SELECT AVG(nuclear_power_plants.operationality)
    AS active_part
FROM nuclear_power_plants;
```

№104.

1. ```
   SELECT placed_on.river_name,
       COUNT(placed_on.HPP_id) AS plants_amount
   FROM placed_on INNER JOIN
           hydroelectric_power_plants
       ON placed_on.HPP_id =
           hydroelectric_power_plants.id
   WHERE hydroelectric_power_plants.type =
       "гідроакумулююча"
   GROUP BY placed_on.river_name;
   ```
2. ```
   SELECT power_plants.name,
       SUM(power_units.power) AS total_power,
       AVG(power_units.power) AS average_power
   FROM power_units INNER JOIN power_plants
       ON power_units.NPP_id = power_plants.id
   WHERE power_units.reactor_type LIKE "%РБМК%"
   GROUP BY power_plants.id;
   ```

3. Електростанції без таких енергоблоків — це електростанції, що не мають енергоблоків взагалі, або ж для них не виконується умова на тип реактора. Один із варіантів розв'язання (найбільш простий за структурою запиту) — це змінити в запиті до попереднього пункту умову об'єднання, додавши в неї умову на тип реактора. В такому разі об'єднуватимуться лише РБМК-енергоблоки та атомні елетростанції. Аби відобразити в цьому об'єднанні всі електростанції, а не тільки зв'язані, слід змінити внутнішнє об'єднання на праве зовнішнє.

```
SELECT power_plants.name,
    SUM(power_units.power) AS total_power,
    AVG(power_units.power) AS average_power
FROM power_units RIGHT JOIN power_plants
    ON power_units.NPP_id = power_plants.id
    AND power_units.reactor_type LIKE "%РБМК%"
GROUP BY power_plants.id;
```

№105.

1. ```
   SELECT power_plants.name
   FROM power_plants INNER JOIN placed_on
       ON power_plants.id = placed_on.HPP_id
   GROUP BY power_plants.id
   HAVING COUNT(placed_on.river_name) = 2;
   ```
2. ```
   SELECT power_plants.name
   FROM power_plants INNER JOIN power_units
       ON power_plants.id = power_units.NPP_id
   GROUP BY power_plants.id
   HAVING SUM(power_units.power) > 2000;
   ```
3. ```
   SELECT placed_on.river_name
   FROM placed_on INNER JOIN power_plants
       ON placed_on.HPP_id = power_plants.id
   GROUP BY placed_on.river_name
   HAVING AVG(power_plants.power) < 100;
   ```

№106.

1. ```
   SELECT AVG(nuclear_power_plants.operationality)
       AS active_part
   FROM nuclear_power_plants;
   ```
2. В цьому випадку групування слід здійснювати одразу за двома полями: *power_plants.id* та *power_units.reactor_type*. Оскільки назва не є унікальним полем таблиці *power_plants*, групувати за цим полем не можна.

   ```
   SELECT FIRST(power_plants.name) AS name,
       FIRST(power_units.reactor_type) AS reactor_type,
       SUM(power_units.power) AS total_power,
       AVG(power_units.power) AS average_power
   FROM power_units INNER JOIN power_plants
       ON power_units.NPP_id = power_plants.id
   GROUP BY power_plants.id, power_units.reactor_type;
   ```
3. ```
   SELECT FIRST(power_plants.name) AS name
   FROM power_plants INNER JOIN placed_on
       ON power_plants.id = placed_on.HPP_id
   GROUP BY power_plants.id
   HAVING COUNT(placed_on.river_name) = 2;
   ```
   ```

№107.

```
1. SELECT DISTINCT power_plants.name
   FROM power_plants INNER JOIN placed_on
        ON power_plants.id = placed_on.HPP_id
   WHERE placed_on.river_name = "Дністер"
        OR placed_on.river_name = "Ріка"
        OR placed_on.river_name = "Теребля"
   ORDER BY power_plants.name;
2. SELECT DISTINCT power_plants.name
   FROM power_plants INNER JOIN power_units
        ON power_plants.id = power_units.NPP_id
   WHERE power_units.reactor_type LIKE "%ВВЕР%"
   ORDER BY power_plants.name;
```

№108.

```
1. SELECT power_plants.*
   FROM power_plants
   ORDER BY power_plants.power DESC
   LIMIT 3;
2. SELECT placed_on.river_name
   FROM placed_on
   GROUP BY placed_on.river_name
   ORDER BY COUNT(placed_on.HPP_id) DESC
   LIMIT 1;
3. SELECT power_plants.name,
        power_plants.power /
            SUM(power_units.power) AS ECE
   FROM power_units INNER JOIN power_plants
        ON power_units.NPP_id = power_plants.id
   GROUP BY power_plants.id
   ORDER BY ECE
   LIMIT 1;
```

№109.

```
1. SELECT TOP 3 power_plants.*
   FROM power_plants
   ORDER BY power_plants.power DESC;
2. SELECT TOP 1 placed_on.river_name
```

```
FROM placed_on
GROUP BY placed_on.river_name
ORDER BY COUNT(placed_on.HPP_id) DESC;
```

3. Зверніть увагу, що Microsoft Access не підтримує впорядкування за синонімами, тому в *ORDER BY* тут слід вказувати сам вираз, а не наданий йому синонім.

```
SELECT TOP 30 PERCENT FIRST(power_plants.name) AS name,
    FIRST(power_plants.power) /
        SUM(power_units.power) AS ECE
FROM power_units INNER JOIN power_plants
    ON power_units.NPP_id = power_plants.id
GROUP BY power_plants.id
ORDER BY FIRST(power_plants.power) /
    SUM(power_units.power);
```

№110.

1. ```
   SELECT power_plants.*
   FROM power_plants
   WHERE power_plants.power > (
       SELECT AVG(power_plants.power)
       FROM power_plants
   );
   ```
2. ```
   SELECT power_plants.name, power_plants.power / (
       SELECT SUM(power_plants.power)
       FROM power_plants
   ) AS power_share
   FROM power_plants
   ORDER BY power_share DESC;
   ```

№111. Спершу слід побудувати всі можливі пари товарів — їх перехресне об'єднання (або декартів добуток). Далі слід залишити тільки ті пари, в яких ціна першого товару більша або рівна ціні другого. Після цього для кожного товару в першій компоненті такої пари слід обчислити кількість відповідних йому пар — це і буде номер у переліку.

Тут використовується саме знак $<=$, адже у разі строгого порівняння ми втратимо найдешевший товар (для нього жодного разу не виконуватиметься умова, адже не існує товару, дешевшого за нього). Аналогічно, номер товару у списку за зростанням ціни — це кількість товарів з меншою або рівною ціною.

```
SELECT COUNT(goods_1.code) AS position, goods.*
FROM goods, goods AS goods_1
WHERE goods_1.price <= goods.price
GROUP BY goods.code
ORDER BY position;
```

№112.

1.
```
SELECT (
        SELECT COUNT(PP.id)
        FROM power_plants AS PP
        WHERE PP.power < power_plants.power
    ) + 1 AS position, power_plants.*
FROM power_plants
ORDER BY position;
```

2.
```
SELECT power_plants.name,
    hydroelectric_power_plants.type,
    power_plants.power / (
        SELECT SUM(PP.power)
        FROM power_plants AS PP INNER JOIN
                hydroelectric_power_plants AS HPP
            ON PP.id = HPP.id
        WHERE HPP.type =
            hydroelectric_power_plants.type
    ) AS power_share
FROM power_plants INNER JOIN
        hydroelectric_power_plants
    ON power_plants.id =
        hydroelectric_power_plants.id;
```

3. Різниця між цією нумерацією та нумерацією в матеріалах розділу в тому, що умовою порядок енергоблоків визначено неоднозначно: невідомо, як повинні вони розташовуватися, якщо їх потужність однакова. Введемо такий порядок: якщо потужність енергоблоків однакова, то вони впорядковуються за кодом АЕС. Якщо і потужність, і код АЕС однакові — за номером енергоблоку.
   Тепер достатньо відповідним чином змінити умову підзапиту і нумерація повністю відповідатиме поставленій задачі.

```
SELECT (
        SELECT COUNT(PU.number)
        FROM power_units AS PU
```

```
            WHERE PU.power < power_units.power OR
                (PU.power = power_units.power AND
                    PU.NPP_id < power_units.NPP_id
                ) OR (PU.power = power_units.power
                    AND PU.NPP_id = power_units.NPP_id
                    AND PU.number < power_units.number
                )
        ) + 1 AS position, power_units.*
    FROM power_units
    ORDER BY position;
```

№113.

1. Оскільки при порожньому підзапиті модифікатор ALL завжди повертатиме значення ІСТИНА, тут слід додатково відфільтрувати електростанції, що не є атомними, за рахунок відповідного внутрішнього об'єднання.

```
SELECT power_plants.name
FROM power_plants INNER JOIN nuclear_power_plants
    ON power_plants.id = nuclear_power_plants.id
WHERE "ВВЕР-1000" = ALL (
        SELECT power_units.reactor_type
        FROM power_units
        WHERE power_units.NPP_id = power_plants.id
    );
```

2.
```
SELECT power_plants.name, power_plants.power
FROM power_plants
WHERE power_plants.power > ANY (
        SELECT power_plants.power
        FROM power_plants
    ) AND power_plants.power < ANY (
        SELECT power_plants.power
        FROM power_plants
    );
```

3.
```
SELECT power_plants.name
FROM power_plants
WHERE power_plants.id IN (
    SELECT placed_on.HPP_id
    FROM placed_on INNER JOIN rivers
        ON placed_on.river_name = rivers.name
    WHERE rivers.length < 100
```

```
);
```

4.
```
SELECT power_plants.name
FROM power_plants
WHERE power_plants.id NOT IN (
        SELECT nuclear_power_plants.id
        FROM nuclear_power_plants
    ) AND power_plants.id NOT IN (
        SELECT hydroelectric_power_plants.id
        FROM hydroelectric_power_plants
    );
```

5. Аби впевнитися, що всі енергоблоки електростанції містять реактори однакового типу, нам слід знайти принаймні один тип реактора для кожної електростанції (що можна зробити за допомогою групування та агрегатної функції MIN), а потім вказати, що всі типи реакторів для цієї електростанції повинні бути рівні цьому знайденому типові.

```
SELECT power_plants.name
FROM (power_plants INNER JOIN nuclear_power_plants
        ON power_plants.id =
            nuclear_power_plants.id)
    INNER JOIN power_units
        ON power_plants.id = power_units.NPP_id
GROUP BY power_plants.id
HAVING MIN(power_units.reactor_type) = ALL (
        SELECT PU.reactor_type
        FROM power_units AS PU
        WHERE PU.NPP_id = power_plants.id
    );
```

№114.

1. Для цього ми можемо створити два окремих запити, в одному з яких відібрати діючі АЕС і додати до них відповідний текст, а в іншому — виконати аналогічні дії для неактивних АЕС.

```
SELECT power_plants.*
FROM (
    (
        SELECT power_plants.*,
            "діюча" AS operationality
        FROM power_plants INNER JOIN
                nuclear_power_plants
```

```
            ON power_plants.id =
                nuclear_power_plants.id
        WHERE
          nuclear_power_plants.operationality = 1
    ) UNION (
        SELECT power_plants.*,
            "недіюча" AS operationality
        FROM power_plants INNER JOIN
                nuclear_power_plants
            ON power_plants.id =
                nuclear_power_plants.id
        WHERE
          nuclear_power_plants.operationality = 0
    )
) AS power_plants
ORDER BY power_plants.id;
```

2. Переформулюємо умову потрапляння електростанції в результат так: «не існує енергоблоків поточної електростанції з реакторами такого типу, що в Рівненській АЕС відсутні енергоблоки з реакторами такого типу».

```
SELECT power_plants.name
FROM power_plants INNER JOIN nuclear_power_plants
    ON power_plants.id = nuclear_power_plants.id
WHERE NOT EXISTS (
    SELECT power_units.number
    FROM power_units
    WHERE power_units.NPP_id = power_plants.id
        AND power_units.reactor_type NOT IN (
            SELECT power_units.reactor_type
            FROM power_units INNER JOIN
                    power_plants
                ON power_units.NPP_id =
                    power_plants.id
            WHERE power_plants.name =
                "Рівненська АЕС"
        )
);
```

3. Переформулюємо умову потрапляння електростанції в результат так: «не існує енергоблоків поточної електростанції з реакторами такого

типу, що в Рівненській АЕС є енергоблоки з реакторами такого типу».

```
SELECT power_plants.name
FROM power_plants INNER JOIN nuclear_power_plants
    ON power_plants.id = nuclear_power_plants.id
WHERE NOT EXISTS (
    SELECT power_units.number
    FROM power_units
    WHERE power_units.NPP_id = power_plants.id
        AND power_units.reactor_type IN (
            SELECT power_units.reactor_type
            FROM power_units INNER JOIN
                    power_plants
                ON power_units.NPP_id =
                    power_plants.id
            WHERE power_plants.name =
                "Рівненська АЕС"
        )
);
```

№115.

1. ```
   SELECT puzzles.id, puzzles.name
   FROM puzzles LEFT JOIN oral
       ON puzzles.id = oral.id
   WHERE oral.id IS NULL;
   ```
   або
   ```
   SELECT puzzles.id, puzzles.name
   FROM puzzles
   WHERE puzzles.id NOT IN (
       SELECT oral.id
       FROM oral
   );
   ```
2. ```
   SELECT printed.id AS printed1,
       printed_1.id AS printed2
   FROM printed, printed AS printed_1
   WHERE printed.type = printed_1.type
       AND printed.id < printed_1.id;
   ```

№116.

1. ```
   SELECT fibonacci_numbers.number
   ```
   ```

```
FROM fibonacci_numbers INNER JOIN lucas_numbers
    ON fibonacci_numbers.number =
        lucas_numbers.number;
```

2. Всі дільники числа 480 менші за 500, тому ми можемо вважати таблицю *natural_numbers* таблицею потенційних дільників цього числа. Для перевірки того, чи ділиться число 480 на числа таблиці, слід скористатися функцією *MOD* (в MySQL) або оператором *Mod* (в Access) — вони повертають остачу від ділення чисел.

   MySQL:
   ```
   SELECT SUM(natural_numbers.number) AS dividers
   FROM natural_numbers
   WHERE MOD(480, natural_numbers.number) = 0;
   ```
   Access:
   ```
   SELECT SUM(natural_numbers.number) AS dividers
   FROM natural_numbers
   WHERE 480 Mod natural_numbers.number = 0;
   ```

3. Всі дільники чисел до 500 менші за 500, тому ми можемо вважати таблицю *natural_numbers* таблицею потенційних дільників цього числа. Спочатку утворимо бінарне відношення «число є дільником числа» за допомогою перехресного об'єднання, а потім в цьому відношенні за допомогою групування за другим числом обчислимо суму дільників.

   MySQL:
   ```
   SELECT natural_numbers.number,
       SUM(dividers.number) AS dividers
   FROM natural_numbers AS dividers, natural_numbers
   WHERE MOD(natural_numbers.number,
       dividers.number) = 0
   GROUP BY natural_numbers.number;
   ```
   Access:
   ```
   SELECT natural_numbers.number,
       SUM(dividers.number) AS dividers
   FROM natural_numbers AS dividers, natural_numbers
   WHERE natural_numbers.number Mod dividers.number = 0
   GROUP BY natural_numbers.number;
   ```

4. Всі дільники числа 496 менші за 500, тому ми знов-таки можемо вважати таблицю *natural_numbers* таблицею потенційних дільників цього числа. Аби залежно від суми дільників відобразити певне слово, слід скористатися функцією *IF* (в MySQL) або *IIF* (в Access), що приймає

логічну умову та два вирази: той, який буде повернено в разі істинності умови, та той, який буде повернено в разі її хибності.

MySQL:

```
SELECT IF(
        (
            SELECT SUM(natural_numbers.number)
                AS dividers
            FROM natural_numbers
            WHERE MOD(496,
                natural_numbers.number) = 0
                AND natural_numbers.number <> 496
        ) = 496,
        "досконале",
        IF(
            (
                SELECT SUM(natural_numbers.number)
                    AS dividers
                FROM natural_numbers
                WHERE MOD(496,
                    natural_numbers.number) = 0
                    AND
                    natural_numbers.number <> 496
            ) > 496,
            "надлишкове",
            "недостатнє"
        )
    ) AS type;
```

Для Access слід замінити функцію *IF* на *IIF*, а функцію *MOD* — на відповідний оператор *Mod*.

5. У запиті з пункту 3 цієї Вправи слід внести корективи так, аби здійснювалося обчислення суми дільників, окрім самого числа. Тоді пошук досконалих чисел можна здійснити за допомогою додаткової післяумови на рівність числа і обчисленої суми.

MySQL:

```
SELECT natural_numbers.number,
    SUM(dividers.number) AS dividers
FROM natural_numbers AS dividers, natural_numbers
WHERE MOD(natural_numbers.number,
    dividers.number) = 0 AND
```

```
        natural_numbers.number <> dividers.number
GROUP BY natural_numbers.number
HAVING natural_numbers.number = SUM(dividers.number);
```

Access:

```
SELECT natural_numbers.number,
    SUM(dividers.number) AS dividers
FROM natural_numbers AS dividers, natural_numbers
WHERE natural_numbers.number Mod dividers.number = 0
    AND natural_numbers.number <> dividers.number
GROUP BY natural_numbers.number
HAVING natural_numbers.number = SUM(dividers.number);
```

№117.

1. Кожен рядок розкладу представляє собою один урок, тому достатньо підрахувати кількість рядків здійснюючи відповідне групування.

```
SELECT classes.name, schedule.subject_name
FROM classes INNER JOIN schedule
    ON classes.name = schedule.class_name
GROUP BY classes.name, schedule.subject_name
HAVING COUNT(schedule.lesson_no)>2;
```

2. Аби знайти учня з найвищим балом достатньо відсортувати їх за спаданням середнього балу та обмежити кількість результатів до одного. Аби при однакових кількостях балів не виникало неоднозначностей, слід додати додатковий ключ сортування — номер особової справи учня, як це сказано в умові. Аби відобразити в одному полі одразу три значення слід скористатися функцією *CONCAT* (в MySQL) або оператором конкатенації & (в Access).

```
SELECT students.class_name,
    AVG(students.average_score) AS average_score,
    (
        SELECT CONCAT(S.last_name, "␣",
            S.first_name, "␣", S.middle_name)
        FROM students AS S
        WHERE S.class_name = students.class_name
        ORDER BY S.average_score DESC,
            S.personal_file_number
        LIMIT 1
    ) AS best_student
FROM students
```

```
GROUP BY students.class_name;
```

Вираз для виводу в підзапиті в Access матиме наступний вигляд:

```
SELECT students.class_name,
    S.last_name & "␣" & S.first_name &
        "␣" & S.middle_name
```

3.
```
SELECT teachers.last_name, teachers.first_name,
    teachers.middle_name
FROM teachers
WHERE teachers.passport_no NOT IN (
    SELECT class_teachers.passport_no
    FROM class_teachers
);
```
або
```
SELECT teachers.last_name, teachers.first_name,
    teachers.middle_name
FROM teachers LEFT JOIN class_teachers
    ON teachers.passport_no =
        class_teachers.passport_no
WHERE class_teachers.passport_no IS NULL;
```

4.
```
SELECT DISTINCT teachers.last_name,
    teachers.first_name, teachers.middle_name
FROM teachers INNER JOIN schedule
    ON teachers.passport_no =
        schedule.teacher_passport_no
WHERE (schedule.subject_name = "Алгебра" AND
        schedule.lesson_no = 4) OR
    (schedule.subject_name = "Інформатика"
        AND schedule.lesson_no = 5);
```

5.
```
SELECT DISTINCT teachers.last_name,
    teachers.first_name, teachers.middle_name
FROM teachers INNER JOIN schedule
    ON teachers.passport_no =
        schedule.teacher_passport_no
WHERE (schedule.subject_name = "Алгебра" AND
        schedule.lesson_no = 4) AND
    teachers.passport_no NOT IN (
        SELECT schedule.teacher_passport_no
        FROM schedule
        WHERE schedule.subject_name =
```

```
            "Інформатика"
            AND schedule.lesson_no = 5
        )
```

6. Вікном є урок, попередній до уроку з навчальним заняттям, для якого не існує навчального заняття на попередньому уроці.

```
SELECT schedule.class_name, schedule.weekday,
    schedule.lesson_no - 1 AS lesson_no
FROM schedule
WHERE schedule.lesson_no > 1 AND NOT EXISTS (
        SELECT S.*
        FROM schedule AS S
        WHERE S.lesson_no = schedule.lesson_no - 1
        AND S.class_name = schedule.class_name
        AND S.weekday = schedule.weekday
    );
```

7. Перефразуємо умову так: «визначити вчителів, для яких не існує класу, де вони б не викладали».

```
SELECT teachers.last_name,
    teachers.first_name, teachers.middle_name
FROM teachers
WHERE NOT EXISTS (
        SELECT classes.name
        FROM classes
        WHERE classes.name NOT IN (
            SELECT schedule.class_name
            FROM schedule
            WHERE schedule.teacher_passport_no =
                teachers.passport_no
        )
    )
```

8. Перефразуємо умову так: «визначити вчителів, для яких не існує предмету, який вони б викладали, що його не викладає *Леоненко Ксенія Ігорівна*».

```
SELECT teachers.last_name,
    teachers.first_name, teachers.middle_name
FROM teachers
WHERE NOT EXISTS (
```

```
        SELECT schedule.subject_name
        FROM schedule
        WHERE schedule.teacher_passport_no =
                teachers.passport_no AND
            schedule.subject_name NOT IN (
                SELECT schedule.subject_name
                FROM schedule INNER JOIN
                        teachers
                    ON schedule.
                        teacher_passport_no =
                            teachers.passport_no
                WHERE teachers.last_name =
                    "Леоненко" AND
                    teachers.first_name =
                    "Ксенія" AND
                    teachers.middle_name =
                    "Ігорівна"
            )
    )
```

№118.

```
1. (
    SELECT folders.name, "каталог" AS type
    FROM folders INNER JOIN folders AS parent
        ON folders.parent_folder_id = parent.id
    WHERE parent.id = parent.parent_folder_id AND folders.id
        parent.id
    ORDER BY folders.name
  )
  UNION
  (
    SELECT files.name, "файл" AS type
    FROM files INNER JOIN folders
        ON files.folder_id = folders.id
    WHERE folders.id = folders.parent_folder_id
    ORDER BY files.name
  );
2. SELECT files.name
  FROM files
  WHERE (files.name LIKE "%.doc") OR
```

```
        (files.name LIKE "%.docx") OR
        (files.name LIKE "%.docm");
```

3.
```
SELECT *
FROM folders
WHERE folders.id NOT IN (
        SELECT files.folder_id
        FROM files
    ) AND folders.id NOT IN (
        SELECT folders.parent_folder_id
        FROM folders
    );
```

4. Оскільки в таблиці кореневий каталог є своїм же підкаталогом, то завдання значно спрощується.

```
SELECT files.name
FROM (folders AS L2 INNER JOIN folders
        ON L2.parent_folder_id = folders.id)
    INNER JOIN files
        ON files.folder_id = L2.id
WHERE folders.id = folders.parent_folder_id;
```

№119.

1.
```
SELECT rivers.name, rivers.length
FROM rivers INNER JOIN flows
    ON rivers.name = flows.river_name
WHERE flows.country = "Україна";
```

2.
```
SELECT rivers.name
FROM rivers LEFT JOIN reservoirs
    ON rivers.name = reservoirs.river_name
WHERE reservoirs.name IS NULL;
```

   або

```
SELECT rivers.name
FROM rivers
WHERE rivers.name NOT IN (
    SELECT reservoirs.river_name
    FROM reservoirs
);
```

3.
```
SELECT rivers.name
FROM rivers
WHERE NOT EXISTS (
```

```
    SELECT flows.country
    FROM flows
    WHERE flows.river_name = "Десна" AND
        flows.country NOT IN (
            SELECT flows.country
            FROM flows
            WHERE flows.river_name = rivers.name
        )
);
```

№120.

```
1. SELECT cities_to.name
   FROM (
            (cities INNER JOIN countries
                ON cities.country_name =
                    countries.name)
            INNER JOIN borders
                ON borders.region1 = countries.name
        ) INNER JOIN (
            cities AS cities_to INNER JOIN
                    countries AS countries_to
                ON cities_to.country_name =
                countries_to.name
        )
        ON borders.region2 = countries_to.name
   WHERE cities.name = "Київ";
2. SELECT regions.name
   FROM (
            countries INNER JOIN borders
                ON countries.name = borders.region1
        ) INNER JOIN regions
            ON borders.region2 = regions.name
   WHERE countries.name = "Україна";
3. SELECT countries_to.name,
        COUNT(countries_mid.name) AS ways
   FROM (
            (
                (
                    countries INNER JOIN borders
                        ON borders.region1 =
```

```
                countries.name
            ) INNER JOIN countries
                AS countries_mid
            ON borders.region2 =
                countries_mid.name
        ) INNER JOIN borders
            AS borders_to
        ON countries_mid.name = borders_to.region1
    ) INNER JOIN countries
        AS countries_to
    ON borders_to.region2 = countries_to.name
WHERE countries.name = "Україна" AND
    countries_to.name <> "Україна"
GROUP BY countries_to.name;
```

4.
```
SELECT (
        SELECT COUNT(countries.name)
    FROM countries INNER JOIN regions AS regions1
        ON countries.name = regions1.name
    WHERE countries.name NOT IN
        ("Росія", "Азербайджан", "Туреччина")
        AND regions1.area >= regions.area
) AS N, regions.*
FROM countries INNER JOIN regions
    ON countries.name = regions.name
WHERE countries.name NOT IN
    ("Росія", "Азербайджан", "Туреччина")
ORDER BY N;
```

Тут використано оператор *NOT IN* для синтаксичного спрощення наступного виразу:

```
countries.name <> "Росія" AND
    countries.name <> "Азербайджан" AND
    countries.name <> "Туреччина"
```

5.
```
SELECT COUNT(cities.name) AS position
FROM cities, cities AS kyiv
WHERE kyiv.name = "Київ" AND
    kyiv.population <= cities.population;
```

6. MySQL:
```
SELECT borders.region1,
    COUNT(borders.region2) AS borders
```

```
FROM (borders INNER JOIN countries
        ON borders.region1 = countries.name)
    INNER JOIN countries AS c
        ON borders.region2 = c.name
GROUP BY borders.region1
ORDER BY `borders` DESC
LIMIT 3;
```

Access:

```
SELECT TOP 3 borders.region1,
    COUNT(borders.region2) AS borders
FROM (borders INNER JOIN countries
        ON borders.region1 = countries.name)
    INNER JOIN countries AS c
        ON borders.region2 = c.name
GROUP BY borders.region1
ORDER BY `borders` DESC;
```

№121.

1. Вважатимемо, що таблиці з'єднані, якщо тунелі перетинаються. Вертикальний тунель перетинається з горизонтальним, якщо його стовпець розташований між стовпцями горизонтального, а рядок горизонтального — між його рядками. Далі утворюємо відповідне об'єднання, а координатами перетину визначаємо стовпець вертикального та рядок горизонтального тунелю.

```
SELECT vertical.column_no, horizontal.row_no
FROM horizontal INNER JOIN vertical
    ON (vertical.column_no
            BETWEEN horizontal.start_column_no
            AND horizontal.end_column_no
        ) AND (horizontal.row_no
            BETWEEN vertical.start_row_no
            AND vertical.end_row_no
        )
ORDER BY vertical.column_no,
    horizontal.row_no;
```

2. Комірка належить вертикальному тунелю, якщо її стовпець рівний стовпцю тунелю, а рядок розташований між рядками тунелю. Аналогічно, комірка належить горизонтальному тунелю, якщо її рядок рівний рядку тунелю, а стовпець розташований між стовпцями тунелю.

Виконуючи попередні умови, нам потрібно додати чотири числа:

- кількість горизонтальних тунелів, що містять комірку і починаються в першому стовпці;
- кількість горизонтальних тунелів, що містять комірку і закінчуються в останньому стовпці;
- кількість вертикальних тунелів, що містять комірку і починаються в першому рядку;
- кількість вертикальних тунелів, що містять комірку і закінчуються в останньому рядку.

Дійсно, якщо горизонтальний тунель тягнеться через увесь рядок, то він додає одразу два виходи для клітинок, які містить.

```
SELECT (
    SELECT COUNT(horizontal.row_no)
    FROM horizontal
    WHERE horizontal.start_column_no = 1
        AND horizontal.row_no = 3
        AND 4 BETWEEN
            horizontal.start_column_no
            AND horizontal.end_column_no
) + (
    SELECT COUNT(horizontal.row_no)
    FROM horizontal
    WHERE horizontal.end_column_no = 10
        AND horizontal.row_no = 3
        AND 4 BETWEEN
            horizontal.start_column_no
            AND horizontal.end_column_no
) + (
    SELECT COUNT(vertical.column_no)
    FROM vertical
    WHERE vertical.start_row_no = 1
        AND vertical.column_no = 4
        AND 3 BETWEEN
            vertical.start_row_no
            AND vertical.end_row_no
) + (
    SELECT COUNT(vertical.column_no)
    FROM vertical
```

```
    WHERE vertical.end_row_no = 10
    AND vertical.column_no = 4
    AND 3 BETWEEN
        vertical.start_row_no
        AND vertical.end_row_no
) AS val
```

3. Вертикальний тунель містить тупик, якщо він починається не з першого рядка і не існує горизонтального тунелю, що проходить через його початок, або завершуються не в останньому рядку і не існує горизонтального тунелю, що проходить через його кінець. Горизонтальний тунель містить тупик, якщо він починається не з першого стовпця і не існує вертикального тунелю, що проходить через його початок, або завершуються не в останньому стовпці і не існує вертикального тунелю, що проходить через його кінець.

```
(
    SELECT horizontal.*, "горизонтальний" AS type
    FROM horizontal
    WHERE (
        horizontal.start_column_no > 1 AND
        NOT EXISTS (
            SELECT vertical.*
            FROM vertical
            WHERE vertical.column_no =
                horizontal.start_column_no AND (
                    horizontal.row_no
                    BETWEEN vertical.start_row_no
                    AND vertical.end_row_no
                )
        )
    ) OR (
        horizontal.end_column_no < 10 AND
        NOT EXISTS (
            SELECT vertical.*
            FROM vertical
            WHERE vertical.column_no =
                horizontal.end_column_no AND (
                    horizontal.row_no
                    BETWEEN vertical.start_row_no
                    AND vertical.end_row_no
```

```
                    )
                )
            )
        ) UNION (
            SELECT vertical.*, "вертикальний" AS type
            FROM vertical
            WHERE (
                vertical.start_row_no > 1 AND
                NOT EXISTS (
                    SELECT horizontal.*
                    FROM horizontal
                    WHERE horizontal.row_no =
                        vertical.start_row_no AND (
                            vertical.column_no
                        BETWEEN
                            horizontal.start_column_no
                        AND horizontal.end_column_no
                    )
                )
            ) OR (
                vertical.end_row_no < 10 AND
                NOT EXISTS (
                    SELECT horizontal.*
                    FROM horizontal
                    WHERE horizontal.row_no =
                        vertical.end_row_no AND (
                            vertical.column_no
                        BETWEEN
                            horizontal.start_column_no
                        AND horizontal.end_column_no
                    )
                )
            )
        )
```

№122.

1.
```
INSERT INTO other_power_plants
SELECT power_plants.*
FROM power_plants
```

```
    WHERE power_plants.id NOT IN (
        SELECT nuclear_power_plants.id
        FROM nuclear_power_plants
    ) AND power_plants.id NOT IN (
        SELECT hydroelectric_power_plants.id
        FROM hydroelectric_power_plants
    );
```

2.
```
    UPDATE nuclear_power_plants
    SET nuclear_power_plants.potential_power = (
        SELECT SUM(power_units.power)
        FROM power_units
        WHERE power_units.NPP_id =
            nuclear_power_plants.id
    );
```

3.
```
    DELETE power_plants
    FROM power_plants INNER JOIN other_power_plants
        ON power_plants.id = other_power_plants.id;
```

Частина II

Інформаційні системи

Розділ 4

Інформаційні системи в Microsoft Access

Ми ґрунтовно розглянули принципи проектування баз даних, їх реалізації та роботи з ними, в тому числі опрацювали різні можливості, які надає мова SQL для написання запитів на вибірку, додавання, модифікацію та видалення даних.

Бази даних є основною більшості програмних комплексів, які об'єднуються спільною назвою: інформаційні системи. Ця частина книги буде присвячена розгляду фундаментальних принципів їх побудови на основі можливостей програмного засобу Microsoft Access, а також основні поради щодо створення промислових програмних продуктів засобами популярних мов програмування.

Microsoft Access рідко використовується для побудови промислових інформаційних систем через низьку гнучкість та слабкий рівень безпеки. Проте він дуже зручний для вивчення базових принципів побудови таких систем, оскільки не потребує знання будь-яких мов програмування (окрім, звісно, мови запитів SQL), а тому має доволі низький рівень входження.

4.1 Форми та автоматизовані засоби їх створення

Форми — один із п'яти класів об'єктів системи Microsoft Access, поряд з таблицями, запитами, звітами та макросами. На відміну від розглянутих нами раніше таблиць та запитів, форми є частиною не власне бази даних,

а вже інформаційної системи, побудованої на її основі. Іншими словами, форми — перший з розглянутих нами об'єктів цієї СУБД, який відповідає за інтерфейс користувача.

Залежно від обраного режиму відображення, форма — це окреме вікно або окрема вкладка в інтерфейсі системи Access, що споріднює інформаційні системи, створені в Microsoft Access, зі звичайними настільними додатками.

Як таблиці і запити, так і форми можна створювати за допомогою інструментів окремого блоку на вкладці *Create (Створити)*, проте для створення форм система надає значно більше різних варіантів (Рисунок 4.1).

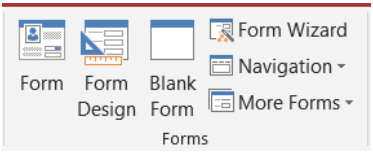


Рис. 4.1: Інструменти для створення форм

4.1.1 Автоформи

Інструмент *Form (Форма)* дозволяє створити форму автоматично на основі таблиці чи запиту, обраного в навігаційній панелі.

Створена таким чином форма найпростіша: вона дозволяє перегляд записів таблиці по одному та додавання нових записів.

Приклад такої форми для таблиці *товари* подано на Рисунку 4.2.

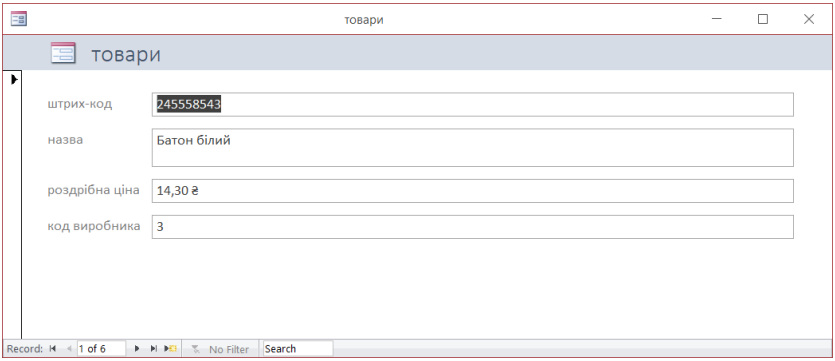


Рис. 4.2: Автоформа *товари*

4.1.2 Майстер форм

Інструмент *Form Wizard (Майстер форм)* дозволяє створити форму автоматично на основі кількох пов'язаних таблиць та/або запитів.

За допомогою майстра (Рисунок 4.3) є можливість створити форму, що представлятиме дані кількох об'єктів бази даних одночасно. Для цього необхідно за допомогою відповідних кнопок додати поля кожної з потрібних таблиць у обрані та продовжити роботу з інструментом.

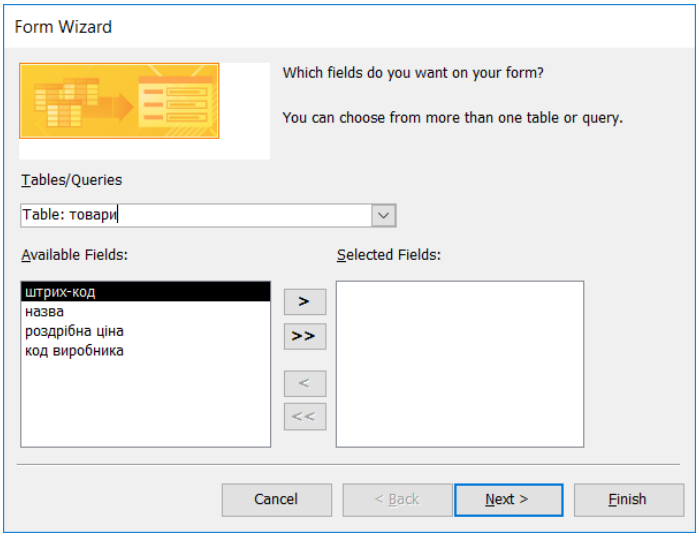


Рис. 4.3: Майстер форм

Нехай необхідно створити форму яка дозволяла б переглядати (з можливістю редагування) дані про виробників та товарів, які кожен з виробників виготовляє. Оскільки ці таблиці з'єднані зв'язком «один до багатьох», після вибору полів для форми майстер запропонує обрати один з двох варіантів подання: перегляд даних за виробником або за товаром. Найбільш цікавим є перший з варіантів (тобто перегляд даних за таблицею з поміткою «один»), адже система автоматично пропонує створити підформу з пов'язаними товарами (записами таблиці з поміткою «багато»), як це показано на Рисунку 4.4.

Одержана за допомогою цього інструменту форма матиме вигляд, поданий на Рисунку 4.5, та дозволятиме переглядати і редагувати не тільки виробників, а й пов'язані з ними товари, подані у вигляді окремої таблиці.

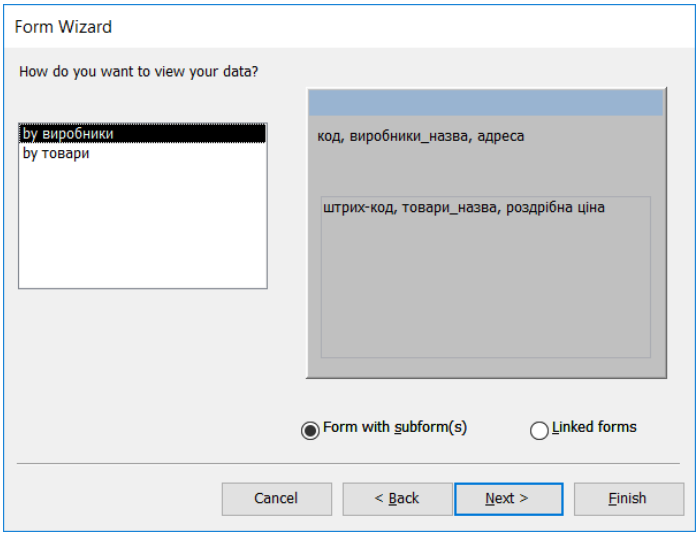


Рис. 4.4: Створення форми з підформою

Слід зазначити, що у разі переходу між записами дані цієї таблиці змінюються автоматично відповідно до поточного товару.

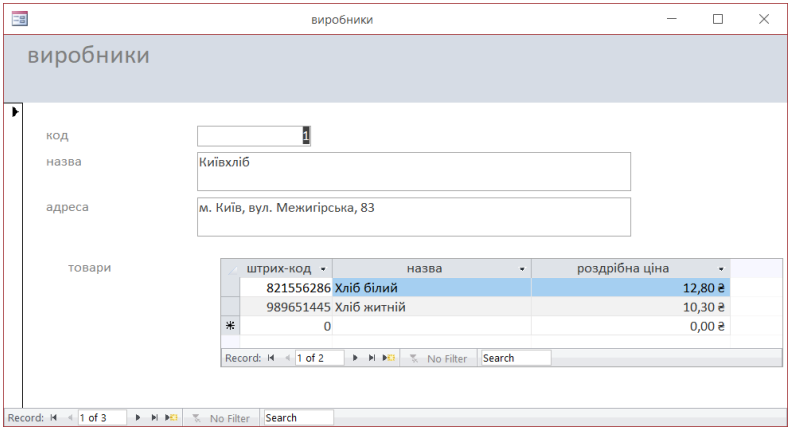


Рис. 4.5: Форма *виробники* з підформою товарів

Якщо додавати в майстрі поля до форми з таблиць, що складають лан-

цюжок зв'язків «один до багатьох» (наприклад, таблиці *виробники*, *товари* та *зберігається*), майстер автоматично додаватиме не одну, а кілька підформ, кожна наступна з яких залежатиме від попередньої. Тобто, в даному випадку, основна форма відображатиме дані про виробників, перша з підформ — про товари цих виробників, а друга — про те, де зберігається виділений в першій підформі товар (Рисунок 4.6).

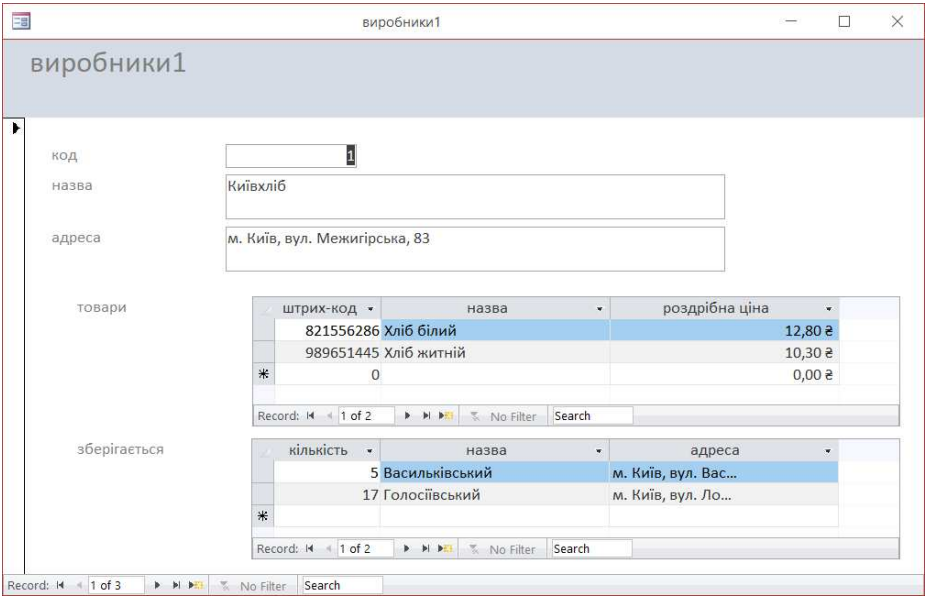


Рис. 4.6: Форма *виробники* з підформами товарів та складів

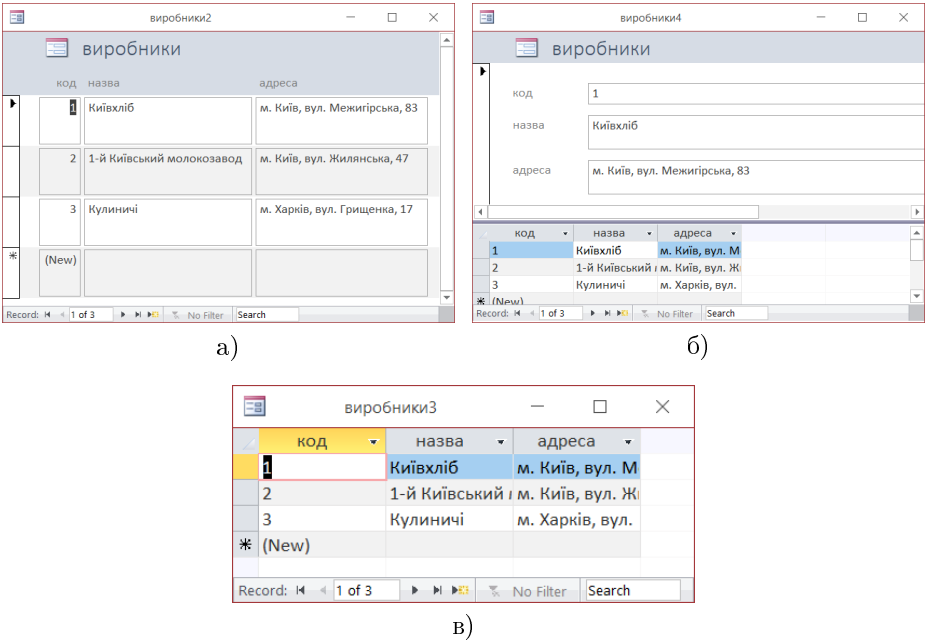
4.1.3 Створення форм особливого вигляду

Microsoft Access надає також можливість створення автоформ особливого вигляду, як-от форм на багато елементів (*Multiple Items*), табличних форм (*Datasheet*) та розділеної форми (*Split Form*). Створення таких форм можливе за допомогою вказаних в дужках інструментів в розкривному списку *More Forms*.

Форма на декілька елементів (Рисунок 4.7а) дозволяє певним чином надати кращого вигляду звичайним таблицям Microsoft Access, проте зберегти табличний стиль подання даних.

Таблична форма (Рисунок 4.7в) повністю повторює вигляд табличного подання запиту чи таблиці.

Розділена форма (Рисунок 4.7б) поєднує в собі звичайну автоформу та табличну форму, розділені між собою пересувною межею.



а) б)

в)

Рис. 4.7: Форми: (а) на декілька елементів, (б) розділена та (в) таблична

4.2 Створення форм засобами конструктора

Microsoft Access надає можливість працювати з формою в трьох поданнях: подання форми (*Form View*), подання макету (*Layout View*) та подання конструктора (*Design View*).

Якщо подання форми призначене для роботи з даними, які ця форма відображає, то решта з них дозволяють змінювати в тій чи іншій мірі вміст форми на рівні її структури.

Оскільки подання конструктора є значно ширшим і включає практично всі інструменти, представлені в поданні макету, ми зупинимося на детальному розгляді саме цього подання, а подання макету пропонується читачеві розглянути самостійно.

Далі розглядатимуться засоби створення готової форми з порожньої (інструмент *Blank Form (Пуста форма)* або *Form Design (Конструктор форм)*), але вони ж можуть бути використані й для редагування форм, створених розглянутими вище автоматичними засобами.

4.2.1 Структура форми в режимі конструктора

Загалом форма в режимі конструктора складається з п'яти елементів: верхнього колонтитула (заголовка) форми (*Form Header*), верхнього колонтитула сторінки (*Page Header*), тіла (*Detail*), нижнього колонтитула (підвалу) сторінки (*Page Footer*) та нижнього колонтитула форми (*Form Footer*).

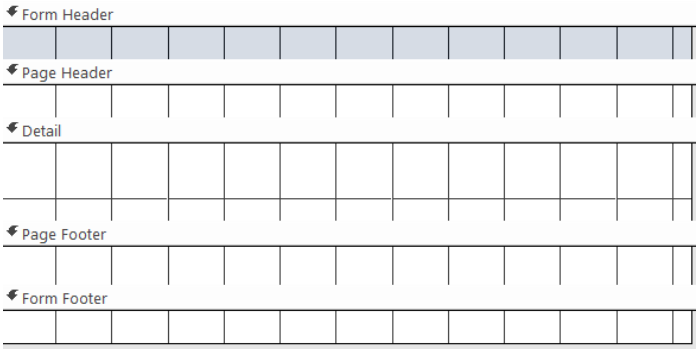


Рис. 4.8: Структура форми в режимі конструктора

Оскільки форми зазвичай призначені для перегляду та маніпуляції даними, їх зазвичай не виводять на друк, проте така можливість все-таки передбачена. При виводі на друк на першій сторінці вгорі друкуватиметься верхній колонтитул форми, після нього на першій сторінці та вгорі кожної наступної сторінки друкуватиметься верхній колонтитул сторінки, внизу кожної сторінки — нижній колонтитул сторінки, а внизу останньої сторінки, після колонтитула сторінки — нижній колонтитул форми.

Якщо друк форми не планується, колонтитули сторінки жодним чином не впливають на форму і можуть бути опущені.

Як і звичайному текстовому документі, різниця між тілом форми та її колонтитулами в тому, що тіло містить переважно змістовну інформацію, отриману з бази даних, тоді як колонтитули — переважно технічну. З іншого боку, це не виключає наявність змістовних даних в колонтитулах та технічних в тілі форми — цей поділ доволі умовний, проте в деяких типах форм (наприклад, у формі на декілька елементів) представлення тіла

форми може значно відрізнятися від представлення її колонтитулів.

Слід зазначити, що тіло форми, подане в режимі конструктора, являє собою певний макет того, які дані відображатимуться *для одного запису форми* і яким чином вони відображатимуться. Тому в межах режиму конструктора ми працюватимемо не з усім результатом певного запиту, а лише з одним його записом на рівні назв відповідних полів.

За промовчанням у порожній формі всі колонтитули вимкнені. Ви можете відобразити їх за допомогою опцій *Page Header/Footer* та *Form Header/Footer* контекстного меню тіла форми (зафарбоване зазвичай білим кольором).

4.2.2 Параметри даних форми. Класифікація форм за вмістом

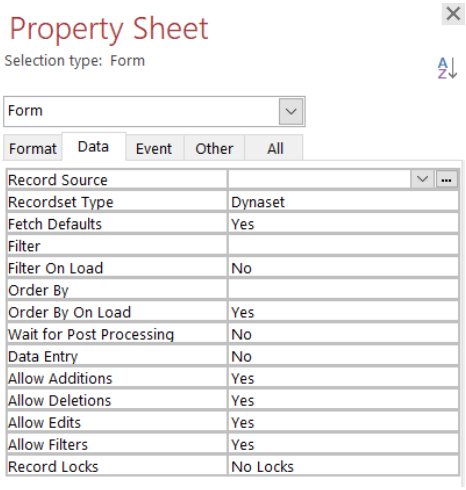


Рис. 4.9: Аркуш властивостей форми

Будь-яка форма має набір параметрів, що пов'язані з даними, які відображаються на формі. Ці параметри в будь-який час можна змінити на вкладці *Data (Дані)* аркуша властивостей (Рисунок 4.9), який можна відобразити за допомогою інструмента *Property Sheet (Аркуш властивостей)* вкладки *Desing (Конструктор)* головного меню.

Слід зазначити, що аркуш властивостей доступний і для інших об'єктів форми, а тому важливо попередньо переконатися, що в розкривному списку

в верхній частині панелі обрано саме пункт *Form (Форма)*.

Одним із найбільш важливих параметрів форми, розміщених тут, є *Record Source (Джерело записів)*. Залежно від змісту форми (і, відповідно, значення цього параметра), розрізняють зв'язані та вільні форми.

Вільні форми мають порожнє джерело записів і зазвичай відображають статичні дані. Прикладом такої форми, є наприклад головна форма, яка містить лише кнопки для переходу до інших форм і не містить записів, отриманих в результаті певного запиту. З цієї точки зору побудова вільних форм є значно легшим процесом, аніж побудова зв'язаних.

Зв'язані ж форми будуються на основі певного запиту — джерела записів, який вказується у вигляді SQL-коду як значення відповідного параметра. Після того, як для форми вказано джерело записів, в режимі конструктора стають доступними поля цього джерела, тобто поля відповідного запиту до бази даних.

Microsoft Access пропонує деякі спрощені варіанти джерел рядків. Наприклад, в якості джерела можна вказати таблицю або збережений запит (власне так і відбувається в автоформах), або ж побудувати окремий запит, натиснувши на кнопку з трикрапкою. Вона відкриває новий запит в режимі конструктора, проте ви з легкістю можете перейти в режим SQL за допомогою інструмента в правому нижньому куті вікна.

Після встановлення джерела записів, у відповідному полі аркуша властивостей форми відображатиметься або назва таблиці чи збереженого запиту, або ж код вбудованого запиту (Рисунок 4.10).

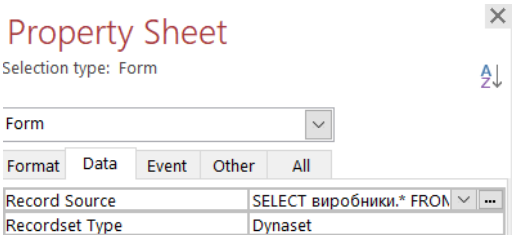


Рис. 4.10: Джерело записів форми

Окрім джерела записів на аркуші властивостей ви можете налаштувати й інші параметри форми, як-от можливість додавання, видалення, редагування записів та їх фільтрування в поданні форми. На деяких з них ми детальніше зупинимося трохи згодом, решту пропонується читачеві дослідити самостійно.

4.2.3 Елементи керування формою

Microsoft Access надає широкий вибір елементів керування формою, за допомогою яких можна в різному форматі представляти дані та маніпулювати ними і інформаційною системою в цілому. Додати елемент керування можна за допомогою інструменту **Controls (Елементи керування)** вкладки **Desing (Конструктор)** головного меню (Рисунок 4.11).

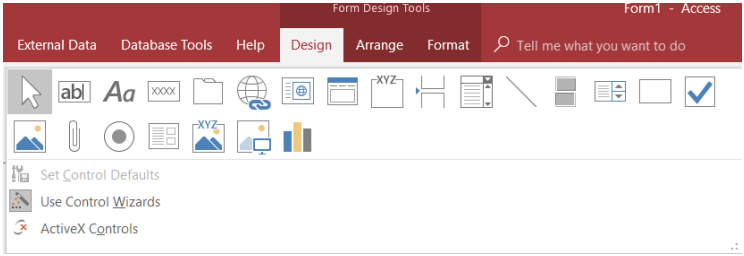


Рис. 4.11: Елементи керування формою

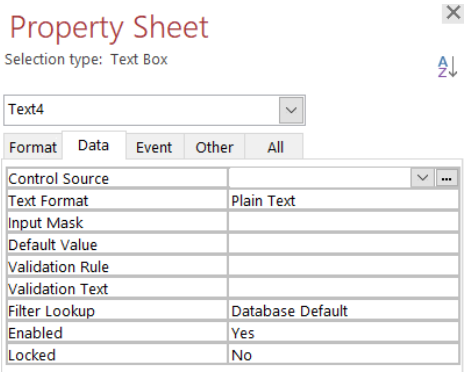


Рис. 4.12: Аркуш властивостей текстового поля

Більшість популярних елементів керування (як-от текстове поле, розкривний список, список, прапорець тощо), як і форми, можуть бути вільними або зв'язаними. Для них на аркуші властивостей (Рисунок 4.12) є можливість встановити параметр *Control Source (Джерело керування)* — поле джерела записів або вираз, що може містити його поля, з якого буде отримуватися значення для відображення у відповідному елементі керування, а також в яке будуть в деяких випадках вноситись зміни, що були внесені в

дані елемента керування.

Аби внести ясність в процес побудови форм, розглянемо наступний приклад.

Приклад 11

Створіть в режимі конструктора форму для перегляду виробників.

Отже, розпочнемо створення з порожньої форми. Оскільки форма повинна відображати дані про виробників, її даними будуть усі дані про виробників, які ми можемо отримати за допомогою запиту з Лістингу 4.1.

```
SELECT виробники.*
FROM виробники;
```

Лістинг 4.1: Виробники

Цей запит є джерелом записів нашої форми.

Кожен виробник характеризується трьома параметрами, тому додамо на форму три поля, відповідним чином змінюючи їх підписи (Рисунок 4.13).

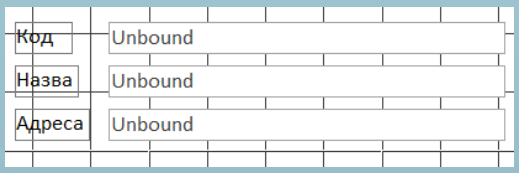


Рис. 4.13: Макет форми в режимі конструктора

Всі три поля нашої форми будуть зв'язаними, адже повинні містити дані з джерела записів. Тому на аркуші властивостей встановлюємо в якості значення параметру *Control Source (Джерело керування)* назву відповідного поля з джерела записів (Рисунок 4.14).

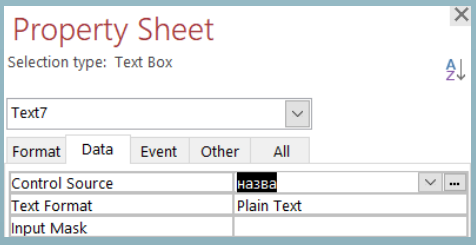


Рис. 4.14: Встановлення джерела керування для назви виробника

Цю дію можна виконати і на макеті форми, просто редагуючи тут вміст відповідних елементів керування (Рисунок 4.15).

Рис. 4.15: Макет форми в режимі конструктора

Власне на цьому етап створення форми завершено. Якщо перейти тепер в подання форми, то можна пересвідчитися, що у формі дійсно відображаються дані про виробників, між якими можна переходити за допомогою інструментів панелі навігації в нижній частині вікна (Рисунок 4.16).

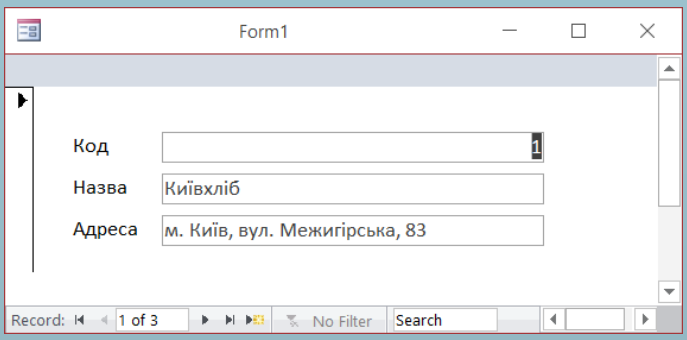


Рис. 4.16: Форма виробників

За потреби, дизайн форми можна доповнити написами, а також змінити кольори шрифтів, прибрати межі текстових полів — всі ці параметри присутні на аркуші властивостей відповідних об'єктів.
Слід зазначити, що в цій формі можна зокрема й редагувати дані, а також додавати нові записи. Чому це можливо, ми розглянемо згодом.

Розкривні списки

Розкривний список (Combo Box), як і текстове поле, може бути зв'язаним елементом керування, але на відміну від нього надає можливість не вводити значення, а обрати його з переліку. При його додаванні Access відкриває майстер розкривних списків, проте ми зосередимо увагу на його ручному налаштуванні.

Отже, розкривний список на аркуші властивостей має додаткові параметри, що визначають дані відповідного списку (Рисунок 4.17).

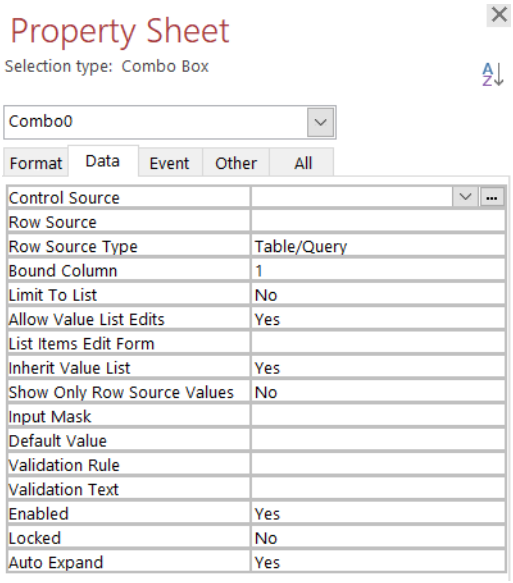


Рис. 4.17: Аркуш властивостей розкривного списку

В якості значення параметра *Row Source (Джерело рядків)* списку встановлюється запит, який повертає дані, необхідні для відображення в списку.

Найпростішим є випадок, коли в розкривному списку відображається єдине значення, яке потрібно зберігати в даному полі.

Приклад 12

Створіть в режимі конструктора форму для перегляду товарів з можливістю вибору коду виробника з розкривного

списку.
Оскільки форма повинна відображати дані про товари, її даними будуть усі дані про товари, які легко отримати за допомогою відповідного запиту. Цей запит є джерелом записів нашої форми.
Кожен виробник характеризується чотирма параметрами, тому додамо на форму три текстові поля та один розкривний список, відповідним чином змінюючи їх підписи та встановлюючи джерела керування (Рисунок 4.18).

Рис. 4.18: Макет форми в режимі конструктора

Розкривний список для вибору коду виробника повинен містити коди всіх виробників, а тому його джерелом рядків слід встановити запит з Лістингу 4.2 (Рисунок 4.19).

```
SELECT виробники.код
FROM виробники;
```

Лістинг 4.2: Виробники

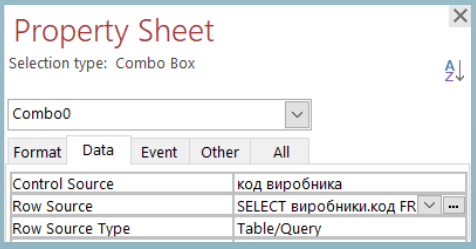


Рис. 4.19: Джерело рядків розкривного списку

Якщо перейти тепер в подання форми, то можна пересвідчитися, що у формі дійсно відображаються дані про товари, а код виробника можна обрати з розкривного списку (Рисунок 4.20).

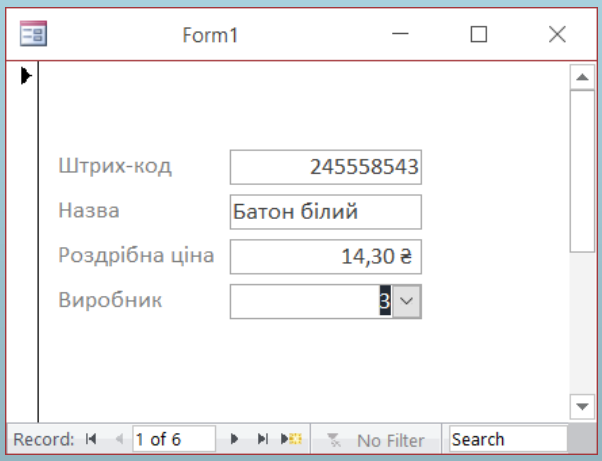


Рис. 4.20: Форма товарів

Але розкривний список з кодів виробників не є достатньо інформативним, як, наприклад, назва виробника чи взагалі всі дані про нього.

Насправді, розкривний список може містити кілька стовпців даних. Для цього необхідно відповідним чином модифікувати джерело рядків, а після цього на вкладці *Format (Формат)* аркуша властивостей встановити відповідне значення параметра *Column Count (Кількість стовпців)*.

Приклад 13

Створіть в режимі конструктора форму для перегляду товарів з можливістю вибору коду виробника з розкривного списку із зазначенням всіх даних про нього.

Розкривний список для вибору коду виробника тепер повинен містити всі дані всіх виробників, а тому його джерелом рядків слід встановити запит з Лістингу 4.3.

```
SELECT виробники.*
FROM виробники;
```

Лістинг 4.3: Виробники

Оскільки запит повертає 3 стопці даних, встановимо в якості значення параметра *Column Count (Кількість стовпців)* число 3 (Рисунок 4.21).

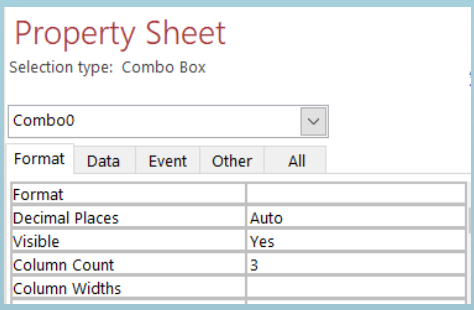


Рис. 4.21: Кількість стовпців розкривного списку

Якщо перейти тепер в подання форми, то можна пересвідчитися, що у розкривному списку дійсно відображаються всі дані про виробника, проте обирається його код (Рисунок 4.22).

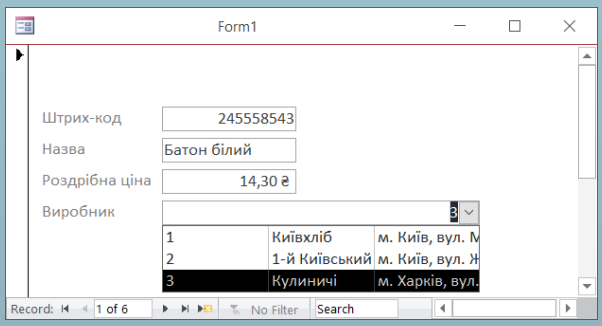


Рис. 4.22: Форма товарів

Але деколи доволі резонно взагалі приховати від користувача код виробника, який є суто технічною інформацією. Вище, на Рисунку 4.21 ми можемо побачити параметр *Column Widths (Ширини стовпців)*, який залишено порожнім, — цей параметр регулює ширини стовців, які відображаються в розкривному списку. Їх слід вказувати в сантиметрах через крапку з комою (Рисунок 4.23).

З одного боку, за допомогою цього параметра можна відрегулювати ширини стовпців (адже наразі не всі дані повністю видимі). Проте, якщо встановити для якогось із стовпців ширину в нуль, він зникне з розкривного списку взагалі. Якщо це зробити для поля, значення якого зберігається, воно теж зникне з списку і складатиметься враження, ніби в полі зберігається наступне поле — назва виробника. Проте, попри таку зовнішню оману, це

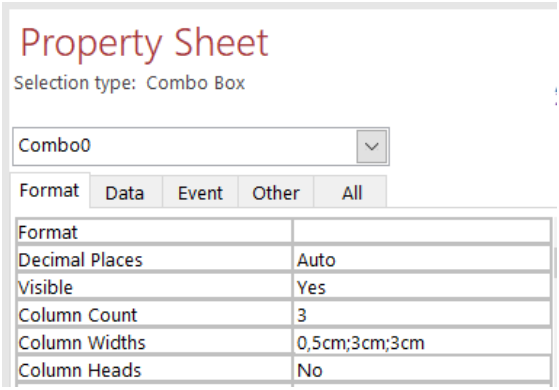


Рис. 4.23: Аркуш властивостей розкривного списку

лише спосіб приховування коду, та в полі все ще зберігається код виробника.

Приклад 14

Створіть в режимі конструктора форму для перегляду товарів з можливістю вибору виробника з розкривного списку із зазначенням назви та адреси, проте прихованням коду.
Оскільки потрібно приховати код виробника, встановимо для нього ширину 0см, а для решти полів — додатню (Рисунок 4.24).

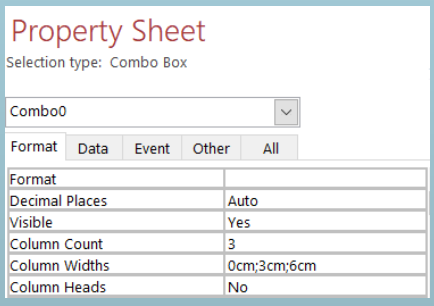


Рис. 4.24: Ширини стовпців розкривного списку

Якщо перейти тепер в подання форми, то можна пересвідчитися, що у розкривному списку дійсно відображаються назви і адреси виробника, а код взагалі відсутній навіть при виборі (Рисунок 4.25).

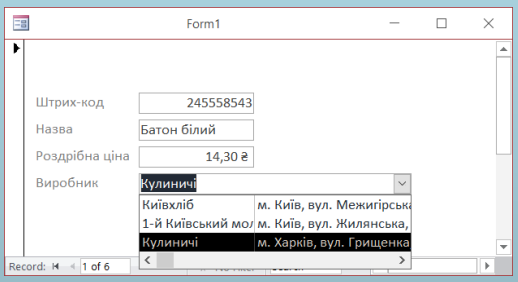


Рис. 4.25: Форма товарів

Пізніше ми переконаємося, що в полі дійсно все ще зберігається код виробника, а не його назва.

Ідентичним є налаштування ще одного елемента керування — *List Box (Список)*.

Налаштувати подання у вигляді розкривного списку можна й для полів таблиці. Для цього можна скористатися засобом *Lookup Wizard (Майстер підстановок)*, або ж налаштувати такі ж параметри, що і вище, на вкладці *Lookup (Підстановка)* властивостей поля, обравши попередньо для властивості *Display Control (Елемент керування)* значення *Combo Box* (Рисунок 4.26).

General | Lookup

Display Control	Combo Box
Row Source Type	Table/Query
Row Source	SELECT виробники.* FROM виробники;
Bound Column	1
Column Count	2
Column Heads	No
Column Widths	0cm;3cm
List Rows	16
List Width	Auto
Limit To List	Yes
Allow Multiple Values	No
Allow Value List Edits	No
List Items Edit Form	
Show Only Row Source Valu	No

Рис. 4.26: Розкривний список поля код виробника таблиці товари

Групи вибору

Якщо розкривні списки та списки підходять найкраще, коли слід здійснити вибір серед записів певної таблиці, то для вибору серед невеликої кількості фіксованих значень краще підходять *групи вибору (Option Group)*.

Під час додавання цього елементу керування на макет форми Microsoft Access запускає майстер груп вибору, який надасть можливість ввести необхідний перелік значень. Наприклад, для поля *кваліфікація* таблиці *працівники*, множина значень якого обмежена чотирма значеннями, відповідний список матиме вигляд, поданий на Рисунку 4.27.

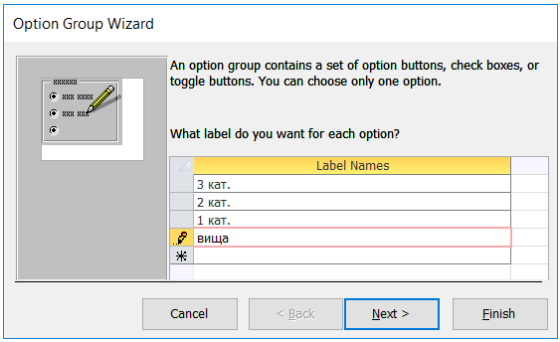


Рис. 4.27: Додавання групи вибору для кваліфікації працівника

Як і решта розглянутих нами елементів керування, цей елемент може бути зв'язаним. Джерело керування групи вибору задається одночасно для усієї групи на аркуші властивостей аналогічно тому, як це можна було виконати для решти елементів керування.

Єдиною проблемою даного типу елементів керування є те, що значенням вибору є не сам текст, а номер по порядку обраного варіанту, що дещо ускладнює роботу з такими інструментами.

Кнопки

Кнопка є прикладом виключно вільного елемента керування формою. Вона не передбачає жодного значення, а зазвичай призначена для виконання певних дій: збереження запису, переходу на іншу форму, виводу повідомлень тощо.

При додаванні кнопки на макет форми Microsoft Access запускає конструктор кнопок, який дозволяє обрати одну із заготовлених дій, яка буде виконуватися під час кліку на кнопку (Рисунок 4.28).

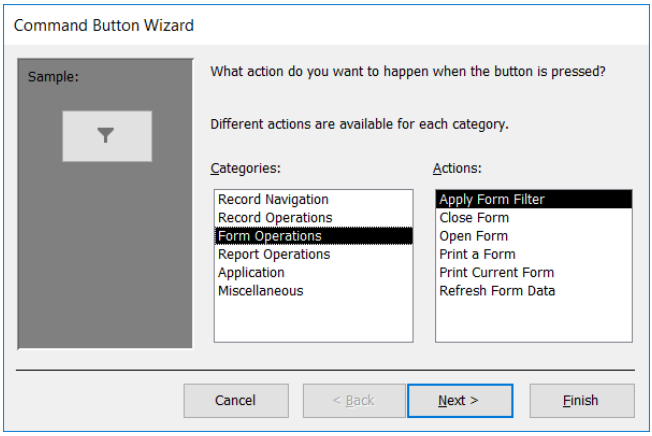


Рис. 4.28: Майстер кнопок

Перелік заготовлений дій зокрема передбачає відкриття і закриття форм та звітів, їх друк, перехід до попереднього або наступного запису на формі, додавання, видалення, копіювання, збереження запису тощо.

Більш тонке налаштування дій, що виконуються при клацанні на кнопку, можна здійснити за допомогою макросів, які будуть розглянуті далі в цій книзі.

4.2.4 Іменування елементів керування

Елементи керування, так само як і поля джерела записів, можуть бути використані під час обчислень як складові виразів. Щоб використання полів під час написання цих виразів було зручним, слід завжди звертати увагу на назви, які ви надаєте елементам керування на формі.

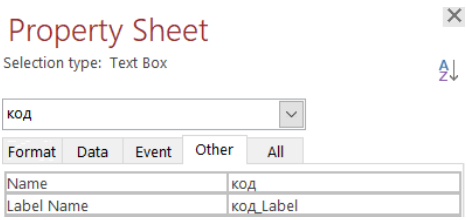


Рис. 4.29: Ім'я елемента керування

Ім'я елемента керування в будь-який момент можна змінити на вкладці *Other (Інше)* його аркуша властивостей (Рисунок 4.29).

Завжди давайте мнемонічні імена, аби надалі легко оперувати назвами елементів керування і не звертатися постійно до форми аби визначити назву потрібного текстового поля чи розкривного списку. Microsoft Access за промовчанням унікально іменує всі елементи керування, проте ці назви складно назвати змістовними.

4.2.5 Обчислювані поля в формах

Джерелом керування елементів керування формою можуть бути не тільки поля джерела записів форми, а й інші елементи керування та вирази над ними.

Наприклад, аби створити на формі товарів поле, в якому б відображався податок на одиницю товару, слід додати текстове поле на макет форми та натиснути на кнопку з трикрапкою в полі параметра *Control Source (Джерело керування)*. Microsoft Access відкриє конструктор виразів (Рисунок 4.30), в якому ви можете ввести будь-який вираз на основі наявних на формі елементів керування або полів джерела записів.

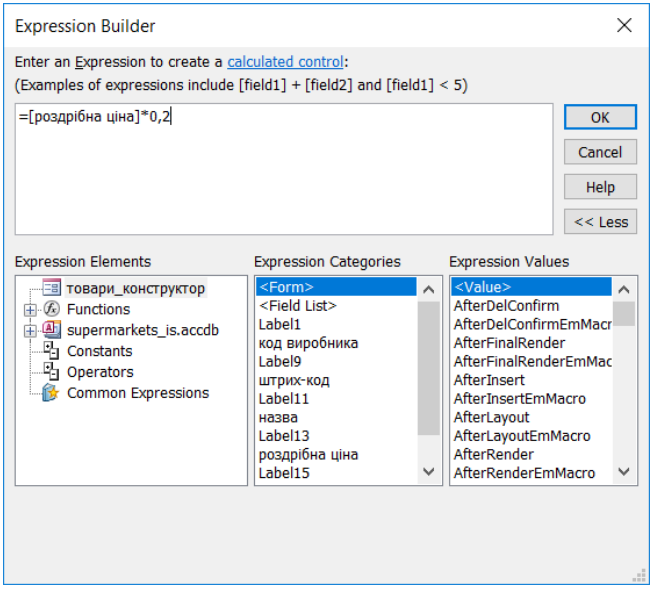


Рис. 4.30: Конструктор виразів

Наприклад, для обчислення податку на додану вартість, слід в якості джерела керування відповідного поля ввести вираз:

= [роздрібна ціна] * 0,2

Слід зазначити, що під час побудови виразів, якщо поле містить пробіл або інші спеціальні символи в своїй назві, його слід брати в квадратні дужки так, як це було показано вище. Аналогічне правило запису полів діє і для SQL-запитів в цій системі управління базами даних.

Аналогічно, в виразах можуть фігурувати й вільні поля. Якщо, наприклад, в полі *ставка* форми *товари* користувачем нашої системи вводитиметься відсоток податку на ціну товару, то сума податку обчислюватиметься виразом:

= [роздрібна ціна] * [ставка] / 100

Після створення такого поля, в режимі форми відповідне значення змінюватиметься одразу після зміни значень полів, за допомогою яких побудовано цей вираз.

Якщо на форму товари додати текстове поле, значення якого визначається виразом:

= [код виробника]

можна переконатися, що розкривний список вибору виробника дійсно зберігає його код, а не назву (Рисунок 4.31).

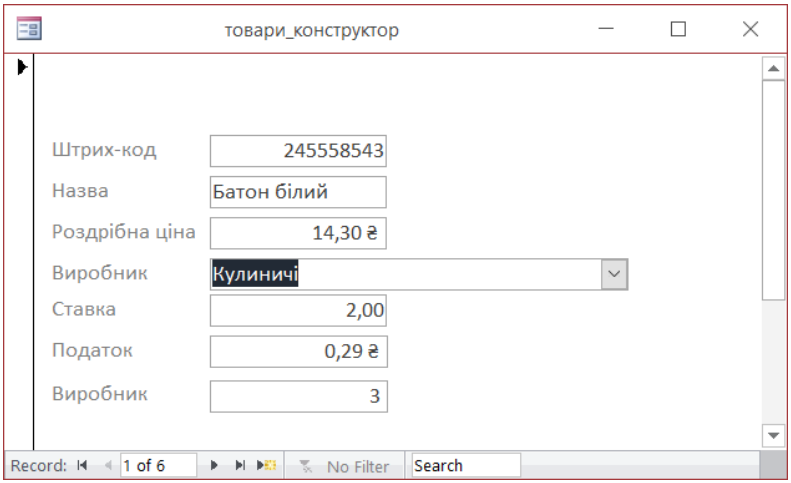


Рис. 4.31: Розкривний список зберігає значення *3*, хоча відображає назву виробника *Кулиничі*

4.2.6 Перевірка даних форм та таблиць

Параметри більшості зв'язаних полів дозволяють встановлення умов, які ці поля повинні виконувати. Ці умови називають правилами перевірки даних та встановлюються як значення параметра *Validation Rule (Правило перевірки)*.

Значення цього параметру — логічний вираз, що може використовувати значення як поточного, так і інших елементів керування форми.

Наприклад, якщо ціна товару не може бути меншою однієї гривні, слід вказати на аркуші властивостей поля *роздрібна ціна* для параметра *Validation Rule (Правило перевірки)* наступне значення (Рисунок 4.32):

[роздрібна ціна]>=1

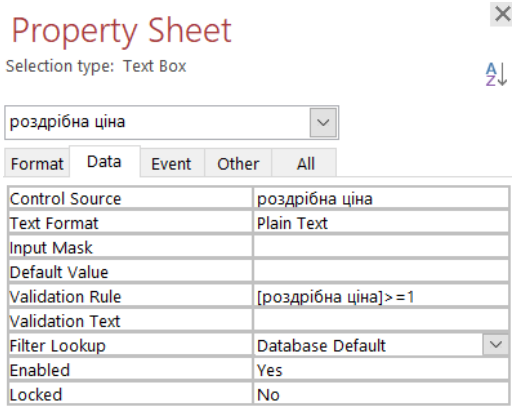


Рис. 4.32: Правило перевірки для ціни товару

Тоді, при додаванні нового запису в таблицю товари за допомогою цієї форми з ціною, мешою за 1, буде відображено повідомлення про помилку: *Введене значення не задовольняє правило перевірки, визначене для поля чи елемента керування.*

У разі необхідності, можна задати текст повідомлення, який буде відображено у разі невиконання заданої умови. Цей текст слід ввести в якості значення параметра *Validation Text (Текст перевірки)*. Приклад можливого повідомлення для ціни товару подано на Рисунку 4.33.

Подібним чином можна встановлювати правила перевірки й для полів таблиць в режимі конструктора. Таким чином можна убезпечити базу даних від некоректних даних.

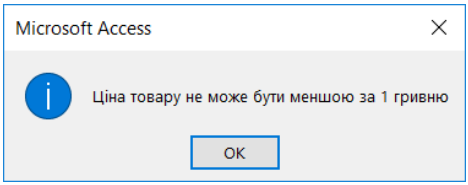


Рис. 4.33: Ціна товару не може бути меншою за 1 гривню

4.2.7 Підформи

Підформи є одночасно формами і елементами керування, оскільки відповідають за відображення даних певного запиту у зв'язку з даними деякої іншої форми.

Ми вже мали справу з підформами, коли розглядали створення форм за допомогою майстра. На Рисунку 4.34 перелік товарів відображається в межах підформи основної форми *виробники*.

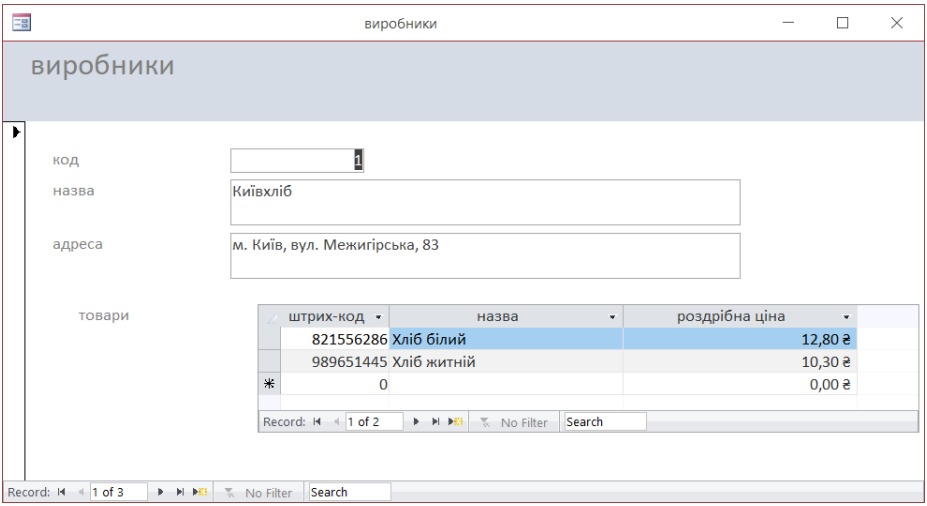


Рис. 4.34: Форма *виробники* з підформою товарів

В даному випадку ми маємо справу з табличною підформою, хоча насправді тип підформи може бути довільним.

Аби додати підформу на форму виробників, створену нами раніше, слід додати на її макет елемент керування *Subform/Subreport (Підформа/Пі-*

дзвіт). При його додаванні Microsoft Access відкриває майстер підформ, в якому можна налаштувати зв'язок між основною формою та підформою. Ми ж зосередимося на налаштуванні відповідних параметрів вручну.

Аркуш властивостей елемента керування містить три спеціальних параметри (Рисунок 4.35). Параметр *Source Object (Об'єкт-джерело)* визначає форму, звіт, запит чи таблицю, на основі якої буде побудовано підформу, а параметри *Link Master Fields (Пов'язані поля основної форми)* та *Link Child Fields (Пов'язані поля підформи)* визначають зв'язок між полями основної форми та підформи.

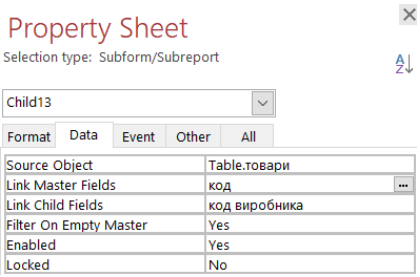


Рис. 4.35: Аркуш властивостей підформи

Два останні поля вказують те, які поля повинні бути рівними у пов'язаних записів. Цю відповідність можна налаштувати за допомогою інструменту пов'язування полів підформи, який відкривається при натисненні на кнопку з трикрапкою в полі будь-якого з цих двох параметрів (Рисунок 4.36).

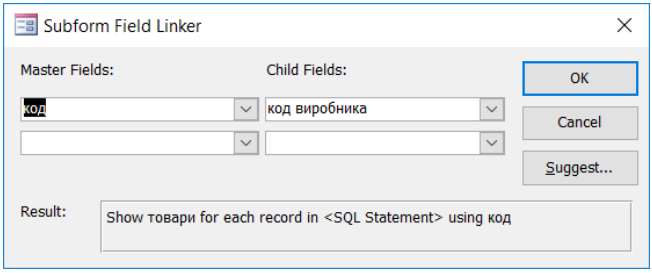


Рис. 4.36: Пов'язування полів підформи з полями основної форми

В даному прикладі полю *код* основної форми відповідає поле *код виробника* підформи товарів.

Що стосується ланцюгів підформ, де перша підформа залежить від основної, друга — від першої і т.д., то їх реалізація в режимі конструктора є значно складнішою, аніж створення за допомогою майстра, а тому в рамках цієї книги не розглядатиметься.

4.3 Параметризовані запити

Далеко не завжди вистачає звичайних запитів для задоволення усіх потреб, адже вони не мають змоги врахувати дані, які ввів користувач в межах інформаційної системи, а отже, не здатні реалізувати навіть звичайний пошук.

З огляду на це в Microsoft Access підтримуються запити з параметрами: будь-яке поле, що не належить до табличного виразу запиту, вважається тут параметром.

Нехай необхідно відобразити перелік товарів за частиною штрих-коду, введеною користувачем. В попередній частині цієї книги ми вже розглядали подібні запити — це запит на порівняння з шаблоном за допомогою оператора *LIKE*.

Оскільки шаблон тепер повинен містити не фіксований рядок, а певний рядок, який введе користувач, для формування шаблону доведеться скористатися оператором конкатенації &, який ми вже розглядали під час розбору однієї з вправ раніше.

Якщо ми бажаємо, щоб користувач запрошувався до вводу штрих-коду повідомленням «Введіть штрих-код», слід виконати запит, код якого подано в Лістингу 4.4.

```
SELECT товари.*
FROM товари
WHERE [штрих-код] LIKE "*" & [Введіть штрих-код] & "*";
```

Лістинг 4.4: Товари за частиною штрих-коду

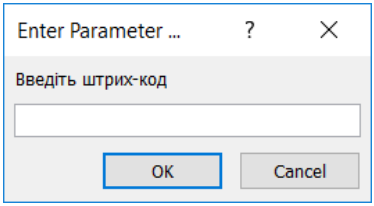


Рис. 4.37: Запрошення до вводу параметра

Після запуску цього запиту Microsoft Access відобразить запрошення до вводу параметра з потрібним нам повідомленням (Рисунок 4.37).

А після введення користувачем значення параметра буде відображено перелік результатів такого пошуку.

Проте використання такого варіанту взаємодії з користувачем не є зручним, оскільки запити все-таки є частиною бази даних, а не інформаційної системи, їх не можна видозмінювати, форматувати тощо.

Саме тому в запитах Microsoft Access окрім вище розглянутих параметрів є можливість використовувати в якості параметрів також значення елементів керування формою.

Загалом до будь-якого елемента керування форми, якщо вона відкрита в режимі форми, можна звернутися наступним чином:

Forms!<назва форми>!<ім'я елементу керування>

Нехай на вільній (з порожнім джерелом записів) формі *Головна* розміщено текстове поле, в яке користувач повинен вводити штрих-код товару. Тоді код запиту, який шукатиме товари за введеним фрагментом штрих-коду, матиме вигляд, поданий в Лістингу 4.5.

```
SELECT товари.*
FROM товари
WHERE товари.[штрих-код] LIKE
    "*" & Forms!Головна![штрих-код] & "*";
```

Лістинг 4.5: Товари за частиною штрих-коду

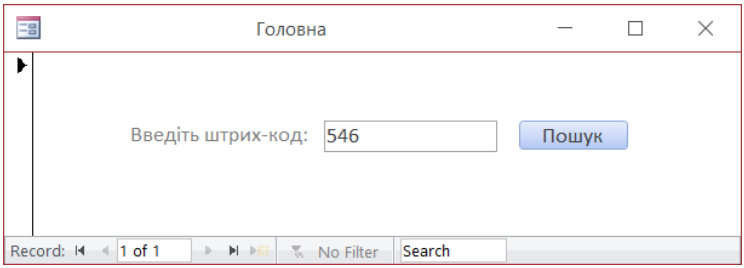


Рис. 4.38: Форма пошуку товарів

Якщо побудувати на основі даного запиту форму на декілька елементів, а на форму *Пошук товарів* додати кнопку, що її відкриває, то можна отримати цілком повноцінну пошукову систему товарів (Рисунки 4.38–4.39).

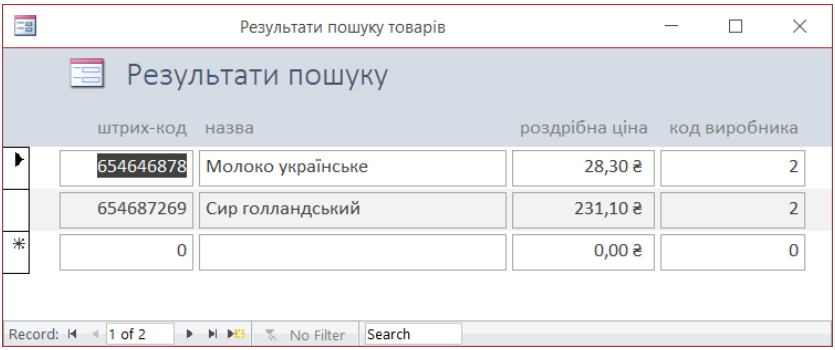


Рис. 4.39: Форма результатів пошуку

Слід зазначити, що в якості параметра в запиті можна використовувати й зв'язані поля форми. В такому разі значення цього поля в певний момент залежатиме від того, який запис наразі активний на формі.

Таким чином можна, наприклад, відобразити перелік складів, на яких зберігається поточний товар. Відповідний запит подано в Лістингу 4.6.

```
SELECT склади.*
FROM склади INNER JOIN зберігається
    ON склади.код = зберігається.[код складу]
WHERE зберігається.[штрих-код товару] =
    Forms![товари]![штрих-код];
```

Лістинг 4.6: Товари за частиною штрих-коду

Можна аналогічним чином побудувати на основі даного запиту форму на декілька елементів, а на форму *Пошук товарів* додати кнопку, що її відкриває.

4.4 Огляд засобів створення звітів

Звіти в Microsoft Access зазвичай призначені для документів з певними узагальненими даними, що передбачають подальший друк.

В цілому, створення звітів майже ідентичне створенню форм, оскільки звіти так само відображають дані з певного запиту. На відміну від форм, звіти дуже рідко бувають вільними, хоча така можливість і не виключається, а також мають окремий дизайн.

Як і для форм, для кожного звіту існують подання конструктора, макету та власне звіту. Також існує додаткове подання попереднього перегляду, що передбачає перегляд звіту в тому вигляді, в якому його буде подано на друк.

Як і форми, звіти можуть бути створені автоматично, зокрема за допомогою майстра (Рисунок 4.40), або ж за допомогою конструктора, починаючи з порожнього макету.

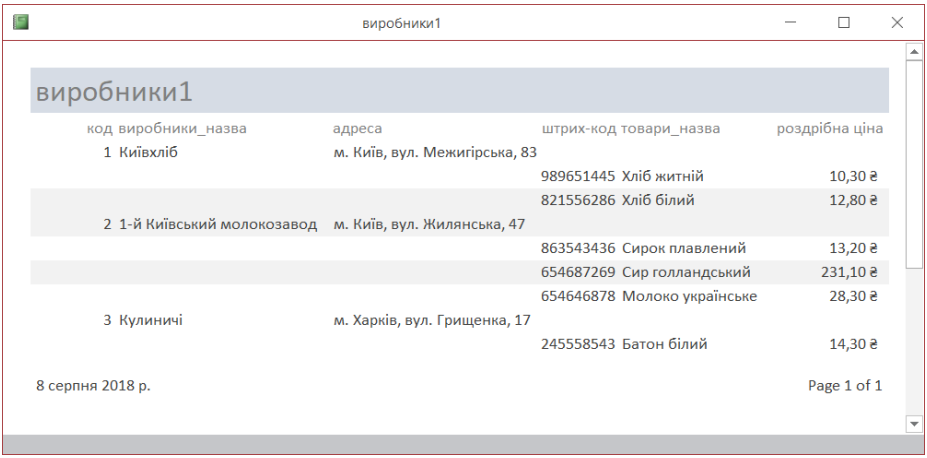


Рис. 4.40: Звіт про виробників та їх товари, створений за допомогою майстра

Основна відмінність звіту від форми — він завжди подає дані з джерела записів у вигляді таблиці. В деякій мірі звіт можна вважати певним розширенням форми на декілька елементів, що передбачає можливість групування даних.

4.4.1 Групування даних у звітах

Групування даних у звітах дещо відрізняється від операції групування в SQL-запитах. Тут групування дозволяє візуально виділити записи, що мають однакові значення певних полів.

Наприклад, на Рисунку 4.40 у звіті, створеному майстром, автоматично здійснено групування за даними про виробника. Таким чином, користувачеві легше визначити візуально в кого з виробників, наприклад, більший вибір продукції.

Розглянемо автозвіт, побудований на основі запиту, що визначає для кожного товару дані його виробника (Рисунок 4.41).

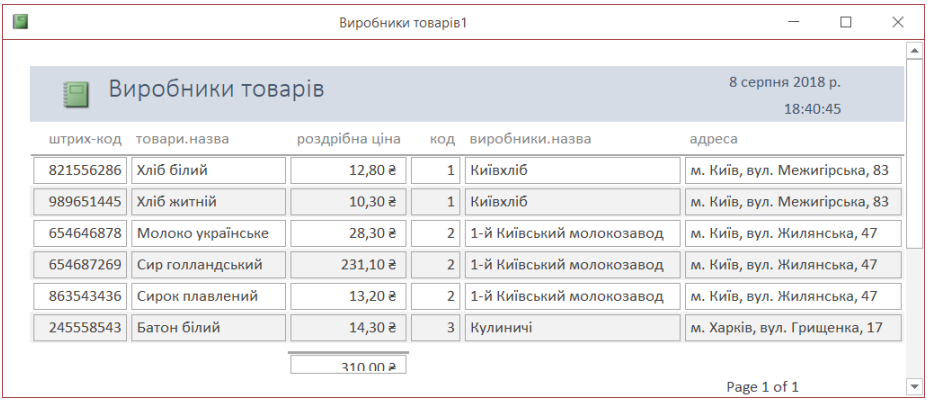


Рис. 4.41: Автозвіт на основі запиту про виробників та їх товари

Якщо ми бажаємо згрупувати дані в цій таблиці за кодом виробника, слід перейти в режим макету звіту за допомогою інструменту в правому нижньому куті вікна та в контекстному меню стовпця код обрати пункт *Group On код (Групувати за полем код)*. Після цього в режимі конструктора з'явиться додатковий блок під назвою *код Header (Заголовок групи код)* (Рисунок 4.42).

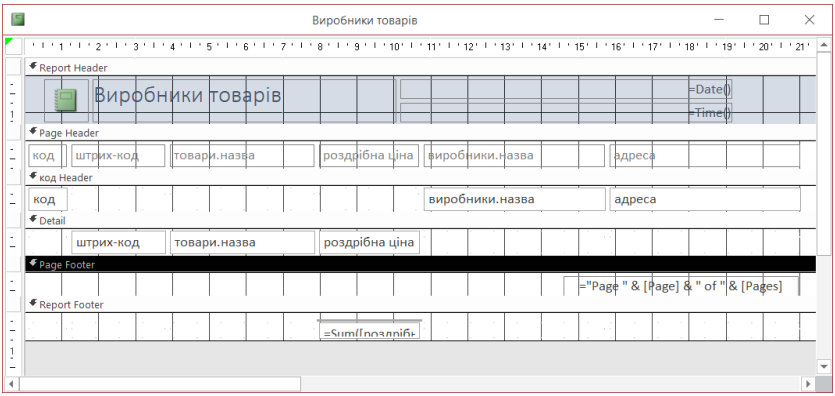


Рис. 4.42: Звіт з групуванням в режимі конструктора

В цьому блоці містяться дані, які відображатимуться на початку кожного нового значення коду виробника. Сюди слід перетягнути й назву та адресу виробника, щоб досягнути ідентичного вигляду до Рисунка 4.40.

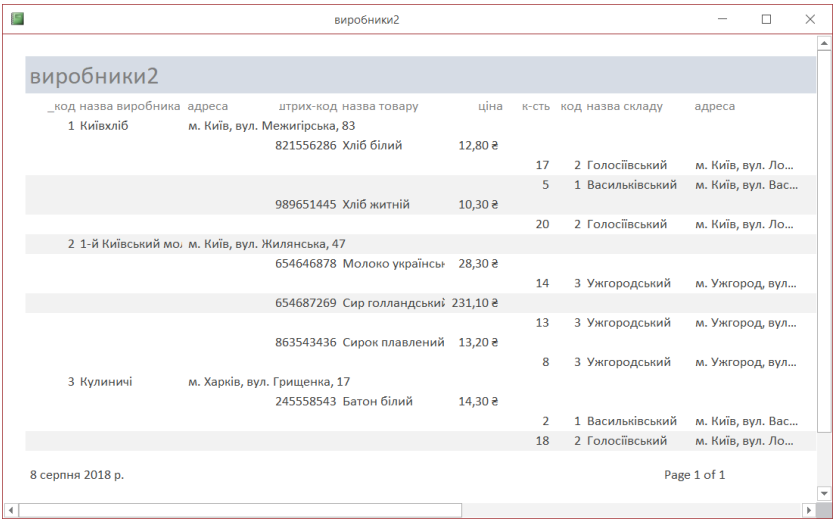


Рис. 4.43: Звіт з багаторазовим групуванням

Кількість групувань в звітах необмежена: подальше повторення цих дій для інших стовпців створюватиме ієрархію так, як це показано на Рисунку 4.43.

4.5 Події та їх обробники. Макроси

Означення 51. *Подія (в програмуванні) – це дія або випадок, що розпізнається програмним забезпеченням та може бути оброблена.*

Прикладами подій в Microsoft Access є клацання на об'єкт, оновлення значення об'єкта, введення символу в текстове поле тощо. Їх перехоплює Access та дозволяє визначати обробники за допомогою макросів. Кожен об'єкт інтерфейсу (форма, звіт, елементи керування тощо) має окремий набір властивих йому подій, обробники до яких можна встановити на вкладці *Events (Події)* їх аркуша властивостей (Рисунок 4.44).

Макроси, подібно до звичайної програми, містять набір інструкцій — макрокоманд, кожна з яких виконує певну дію. Макрокомандами Microsoft Access, наприклад, є відкриття та закриття таблиць, запитів, форм, звітів, відображення повідомлення тощо.

Макроси можна створювати за допомогою інструменту *Macro* окремого

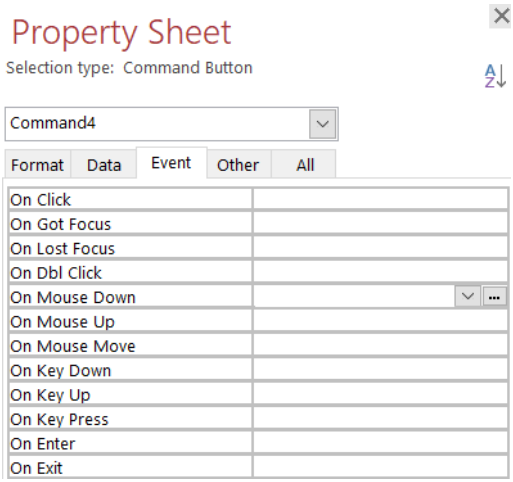


Рис. 4.44: Типові події для кнопок

блоку на вкладці *Create (Створити)* (Рисунок 4.45). Для призначення такого макросу обробником події слід вибрати у відповідному полі його назву з розкривного списку. Також на аркуші властивостей можна створювати вбудовані макроси для кожного обробника, натискаючи на кнопку з трикрапкою у полів відповідної події.

Рис. 4.45: Створення макросу

Порада. Надавайте перевагу збережуваним макросам над вбудованими, оскільки вони можуть використовуватися повторно.

Конструктор, що відкривається після створення макросу, надає доступ до каталогу макрокоманд, який пропонує кілька десятків команд, згрупованих за вісьмома категоріями: *Data Entry Operations (Операції вводу даних)*, *Data Import/Export (Імпорт/Експорт даних)*, *Database Objects (Об'єкти бази даних)*, *Filter/Query/Search (Фільтр/Запит/Пошук)*, *Macro Commands (Команди макросів)*, *System Commands (Системні команди)*, *User Interface Commands (Команди інтерфейсу)* та *Window Management (Керування ві-*

кнами).

Також конструктор надає чотири структури контролю за ходом виконання макросу: *Comment (Коментар)*, *Group (Група)*, *If (Якщо)*, *Submacro (Підмакрос)*.

Ми зупинимося на використанні основних макрокоманд цих категорій в рамках вирішення типових завдань в ході створення інформаційних систем. З повним їх переліком ви завжди можете познайомитися в конструкторі макросів, де Microsoft Access надає зокрема й короткий опис призначення цих команд.

4.5.1 Пошук даних за допомогою фільтра

Раніше ми розглянули один із варіантів здійснення пошуку товару за фрагментом його штрих-коду. Проте в ньому ми повинні були запускати клацанням на кнопку іншу форму для відображення результату. А що, якщо результат необхідно відобразити в тому ж вікні?

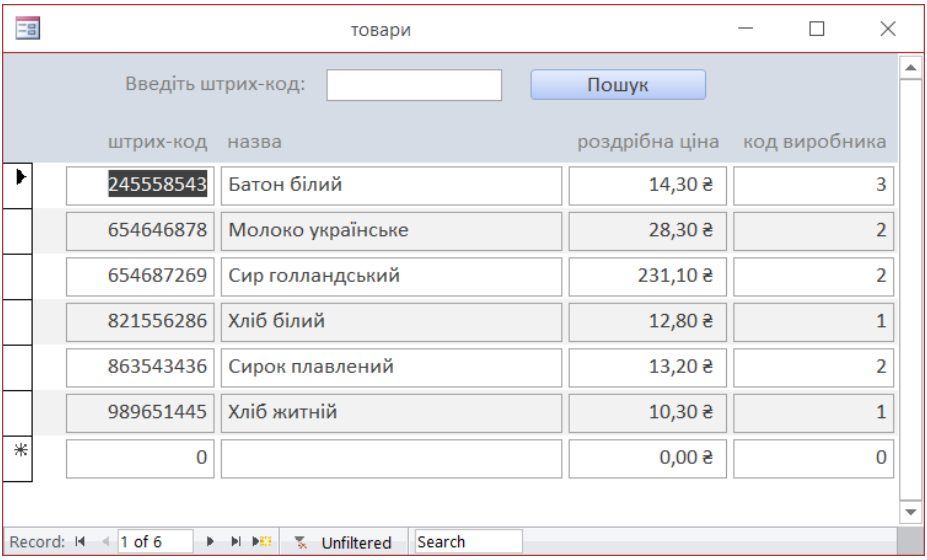


Рис. 4.46: Форма *товари* з полем для пошуку

Нехай маємо форму *товари* на декілька елементів, в заголовку якої розміщено текстове поле *фрагмент*, в яке користувач вводитиме фрагмент штрих-коду, та кнопку для здійснення пошуку (Рисунок 4.46).

Макрокоманда *SetFilter (ВстановитиФільтр, ЗадатьФильтр)* категорії *Filter/Query/Search* надає можливість відфільтровувати (приховувати) ті записи джерела записів форми, які не задовольняють вказану умову. А тому, нам потрібно створити макрос, який задавав би відповідний фільтр для нашої форми.

Умова відповідності запису ключу пошуку залишається незмінною:

[штрих-код] LIKE "*"& Forms!товари!фрагмент & "*"

Чому ж перший параметр команди вказаний безпосередньо, а для другого використано повний шлях до елемента керування?

Слід відрізняти поля форми від значень елементів керування. Поле форми — це поле джерела її рядків, воно являє собою стовпець певного запиту. Елемент керування формою, як було зазначено вище, може бути вільним, а може бути зв'язаним з певним полем.

В кожен певний момент часу на формі активний певний запис форми, якому відповідає певний запис джерела її рядків. А тому елемент керування, розташований на формі, теж має одне значення, що відповідає активному запису.

В такому разі вираз **Forms!товари![штрих-код] LIKE "*"& Forms!товари!фрагмент & "*"** трактуватиметься як порівняння значень елементів керування і порівнюватиме значення штрих-коду в активному записі зі значенням текстового поля пошуку, а вираз **[штрих-код] LIKE "*"& Forms!товари!фрагмент & "*"** система трактуватиме як порівняння поля зі значенням елементу керування, через що тут використовуватимуться відповідні значення штрих-коду для кожного запису джерела керування і кожне з них порівнюватиметься з текстовим полем пошуку.

Слід зауважити, що з використанням повного шляху до елемента керування, його значення можна використовувати будь-де, в тому числі і в заголовку чи колонтитулах поточної форми та будь-де в інших активних формах.

Для роботи макросу *Пошук товарів* достатньо в якості значення параметру *Where Condition (Умова)* вказати вищенаведену умову (Рисунок 4.47).

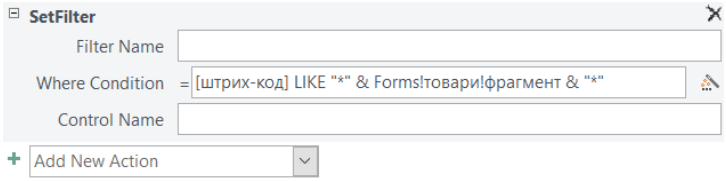


Рис. 4.47: Макрокоманда *SetFilter (ВстановитиФільтр, ЗадатьФильтр)*

Створений макрос тепер слід встановити в якості обробника події *On Click (Натиснення)* для відповідної кнопки (Рисунок 4.48).

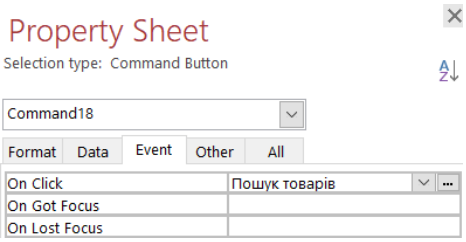


Рис. 4.48: Встановлення обробника події *On Click (Натиснення)*

Після виконання вказаних вище дій ми отримаємо форму з можливістю пошуку товарів за фрагментом штрих-коду по кліку на відповідну кнопку. Наприклад, результат роботи для фрагменту штрих-коду «44» зображено на Рисунку 4.49.

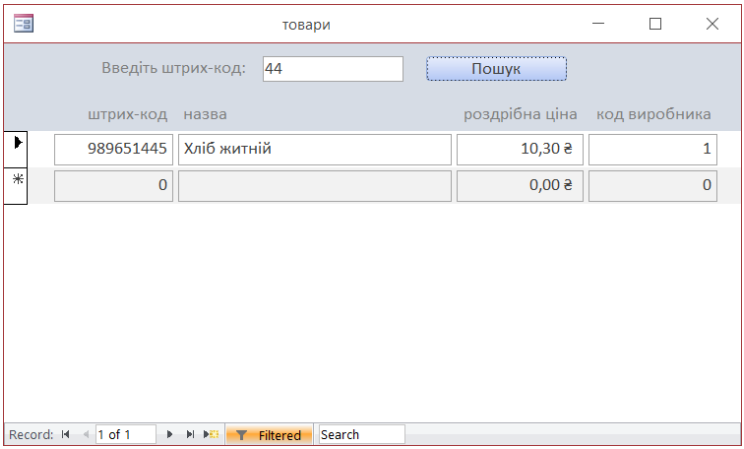


Рис. 4.49: Результати пошуку

Слід зазначити, що характерною особливістю такої реалізації пошуку є початковий стан форми. Перед здійсненням пошуку, виконаного за допомогою фільтру, на формі завжди відображаються усі записи джерела.

4.5.2 Пошук даних за допомогою запиту

Ще один варіант реалізації пошуку в межах однієї форми використовує запит зі значенням відповідного елемента керування в якості параметра.

Для тієї ж форми *товари* на декілька елементів, в заголовку якої розміщено текстове поле *фрагмент* (Рисунок 4.46) такий SQL-запит матиме вигляд, поданий в Лістингу 4.7.

```
SELECT товари.*
FROM товари
WHERE товари.[штрих-код] LIKE
    "*" & Forms!товари!фрагмент & "*";
```

Лістинг 4.7: Товари за частиною штрих-коду

Аби результати пошуку відображалися в цій же формі, слід замінити джерело її записів на цей запит.

Якщо спробувати вносити якісь зміни до поля фрагмент ми пересвідчимося, що до жодних змін це не призводить. Справді, Microsoft Access не оновлює автоматично джерело записів. Якщо зараз натиснути F5 після введення фрагменту штрих-коду (ця клавіша зазвичай відповідає за оновлення), то результат відповідатиме нашим очікуванням. Проте як просимулювати натискання цієї клавіші при натисненні на кнопку?

Макрокоманда *Requery (ПовторитиЗапит, Обновление)* категорії *Filter/Query/Search* надає можливість оновити зазначений в якості параметра елемент керування форми. Якщо параметр макрокоманди не встановлено (Рисунок 4.50), здійснюється оновлення результатів джерела записів активного об'єкту. А тому, нам потрібно створити макрос, який би запускав цю команду без встановленого значення параметра.

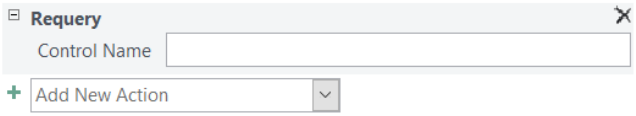


Рис. 4.50: Макрокоманда *Requery (ПовторитиЗапит, Обновление)*

Створений макрос тепер слід встановити в якості обробника події *On Click (Після клацання)* для відповідної кнопки. Після виконання цих дій ми отримаємо форму з можливістю пошуку товарів за фрагментом штрих-коду по кліку на відповідну кнопку. Наприклад, результат роботи для фрагменту штрих-коду «44» зображено на Рисунку 4.51.

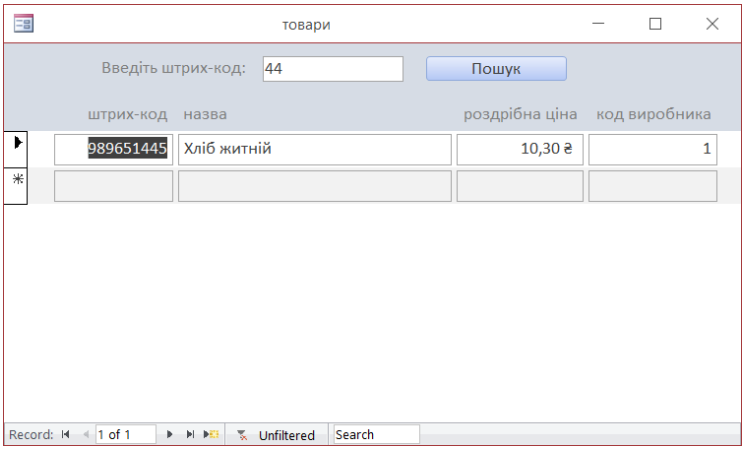


Рис. 4.51: Результати пошуку

4.5.3 Обробка крайніх випадків

Далеко не завжди операції, які запитує користувач, виконуються успішно. Деколи це відбувається з вини користувача, коли він вводить некоретні вхідні дані, а деколи — з певних технічних причин. В промислових інформаційних системах важливо повідомляти користувача про такі крайні випадки та надавати, за змоги, пропозиції щодо можливих шляхів вирішення проблеми.

Продовжимо удосконалювати форму для пошуку товарів. Наразі у випадках, коли пошук був неуспішним (тобто за фрагментом, введеним користувачем, нічого не було знайдено), форма просто стає порожньою. З точки зору користувача невідомо, чи це власне і є результат пошуку, або ж дані все ще вантажаться чи взагалі виникла якась помилка у роботі системи. Тож забезпечимо користувача відповідною інформацією у вигляді повідомлення.

Очевидно, що в даному випадку нам доведеться скористатися умовним переходом, адже повідомлення повинно бути відображене лише у разі порожнього результату. Умовний перехід в макросах Microsoft Access забезпечується блоком *If (Якщо, Если)*, що належить до структур контролю за ходом виконання макросу (*Program Flow*). Цей блок складається з умовного виразу (*Conditional expression*) та гілки *Then (To)* — послідовності команд, які повинні бути виконані у разі істинності цього виразу. Необов'язкова гілка *Else (Інакше, Иначе)* визначає послідовність команд, які повинні бути виконані у разі хибності умови.

В нашому випадку, умовою є порожність результату запиту на пошук товару, а в гілці *Then (То)* повинно відображатися відповідне повідомлення.

Результат запиту є порожнім тоді і тільки тоді, коли кількість записів в ньому рівна нулю. А тому для запису умовного виразу скористаємося вбудованою функцією *DCount*, яка приймає в якості агрументів назву поля та назву об'єкта (таблиці чи запиту), і повертає кількість записів з відповідним непорожнім полем. Власне ця вбудована функція працює аналогічно до агрегатної функції *COUNT*.

В нашому випадку, якщо запит для пошуку товарів за фрагментом штрих-коду (Лістинг 4.7) збережено під назвою *Результати пошуку товарів*, умовний вираз матиме наступний вигляд:

$$\text{DCount}(\text{"[штрих-код]"};\ \text{"[Результати пошуку товарів]"}) = 0$$

Залишається тільки додати відповідну команду для відображення повідомлення. Макрокоманда *MessageBox (ВікноПовідомлення, ОкноСообщения)* категорії *User Interface Command* надає можливість відобразити вікно повідомлення, що містить попередження чи інформацію. В якості параметрів команда приймає вміст повідомлення, необхідність звукового сигналу, тип повідомлення (інформаційне, попереджувальне, критичне тощо) та заголовок. А тому, нам потрібно створити макрос, який би запускав цю команду з відповідними значеннями параметрів. Загальний код макросу пошуку подано на Рисунку 4.52.

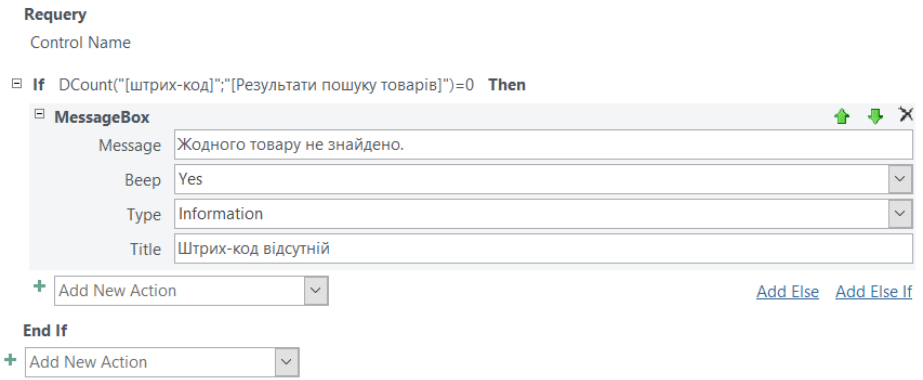


Рис. 4.52: Макрос для здійснення пошуку з додатковим інформуванням

Після внесення наведених вище змін до макросу здійснення пошуку, при невдалому пошуку користувачеві відображатиметься інформаційна підказ-

ка (Рисунок 4.53).

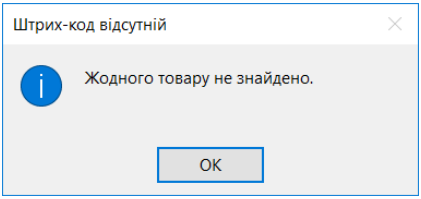


Рис. 4.53: Жодного товару не знайдено

Макрос, поданий вище, спочатку оновлює вміст форми пошуку товарів (команда *Requery (ПовторитиЗапит, Обновление)*), а вже потім перевіряє, чи отриманий результат порожній. Якщо у випадках порожнього результату не слід змінювати відображений раніше перелік (тобто певні дії потрібно виконувати лише за нормальних умов, а не завжди), поведінку за коректних умов слід вмістити в гілку *Else (Інакше, Иначе)*, як це зображено на Рисунку 4.54.

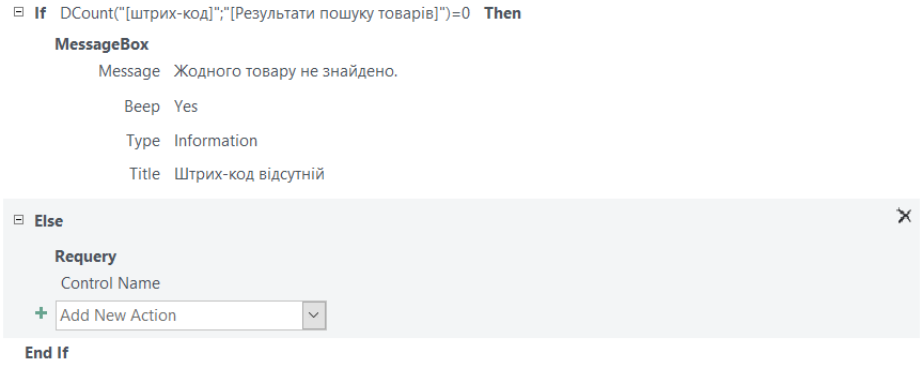


Рис. 4.54: Макрос для здійснення пошуку з додатковим інформуванням (дані оновлюються лише за нормальних умов)

Зверніть увагу, що функція DCount приймає в якості агрументів два рядки — назву поля та назву об'єкта (таблиці чи запиту). А тому, якщо викликати її так, як це вказано в Лістингу 4.8, спрацювання функції майже завжди буде некоректним, оскільки в цьому разі ви передаєте не самі рядки «[штрих-код]» та «[Результати пошуку товарів]», а значення відповідних полів активної форми.

```
DCount([штрих-код]; [Результати пошуку товарів]) = 0
```

Лістинг 4.8: Некоректна умова

Оскільки ми маємо справу з формою товарів, то перший параметр отримає значення штрих-коду поточного товару на нашій формі, а другий параметр взагалі не буде обчислений (адже поля з такою назвою немає) і буде відображено помилку, подану на Рисунку 4.55.

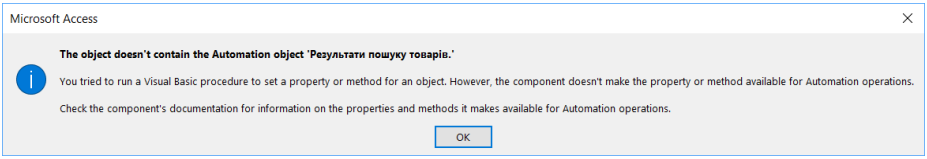


Рис. 4.55: Помилка: *Об'єкт не містить елемента "Результати пошуку товарів"*

Ця помилка є доволі поширеною, а тому завжди перевіряйте коректність введений вами параметрів.

4.5.4 Обмін інформацією між об'єктами системи

Вище ми вже розглядали можливості, які надає Microsoft Access для обміну даними між різними його об'єктами. Проте для такого обміну обов'язковою умовою є перебування цих об'єктів у відкритому стані, що не завжди є зручним. Адже користувач може випадково (або й умисно) закрити критично необхідну для роботи системи форму, через що деякі важливі для роботи інших об'єктів дані буде безповоротно втрачено. Саме в таких ситуаціях на допомогу приходять змінні.

Загалом, в макросах виділяють два типи змінних: локальні та тимчасові. Локальні змінні діють тільки в межах одного виконання макросу і поставлену проблему вирішити не здатні, а от тимчасові змінні, які здатні зберігати значення впродовж активності бази даних, цілком можуть допомогти у вирішенні певних проблем роботи інформаційної системи.

Раніше ми реалізовували можливість розрахунку податку на одиницю товару, дозволяючи користувачеві вводити відсоткову ставку. Проте якщо робити таке введення можливим щоразу, коли необхідно розрахувати податок, зростає можливість для користувача зробити випадкову чи навмисну помилку, вказавши в різних місцях системи різну ставку. Спробуємо запобігти виникненню цієї можливості.

Нехай маємо форму (Рисунок 4.56), призначену для встановлення відсоткової ставки податку. Вона містить текстове поле *ставка*, в яке користувач має ввести число, та кнопку. При натисненні на кнопку *Зберегти* ми очікуємо, що введену відсоткову ставку буде успішно збережено в базі даних і її можна буде в подальшому використовувати.

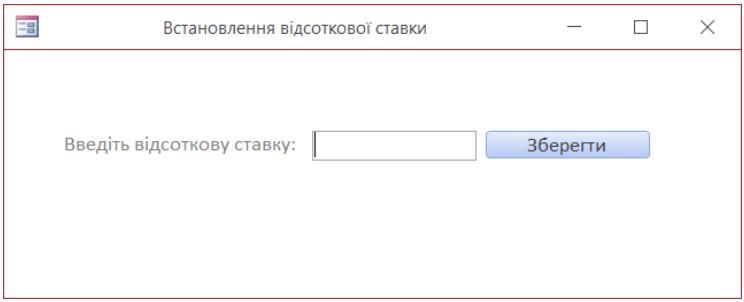


Рис. 4.56: Форма для встановлення відсоткової ставки податку

Отже, кнопка *Зберегти* повинна зберегти значення ставки у певну тимчасову змінну.

Макрокоманда *SetTempVar (ВстановитиТимчасовуЗмінну, ЗадатьВремПеременную)* категорії *Macro Commands* надає можливість встановлення значення деякої тимчасової змінної за її назвою. А тому, нам потрібно створити макрос, який би запускав цю команду, надаючи змінній *ставка* значення поля створеної нами форми, використовуючи його повне ім'я (Рисунок 4.57).

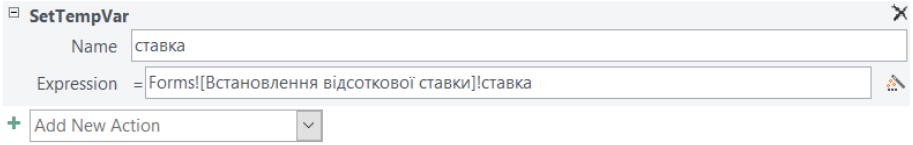


Рис. 4.57: Макрокоманда *SetTempVar (ВстановитиТимчасовуЗмінну, ЗадатьВремПеременную)*

Створений макрос тепер слід встановити в якості обробника події *On Click (Натиснення)* для відповідної кнопки. Після виконання цих дій натиснення на кнопку *Зберегти* призводитиме до збереження відсоткової ставки в змінній *ставка*.

Загалом, значення будь-якої тимчасової змінної можна отримати наступним чином:

TempVars!<назва тимчасової змінної>

Отож, ми можемо додати нові елементи керування до форми *товари* на декілька елементів, аби відобразити ставку податку (=[TempVars]![ставка] & "%") та суму податку (=[роздрібна ціна]*[TempVars]![ставка]/100) (Рисунок 4.58).

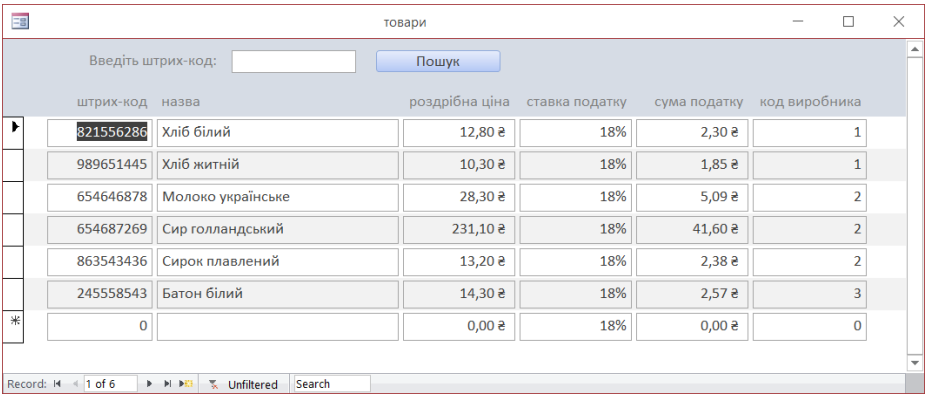


Рис. 4.58: Форма *товари* з даними про податки

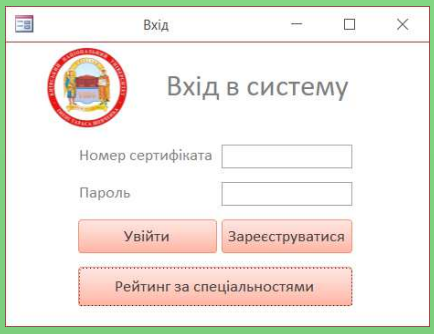
4.6 Вправи

Пропонуємо закріпити знання, виконуючи запропоновані нижче вправи. Ці вправи вимагають додаткових матеріалів, які ви можете завантажити на сторінці книги в Інтернеті.

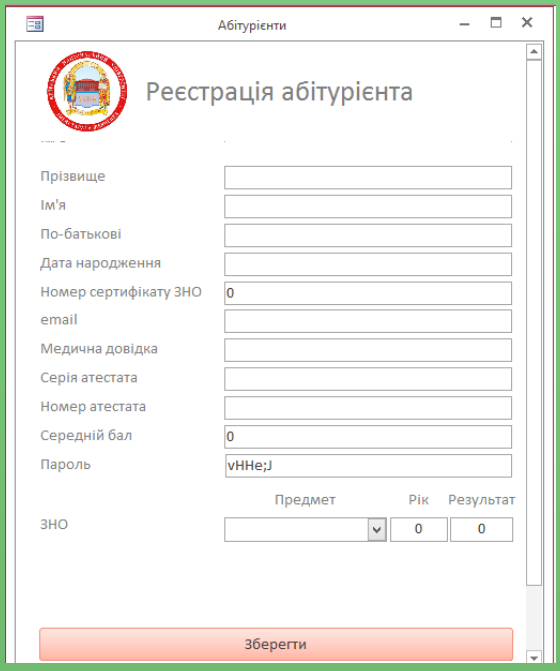
Вправа 123

II етап II Дистанційного турніру з інформаційних технологій серед учнівської молоді
Київський національний університет імені Тараса Шевченка вирішив запустити власний аналог системи електронного вступу для значного полегшення процедури подання абітурієнтами заяв на вступ до бакалаврату. З цього приводу адміністрація університету вирішила звер-

нутися до учасників Дистанційного турніру.
Ваше завдання – у середовищі системи керування базами даних створити застосунок, який відкривається вікном входу користувача:



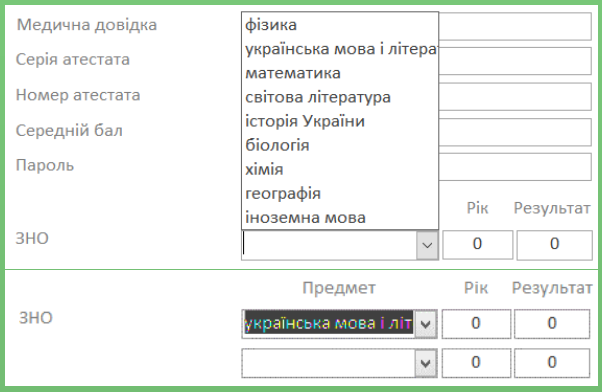
Натискання на кнопку «Зареєструватися» призводить до закриття вікна входу та відкриття відповідного вікна реєстрації нового абітурієнта в системі:



В цьому вікні розміщується форма реєстрації з полями:

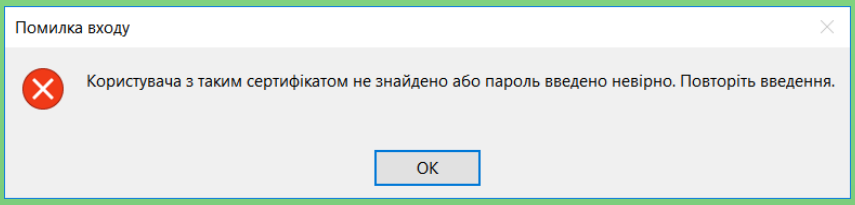
- Прізвище, Ім'я, По-батькові – маска вводу: перша літера велика, інші – малі (не більше 10 символів кожне);
- Дата народження – формату дати;
- Номер сертифікату ЗНО – унікальне число;
- email – унікальний текстовий рядок у форматі %@%.% (де % — рядок, що складається не менш, ніж з одного символу, і не містить символів «;» та «,»);
- Медична довідка – непорожній рядок;
- Серія атестату – маска вводу: рядок з двох літер;
- Номер атестату – маска вводу: рядок з 8 цифр;
- Середній бал – число від 1 до 12, не більше однієї цифри після коми;
- Пароль – автоматично згенерований рядок з шести випадкових символів ASCII-таблиці з номерами від 48 до 122;
- ЗНО – список складених іспитів ЗНО: предмет, рік складання та набрана кількість балів відповідно (число від 100 до 200, не більше однієї цифри після коми).

В полі ЗНО предмет обирається з розкривного списку серед записів таблиці «Предмети». При виборі предмета автоматично додається ще один рядок для додавання нового тесту ЗНО, як це показано нижче:

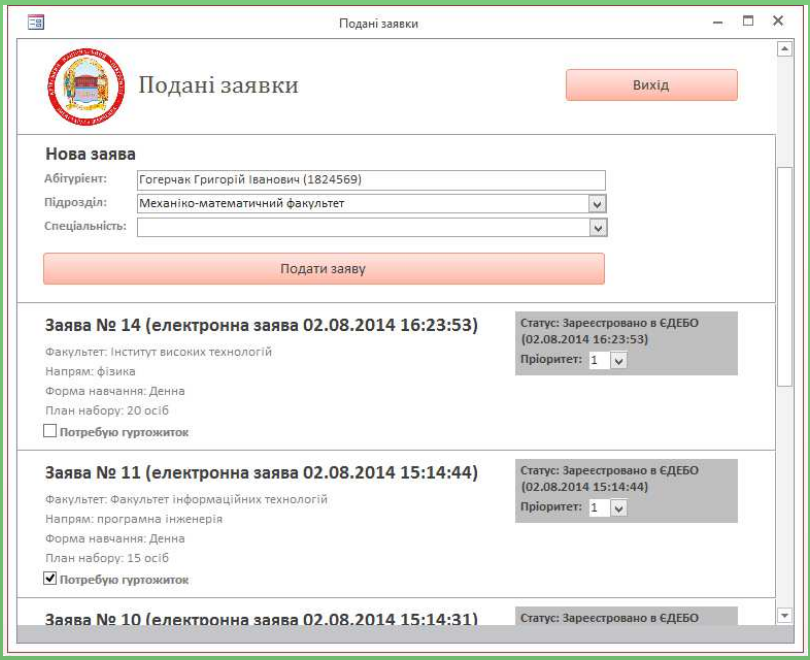


Усі поля, окрім ЗНО, є обов'язковими для заповнення. Причому кожен абітурієнт не може складати двічі тест з одного предмету впродовж одного року.

Натискання на кнопку «Зберегти» вносить дані про абітурієнта у відповідні таблиці бази даних, закриває вікно та відкриває вікно входу. Вікно входу містить окрім кнопок 2 текстових поля: номер сертифіката та пароль відповідно. Користувач заповнює ці поля відповідними даними та натискає «Увійти», після чого якщо в базі існує користувач з такою комбінацією «номер сертифіката-пароль», відбувається закриття вікна входу та перехід до вікна «Подані заяви», інакше виводиться повідомлення про помилку:

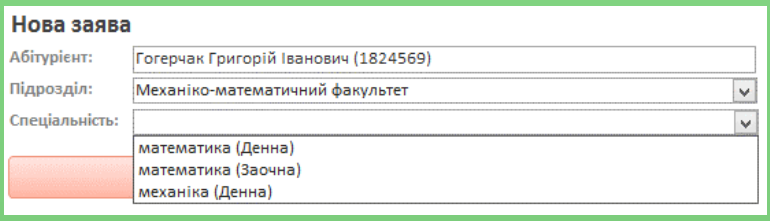


Вікно «Подані заяви» містить два блоки: подання нової заяви та список поданих заяв.

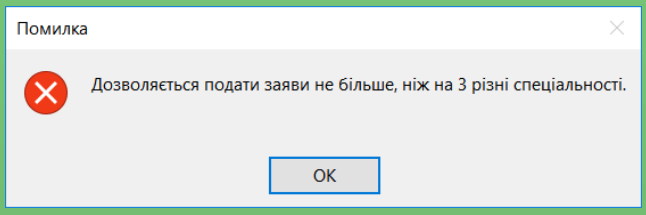


Блок «Нова заява» містить три поля:

- поле Абітурієнт, яке автоматично заповнюється даними поточного користувача (прізвище, ім'я, по-батькові та номер сертифікату ЗНО в дужках);
- поле Підрозділ з розкривним списком (за даними таблиці «Спеціальності», без повторень);
- поле Спеціальність з розкривним списком.

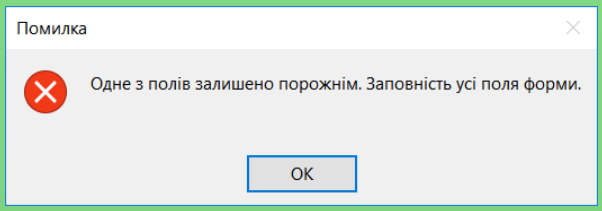


Значення розкривного списку поля Спеціальність залежать від обраного підрозділу. Спеціальності, на які вже подано заяви, а також ті, на які подання заяви неможливе у відповідності до комплекту складених користувачем тестів ЗНО (користувач має право подавати заяву на спеціальність, якщо ним складено усі обов'язкові для цієї спеціальності іспити ЗНО), у розкривному списку не відображаються. Якщо для певного підрозділу доступних спеціальностей немає, він також не відображається у розкривному списку поля Підрозділ.
Натискання на кнопку «Подати заяву» вносить відповідні дані до бази даних, якщо сукупна кількість різних спеціальностей (за назвою), на які абітурієнтом подані заяви, враховуючи поточну заяву, не перевищує трьох. Інакше виводиться відповідне повідомлення про помилку:



Наприклад, спеціальності «Програмна інженерія» на факультеті кібернетики та на факультеті інформаційних технологій – однакові в цьому розумінні.

Також помилка виводиться у разі, якщо принаймні одне з полів порожнє:



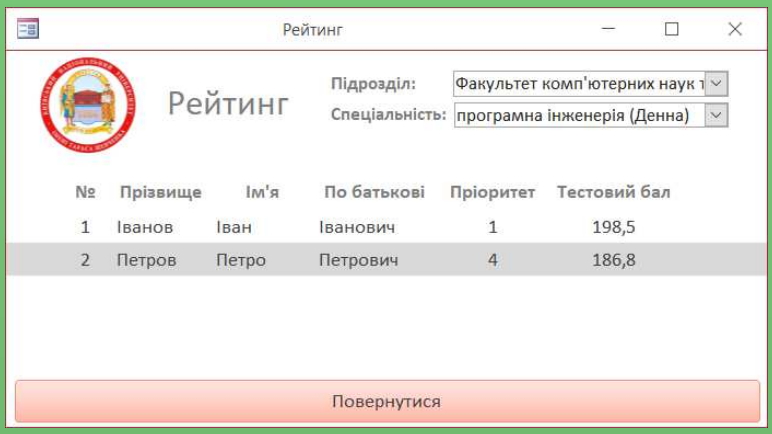
Після додавання заяви вона одразу з'являється в списку поданих заяв вікна «Подані заяви».

Блок «Подані заяви» містить список поданих користувачем заяв з відповідною інформацією (номер заяви, дата та час подання (вносяться до бази автоматично), факультет, напрям, форма навчання, план набору, потреба в гуртожитку та пріоритет спеціальності).

Клацання на прапорець «Потребую гуртожиток», призводить до зміни відповідних даних в базі. Розкривний список «Пріоритет» містить розкривний список з числами від 1 до 10. Вибір значення в списку призводить до зміни відповідних даних у базі.

Кнопка «Вихід» в заголовку форми призводить до закриття вікна та відкриття вікна входу з видаленням з системи даних про поточного користувача.

Кнопка «Рейтинг за спеціальностями» вікна входу закриває його та відкриває вікно з рейтингом абітурієнтів, список яких залежить від значень, обраних у списках підрозділів і спеціальностей:



Залежно від обраного в списку підрозділу змінюється перелік спеціальностей. Після вибору спеціальності перелік поданих заяв автоматично змінюється. В переліку містяться відомості про прізвище, ім'я, по-батькові абітурієнта, пріоритет заяви та тестовий бал, за спаданням якого список сортується. Також відображається порядковий номер абітурієнта в списку. Якщо кілька абітурієнтів мають однакову сумарну кількість балів, вони розміщуються в списку між собою в довільному порядку.
Якщо абітурієнт двічі здавав тест з одного й того ж предмету (в різні роки), при обчисленні суми враховується результат з найбільшою кількістю балів.
Тестовий бал розраховується за наступною формулою та округляється до одного знаку після десяткової коми:

$$T_1 * K_1 + T_2 * K_2 + T_3 * K_3 + 200 * \frac{A}{12} * (1 - K_1 - K_2 - K_3)$$

Тут T_i — результати ЗНО з трьох обов'язкових для даної спеціальності предметів, K_i — коефіцієнти відповідних предметів з таблиці *Спецпредмет*, а A — середній бал атестату.
Натискання на кнопку «Повернутися» закриває поточне вікно і відкриває вікно входу.

4.7 Відповіді та вказівки до розв'язання вправ

№123. Інженерія інформаційної системи розпочинається з проектування бази даних. В даному випадку вона є доволі простою. З умови завдання відомо, що система повинна зберігати дані про абітурієнтів, складені ними ЗНО та подані заяви на різні спеціальності. Наданий до завдання шаблон вже містить три інші таблиці, що зберігають дані про предмети, спеціальності та перелік предметів для кожної відповідної спеціальності.

Оскільки абітурієнт складає декілька ЗНО, то відповідні результати тесту потребують додаткової таблиці (адже об'єднання даних абітурієнта та складених ним ЗНО в одну таблицю суперечитиме третій нормальній формі). Загалом до шаблону слід додати три нові таблиці: *абітурієнти*, *ЗНО* та *заявки*.

Загалом між наявними в базі даних таблицями існуватимуть наступні зв'язки:

- «абітурієнт отримав результат ЗНО з предмету в певному році» — бінарний зв'язок множинністю «багато до багатьох», який розривається допоміжною таблицею «ЗНО» на два зв'язки «один до багатьох»; цей зв'язок реалізується знесенням первинних ключів в допоміжну таблицю з додаванням до неї атрибутів зв'язку (*рік* та *результат*); ключем допоміжної таблиці в даному випадку є комбінація полів *сертифікат абітурієнта*, *предмет* та *рік*, оскільки один і той самий абітурієнт не може складати ЗНО двічі з одного й того предмету в один рік;
- «абітурієнт отримав результат ЗНО з предмету в певному році» — теж бінарний зв'язок множинністю «багато до багатьох», який розривається допоміжною таблицею «заявки» на два зв'язки «один до багатьох»; цей зв'язок реалізується знесенням первинних ключів в допоміжну таблицю з додаванням до неї атрибутів зв'язку (*номер заяви* (автонумерація), *дата та час подання*, *пріоритет* та *гуртожиток*); ключем допоміжної таблиці в даному випадку є комбінація полів *сертифікат абітурієнта* та *спеціальність*, оскільки один і той самий абітурієнт не може подати документи двічі на одну й ту ж спеціальність.

У Вправі 51 ми вже побудували таблицю *абітурієнти* з відповідними полями та налаштували для них необхідні маски. Тепер додамо до цієї таблиці нове поле *пароль*, додамо дві нові таблиці та налаштуємо відповідні правила перевірки відповідно до нових вимог.

Аби забезпечити перевірку коректності електронної адреси, ми повинні забезпечити перевірку даних цього поля на відповідність наступній умові:

```
([email] LIKE "?*@?*.?*") AND
    ([email] NOT LIKE "*[;,]*")
```

При встановленні даного логічного виразу в якості правила перевірки (*Validation Rule*) поля *email* у разі некоректного введення електронної адреси користувачеві буде відображено повідомлення про помилку. Його можна задати власноруч за допомогою параметру *Validation Text*, хоча це не вимагається специфікацією системи.

Аби забезпечити перевірку середнього балу атестату, ми повинні здійснювати перевірку наступної логічної умови:

```
[середній бал атестату] BETWEEN 1 AND 12
```

Аналогічно, для кількості балів тесту ЗНО правило перевірки матиме наступний вигляд:

```
[результат] BETWEEN 100 AND 200
```

Для автоматичної генерації паролю ще на рівні таблиці, задамо для поля пароль властивість *Default Value (Значення за промовчанням)* за допомогою наступного виразу:

```
= Chr(Int(Rnd()*75) + 48) &
  Chr(Int(Rnd()*75) + 48) &
  Chr(Int(Rnd()*75) + 48) &
  Chr(Int(Rnd()*75) + 48) &
  Chr(Int(Rnd()*75) + 48) &
  Chr(Int(Rnd()*75) + 48)
```

Кожен з шести символів генерується випадковим чином однаково: за допомогою вбудованої функції *Rnd* ми отримуємо випадкове значення в діапазоні $[0,1)$; після множення числа на 75 ми отримуємо число в діапазоні $[0,75)$, а після взяття його цілої частини за допомогою функції *Int* — в діапазоні $[0,74]$; додавання числа 48 дає нам число в діапазоні $[48,122)$, яке ми передаємо в функцію *Chr*, що повертає символ за його кодом.

Для полегшення подальшої роботи, одразу налаштуємо розкривний список для поля *предмет* таблиці *ЗНО*. Це можна зробити за допомогою Майстра підстановок або за допомогою встановлення відповідних властивостей поля.

Джерелом рядків цього списку буде наступний запит:

```
SELECT Предмети.*
FROM Предмети;
```

Оскільки ключем таблиці *Предмети* є ідентифікатор предмету, а з зображень системи ми бачимо, що в списку слід обирати його назву, необхідно приховати поле ключа за допомогою встановлення нульової його ширини:

General | Lookup

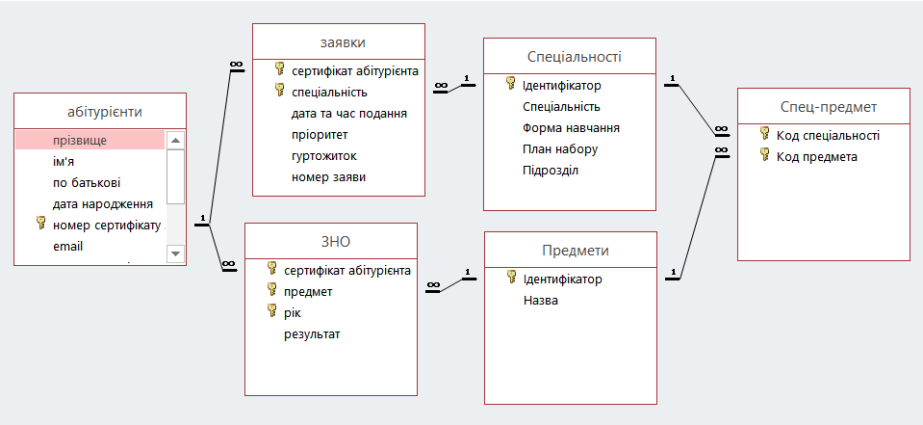
Display Control	Combo Box
Row Source Type	Table/Query
Row Source	SELECT Предмети.* FROM Предмети;
Bound Column	1
Column Count	2
Column Heads	No
Column Widths	0cm;5cm
List Rows	16
List Width	Auto
Limit To List	Yes
Allow Multiple Values	No
Allow Value List Edits	Yes
List Items Edit Form	
Show Only Row Source Valu	No

Слід зазначити, що в таблиці *заявки* для поля *пріоритет* теж можна задати розкривний список зі статичного переліку чисел від 1 до 10. Для цього слід на вкладці *Lookup (Підстановка)* властивостей поля обрати *Value List (Список значень)* в якості значення параметру *Row Source Type (Тип джерела рядків)* та задати перелік значень в якості значення параметру *Row Source (Джерело рядків)*:

General | Lookup

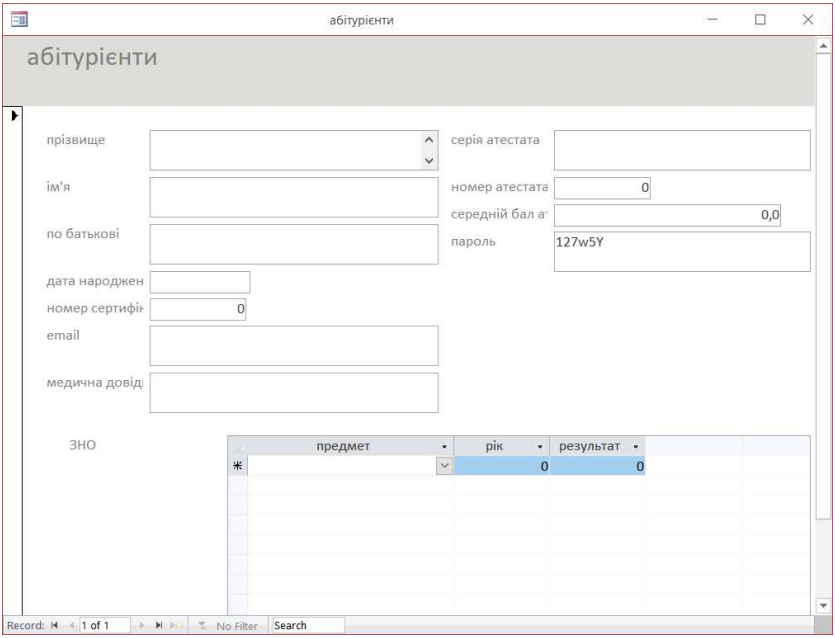
Display Control	Combo Box
Row Source Type	Value List
Row Source	"1";"2";"3";"4";"5";"6";"7";"8";"9";"10"
Bound Column	1
Column Count	1
Column Heads	No
Column Widths	
List Rows	16
List Width	Auto
Limit To List	No
Allow Multiple Values	No
Allow Value List Edits	Yes
List Items Edit Form	
Show Only Row Source Valu	No

Налаштування зв'язків є доволі простим, оскільки усі вони мають множинність «один до багатьох». Після цього схема даних повинна мати наступний вигляд:



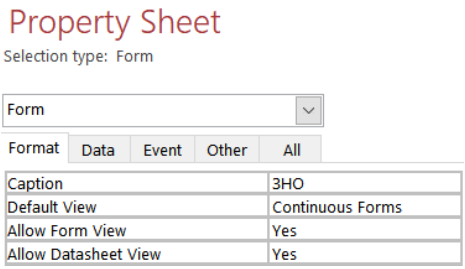
Після налаштування зв'язків між таблицями ми можемо створити форму реєстрації за допомогою майстра форм, обравши усі поля таблиці *абітурієнти* та поля *предмет*, *рік* та *результат* таблиці *ЗНО*.

Ми отримаємо форму наступного вигляду:

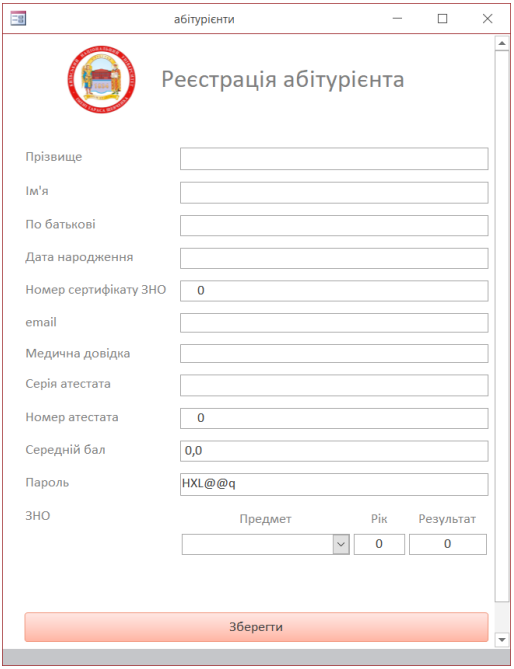


Аби підформа з результатами ЗНО стала формою на декілька елементів

замість табличної, слід встановити на аркуші властивостей цієї підформи значення *Continuous Forms (Безперервні форми)* властивості *Default View (Подання за промовчанням)*:



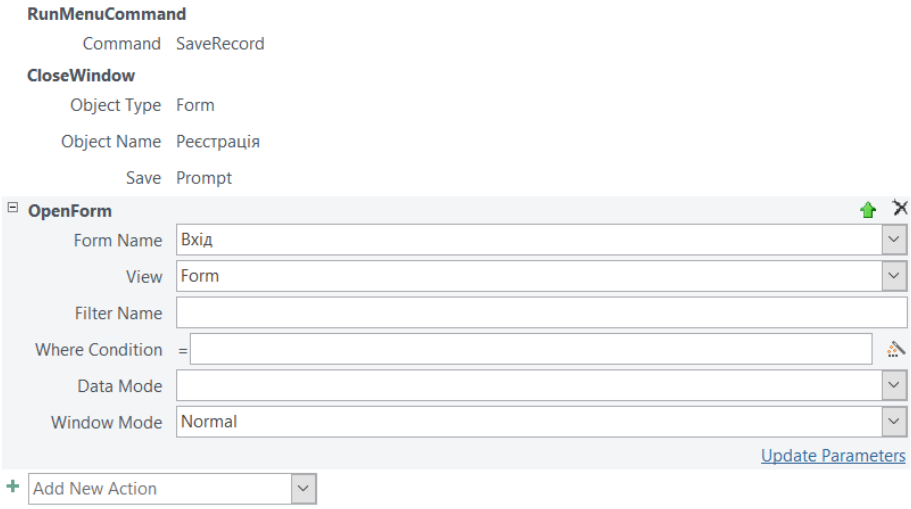
Після цих дій, додавання кнопки та оформлення форми відповідно до зразка за допомогою параметрів вкладки *Format (Формат)* аркуша властивостей її елементів, ми отримаємо форму реєстрації відповідно до зразка:



Можна також заборонити редагування поля пароль, встановивши значення *Yes (Так)* для параметра *Locked (Заблоковане)*.

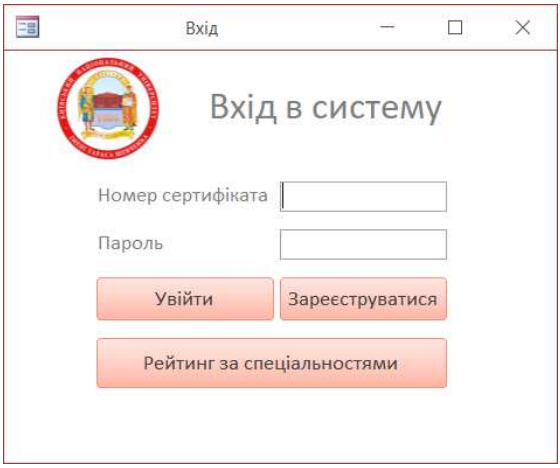
Оскільки кнопка Зберегти повинна зберігати поточний запис в таблицю, нам слід скористатися макрокомандою *SaveRecord (ЗберегтиЗапис, СохранитьЗапись)* категорії *Data Entry Operations*. Також для закриття поточної форми та відкриття форми входу нам знадобляться команди *CloseWindow (ЗакритиВікно, ЗакрытьОкно)* категорії *Window Management* та *OpenForm (ВідкритиФорму, ОткрытьФорму)* категорії *Database Objects*.

Створимо порожню форму для входу з назвою *Вхід* та побудуємо відповідний макрос:

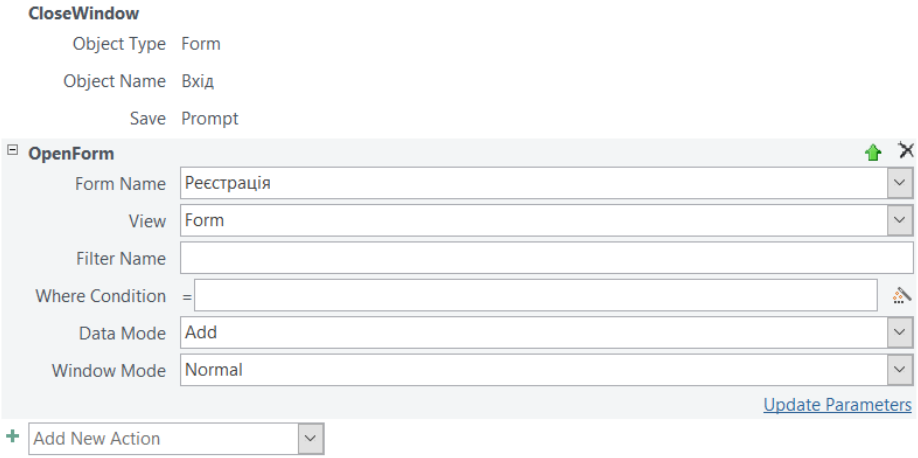


Для коректної роботи кнопки слід призначити створений макрос обробником події *On Click (Після клацання)*.

Перейдемо до побудови форми входу в систему. Для початку додамо на форму вільні текстові поля (з іменами *сертифікат* та *пароль* відповідно), написи та кнопки (наразі жодних дій для них ми не призначаємо) та здійснимо її оформлення відповідно до зразка:



Налаштуємо для початку роботу кнопки Зареєструватися. Вона повинна закривати вікно входу та відкривати форму реєстрації нового абітурієнта в режимі додавання записів. Для цього скористаємось макрокомандами *CloseWindow (ЗакритиВікно, ЗакрытьОкно)* категорії *Window Management* та *OpenForm (ВідкритиФорму, ОткрытьФорму)* категорії *Database Objects*:



Слід зауважити, що значення *Add (Додавання)* параметру *Data Mode (Режим даних)* забезпечує відкриття форми саме в режимі додавання нових записів.

Для коректної роботи кнопки слід призначити створений макрос обробником події *On Click (Після клацання)*.

Для зручності, зареєструємо кількох абітурієнтів аби мати можливість тестувати форму входу в майбутньому. Наприклад, додамо *Іванова Івана Івановича* з номером сертифікату *125813* та паролем *127w5Y*, а також Петрова Петра Петровича з номером сертифікату *564644* та паролем *ETd35k*.

Один із варіантів реалізації пошуку дуже схожий на те, як ми здійснювали обробку крайнього випадку, коли реалізовували пошук товарів. Створимо запит *Вхід*, який шукатиме серед абітурієнтів таких, що їх номер сертифікату та пароль зберігаються з введеними:

```
SELECT абітурієнти.*
FROM абітурієнти
WHERE абітурієнти.[номер сертифікату ЗНО] =
      Forms!Вхід!сертифікат AND
      абітурієнти.пароль = Forms!Вхід!пароль;
```

Якщо користувач ввів вірні дані, результат цього запиту міститиме рівно один запис, якщо хибні — жодного.

Перейдемо до створення макросу входу. За умовою, якщо користувач ввів вірну комбінацію сертифікату та паролю — здійснюється вхід і перехід до вікна поданих заяв із закриттям поточного вікна, а якщо невірну — відображається повідомлення. Отже, нам знадобиться структура умовного переходу.

Скористаємось вже знайомою нам умовою перевірки кількості записів в результаті запиту:

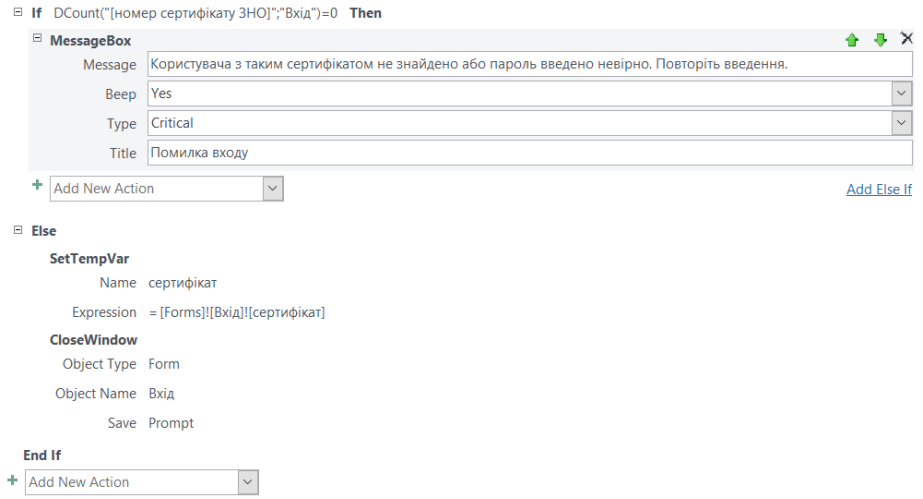
```
DCount("[номер сертифікату ЗНО]"; "Вхід") = 0
```

У разі істинності умови користувач ввів некоректні дані, тому в цій гілці слід скористатися макрокомандою *MessageBox (ВікноПовідомлення, ОкноСообщения)* категорії *User Interface Command* для відображення повідомлення про помилку. Слід зауважити, що піктограма повідомлення на зразку досягається за допомогою значення *Critial (Критичне)* параметру *Type (Тип)* цієї команди.

Гілка *Else (Інакше, Иначе)*, яку можна додати справа під блоком *If (Якщо, Если)*, міститиме послідовність дій, пов'язаних зі входом.

Наразі інформація про поточного користувача у разі успішного входу зберігається тільки на формі *Вхід*. Оскільки після входу вимагається її закриття, слід завчасно зберегти номер сертифікату у тимчасову змінну *сертифікат* і тільки після цього закривати форму входу.

Макрос без відкриття форми заявок матиме наступний вигляд:

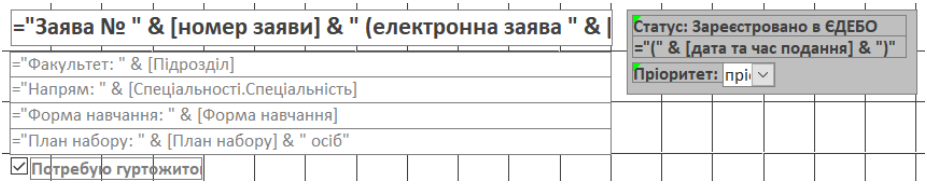


Наступна форма під назвою *Подані заявки* повинна відображати подані користувачем заявки та надавати можливість створення нових.

Побудуємо запит, що відображатиме заявки та пов'язані з ними дані для певного номеру сертифіката, який отримаємо з тимчасової змінної:

```
SELECT заявки.*, Спеціальності.*
FROM заявки INNER JOIN Спеціальності
    ON заявки.спеціальність =
        Спеціальності.Ідентифікатор
WHERE заявки.[сертифікат абітурієнта] =
    TempVars!сертифікат
ORDER BY заявки.[номер заяви] DESC;
```

На основі написаного запиту побудуємо форму на декілька елементів. Шляхом переміщення полів та їх форматування, а також використання виразів над полями джерела записів, нам слід досягти наступного результату в режимі конструктора:



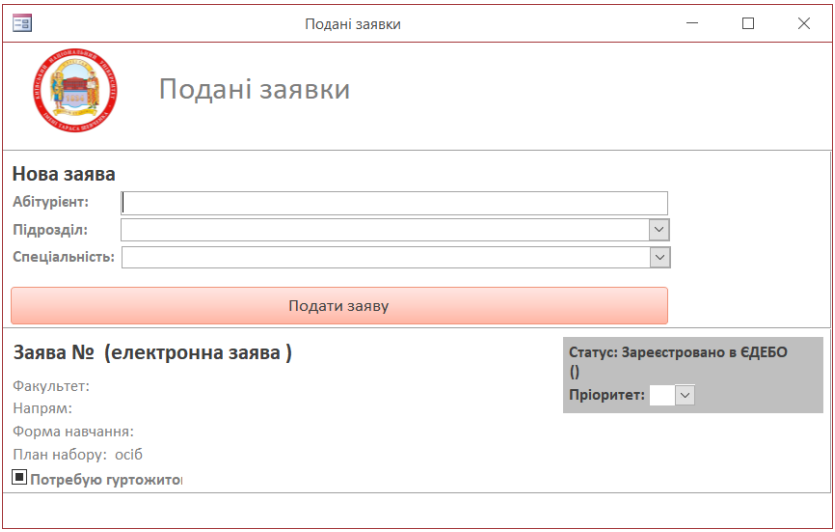
Слід зазначити, що деякі частини тут можна оформити й у вигляді написів без використання оператора конкатенації *&*, проте його використання значно полегшить вирівнювання тексту та його форматування, а також унеможливить редагування відповідних полів записів форми.

Прапорець *Потребую гуртожиток* та розкривний список з пріоритетами тут зв'язані з відповідними полями джерела записів.

Сіре забарвлення частині запису можна надати, скориставшись елементом керування *Rectangle (Прямокутник)*.

Як ми вже раніше зазначали, макрокоманда *SaveRecord (ЗберегтиЗапис, СохранитьЗапись)* категорії *Data Entry Operations* дозволяє зберегти в базі даних поточний запис. Тому, аби зміни в полях потреби в гуртожитку та пріоритету одразу відображалися в таблиці *абітурієнти*, слід наступний макрос призначити обробником події *After Update (Після оновлення)* для обох цих полів:

Перейдемо до реалізації блоку подання заявки. Додамо підписи, текстове поле, два порожніх розкривних списки (*підрозділ* та *спеціальність*) та кнопку. Здійснимо форматування форми відповідно до зразка:



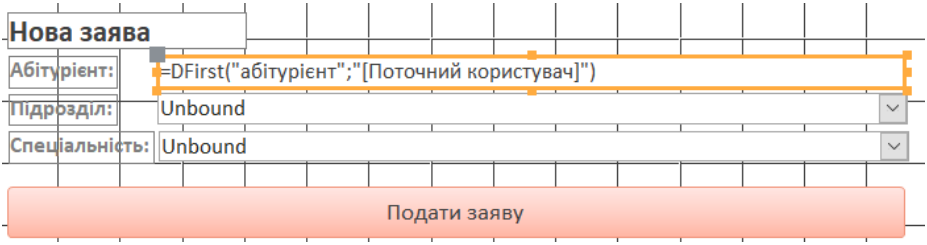
Аби окрім номеру сертифіката відобразити ще й прізвище, ім'я та по батькові поточного користувача, слід побудувати наступний запит:

```
SELECT абітурієнти.прізвище & "␣" &
       абітурієнти.[ім'я] & "␣" &
       абітурієнти.[по батькові] & "␣(" &
       абітурієнти.[номер сертифікату ЗНО] & ")"
           AS абітурієнт
FROM абітурієнти
WHERE абітурієнти.[номер сертифікату ЗНО] =
       TempVars!сертифікат;
```

Цей запит завжди повертатиме один запис з відомостями про поточного користувача. Аби відобразити обчислене значення в полі форми скористаємось функцією DFirst, яка повертає значення відповідного поля першого запису відповідного об'єкта (таблиці чи запиту):

```
DFirst("<назва поля>"; "<назва таблиці/запиту>")
```

В нашому випадку нам потрібне значення поля *абітурієнт* єдиного запису в запиті *Поточний користувач*:



Аби в розкривному списку *підрозділ* користувач міг обирати підрозділ зі списку без повторень, в якості його джерела рядків вкажемо наступний запит:

```
SELECT DISTINCT Спеціальності.Підрозділ
FROM Спеціальності;
```

Розкривний список *спеціальність* реалізовується трохи складніше. В переліку тут потрібно відобразити спеціальності обраного підрозділу, на які ще не подано заяви поточним користувачем і на які достатньо предметів, з яких склав ЗНО користувач.

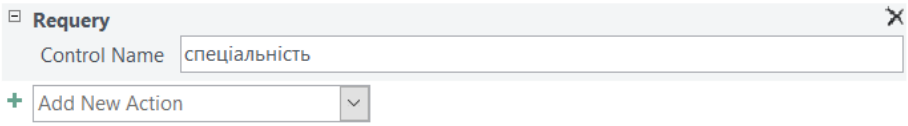
Остання умова вказує, що запит використовує серед іншого множинне порівняння. Переформулюємо останню умову так: «не існує предмету для

цієї спеціальності, що з нього поточний користувач не складав ЗНО». Тоді відповідний запит матиме наступний вигляд:

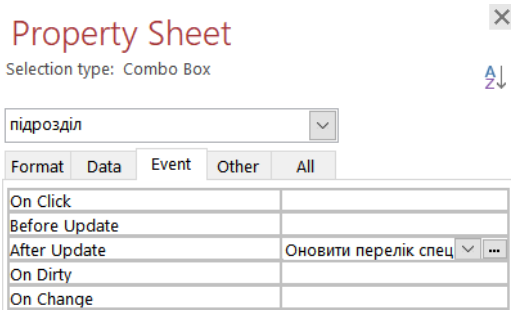
```
SELECT Спеціальності.Ідентифікатор,
    Спеціальності.Спеціальність & "␣(" &
        Спеціальності.[Форма навчання] & ")"
            AS Спеціальність
FROM Спеціальності
WHERE Спеціальності.Підрозділ =
    Forms![Подані заявки]!підрозділ AND
    Спеціальності.Ідентифікатор NOT IN (
        SELECT заявки.спеціальність
        FROM заявки
        WHERE заявки.[сертифікат абітурієнта] =
            TempVars!сертифікат
    ) AND NOT EXISTS (
        SELECT [Спец-предмет].[Код предмета]
        FROM [Спец-предмет]
        WHERE [Спец-предмет].[Код спеціальності] =
            Спеціальності.Ідентифікатор AND
            [Спец-предмет].[Код предмета] NOT IN (
                SELECT ЗНО.предмет
                FROM ЗНО
                WHERE ЗНО.[сертифікат абітурієнта] =
                    TempVars!сертифікат
            )
    );
```

Як ми вже розглядали, аби приховати ідентифікатор спеціальності, слід встановити для нього на вкладці *Format (Формат)* ширину в нуль.

Якщо спробувати зараз змінити підрозділ на формі та відкрити список спеціальностей, ми пересвідчимося, що до жодних змін це не призводить. Справді, ми вже зазначали раніше, що Microsoft Access не оновлює автоматично результати запитів. Якщо зараз натиснути F5 після вибору підрозділу зі списку, то результат відповідатиме нашим очікуванням — список спеціальностей зміниться. Ми вже зазначали, що допомагає в симуляції такого натискання макрокоманда *Requery (ПовторитиЗапит, ПовторитьЗапрос)* категорії *Filter/Query/Search*, проте цього разу слід оновити не всю форму, а лише конкретний елемент керування:



Тепер макрос повинен бути обробником іншої події, адже оновлення повинно відбуватися одразу після вибору елемента у списку підрозділів. За такий випадок відповідає подія *After Update (Після оновлення)*:



Тепер, коли користувач не може обрати спеціальність, на яку не має права подавати заявку, ми можемо реалізувати збереження заявки.

Оскільки форма одночасно і створює, і відображає записи, реалізувати додавання так, як ми це зробити у формі реєстрації, тут не вдасться. Поля, за допомогою яких збирається інформація для створення заявки, є вільними, а отже додавати запис ми можемо лише за допомогою запиту *INSERT*, використовуючи функцію *Now* для отримання поточного часу, значення тимчасової змінної *сертифікат* та поля *спеціальність*:

```
INSERT INTO заявки
    ([сертифікат абітурієнта], спеціальність,
        [дата та час подання], пріоритет)
VALUES (TempVars!сертифікат,
    Forms![Подані заявки]!спеціальність,
    Now(), 1);
```

Але за специфікацією системи заявка повинна прийматися лише в тому випадку, якщо кількість різних спеціальностей, на які (разом з поточною заявою) подав заяви поточний користувач, не перевищує трьох.

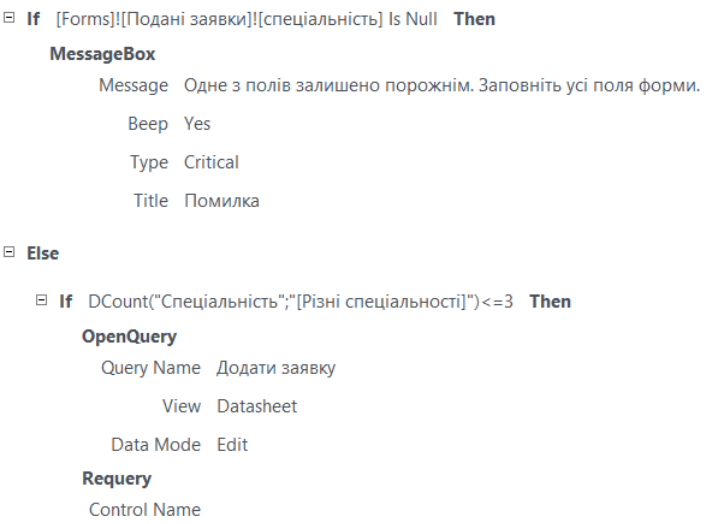
Запишемо запит *Різні спеціальності*, який визначатиме перелік різних спеціальностей, на які користувач подав заяви, включаючи поточну. Нам необхідно визначити без повторень назви спеціальностей, на які вже подано заяви, або рівних обраній спеціальності:

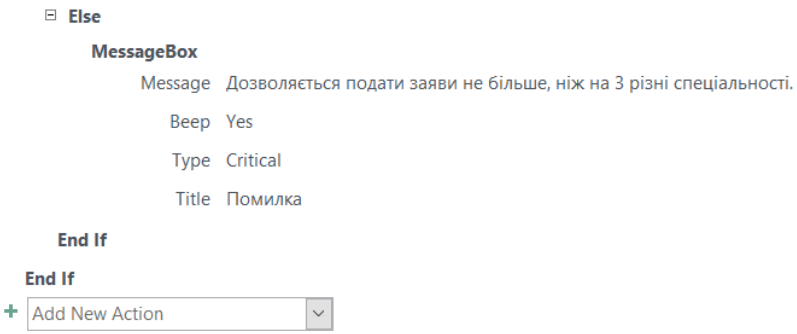
```
SELECT DISTINCT Спеціальності.Спеціальність
FROM Спеціальності
WHERE Спеціальності.Ідентифікатор =
    Forms![Подані заявки]!спеціальність OR
    Спеціальності.Ідентифікатор IN (
        SELECT заявки.спеціальність
        FROM заявки
        WHERE заявки.[сертифікат абітурієнта] =
            TempVars!сертифікат
    );
```

Тоді необхідна умова для подання заявки матиме наступний вигляд:

```
DCount("Спеціальність"; "[Різні спеціальності]") <= 3
```

З урахуванням додаткової перевірки заповнення полів, макрос матиме наступний вигляд:

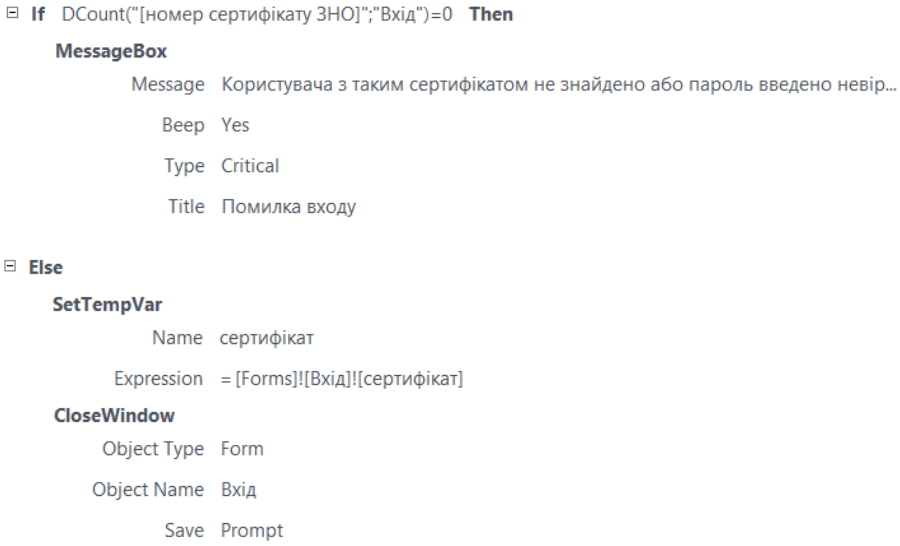


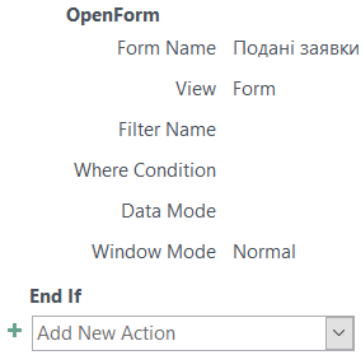


У разі виконання усіх умов вище виконується макрокоманда *Requery (ПовторитиЗапит, ПовторитьЗапрос)* категорії *Filter/Query/Search*, яка відкриває запит на додавання запису, а також здійснюється оновлення даних форми (інакше доданий запис не відобразиться автоматично).

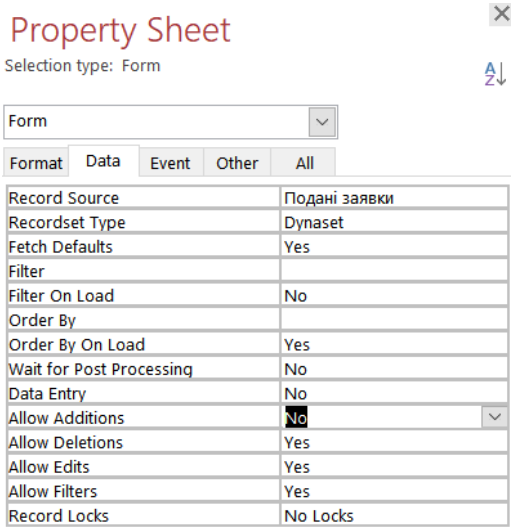
Створений макрос слід призначити обробником події *On Click (Після клацання)* кнопки *Подати заяву*.

Оскільки форму Подані заяви реалізовано, ми можемо повернутися до макросу здійснення входу та додати команду для відкриття форми:





Для того, щоб прибрати в формі зайвий запис для додавання заявки, на аркуші властивостей форми слід встановити значення *No (Ні)* для параметра *Allow Additions (Дозволити додавання)*:



На цій формі залишається тільки додати кнопку *Вихід*, обробник події *On Click (Після клацання)* для неї матиме наступний вигляд:

Перейдемо до створення форми *Рейтинг*.

Створимо запит *Повний рейтинг*, який розраховує тестові бали абітурієнтів для кожної їх заяви. Для кожної заявки візьмемо дані абітурієнта та спеціальності. Взявши предмети, необхідні для вступу на спеціальність, та предмети, з яких абітурієнти складали ЗНО, залишимо тільки ті записи, в яких ці назви предметів співпадають. Таким чином для кожного предмету спеціальності ми отримаємо результат, який з цього предмету отримав абітурієнт. Тепер порахуємо суму добутків цих результатів та відповідних їм коефіцієнтів, як того вимагає формула в умові, а також додамо атестат із залишковим коефіцієнтом.

Слід зазначити, що аби врахувати умову, що у разі повторної здачі тесту враховується кращий результат, таблицю ЗНО в цих зв'язках слід замінити таблицею, що у відповідність абітурієнтові та предмету ставитиме кращих результат:

```
SELECT ЗНО.[сертифікат абітурієнта],
       ЗНО.предмет,
       MAX(ЗНО.результат) AS результат
FROM ЗНО
GROUP BY ЗНО.[сертифікат абітурієнта], ЗНО.предмет;
```

Остаточний код запиту *Повний рейтинг* наступний:

```
SELECT FIRST(абітурієнти.прізвище) AS прізвище,
       FIRST(абітурієнти.[ім'я]) AS [ім'я],
       FIRST(абітурієнти.[по батькові]) AS [по батькові],
       FIRST(заявки.пріоритет) AS пріоритет,
       Round(
            Sum(ЗНО.результат *
                [Спец-предмет].коефіцієнт
            ) + (1 - Sum([Спец-предмет].коефіцієнт)) *
```

```
            FIRST(абітурієнти.[середній бал атестату]) *
            200/12, 1
        ) AS бал,
        заявки.[сертифікат абітурієнта],
        заявки.спеціальність
FROM (
            (
                (
                    Спеціальності INNER JOIN заявки
                    ON Спеціальності.Ідентифікатор =
                        заявки.спеціальність
                ) INNER JOIN абітурієнти
                ON абітурієнти.[номер сертифікату ЗНО] =
                    заявки.[сертифікат абітурієнта]
            ) INNER JOIN (
                SELECT ЗНО.[сертифікат абітурієнта],
                    ЗНО.предмет,
                    MAX(ЗНО.результат) AS результат
                FROM ЗНО
                GROUP BY ЗНО.[сертифікат абітурієнта],
                    ЗНО.предмет
            ) AS ЗНО
            ON абітурієнти.[номер сертифікату ЗНО] =
                ЗНО.[сертифікат абітурієнта]
        ) INNER JOIN [Спец-предмет]
        ON ЗНО.предмет = [Спец-предмет].[Код предмета]
WHERE Спеціальності.Ідентифікатор =
        [Спец-предмет].[Код спеціальності]
GROUP BY заявки.[сертифікат абітурієнта],
        заявки.спеціальність;
```

Примітка. Для кращого розуміння змісту запиту відкрийте його в режимі конструктора.

На основі створеного запиту побудуємо форму на декілька елементів та відформатуємо її у відповідності до зразка в умові. Додамо на цю форму два розкривних списки відповідно до зразка — *підрозділ* та *спеціальність*.

Як і раніше, аби в розкривному списку *підрозділ* користувач міг обирати підрозділ зі списку без повторень, в якості його джерела рядків вкажемо наступний запит:

```
SELECT DISTINCT Спеціальності.Підрозділ
```

```
FROM Спеціальності;
```

Розкривний список *спеціальність* цього разу складається з усіх спеціальностей обраного підрозділу, тож джерело рядків матиме наступний вигляд:

```
SELECT Спеціальності.Ідентифікатор,
    Спеціальності.Спеціальність & " (" &
        Спеціальності.[Форма навчання] & ")"
            AS Спеціальність
FROM Спеціальності
WHERE Спеціальності.Підрозділ =
    Forms!Рейтинг!підрозділ;
```

Як і раніше, аби приховати ідентифікатор спеціальності, слід встановити для нього на вкладці *Format (Формат)* ширину в нуль.

Для оновлення переліку після зміни підрозділу ми можемо скористатися створеним раніше макросом *Оновити перелік спеціальностей*.

Маючи повний рейтинг для усіх спеціальностей та відповідні поля, побудуємо запит, який формує рейтинг обраної спеціальності:

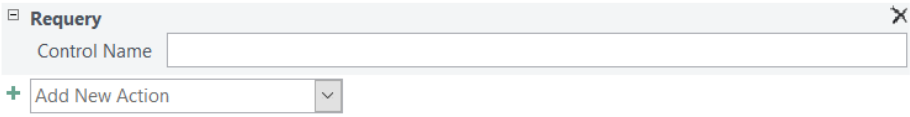
```
SELECT [Повний рейтинг].*
FROM [Повний рейтинг]
WHERE [Повний рейтинг].спеціальність =
    Forms!Рейтинг!спеціальність;
```

Згадаємо, що номер абітурієнта за спаданням тестового балу — це кількість абітурієнтів тієї спеціальності, що мають вищу або рівну кількість балів. Тому остаточне джерело записів для нашої форми рейтингу матиме наступний вигляд:

```
SELECT (SELECT COUNT(P.[сертифікат абітурієнта])
        FROM [Повний рейтинг] AS P
        WHERE P.бал >= [Повний рейтинг].бал AND
            P.спеціальність =
                Forms!Рейтинг!спеціальність
    ) AS N, [Повний рейтинг].*
FROM [Повний рейтинг]
WHERE [Повний рейтинг].спеціальність =
    Forms!Рейтинг!спеціальність
ORDER BY [Повний рейтинг].бал DESC;
```

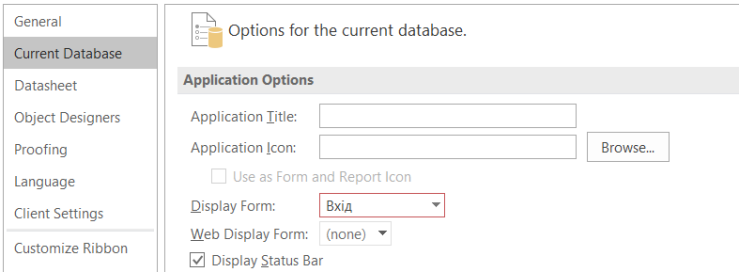
Залишається тільки замінити джерело записів форми *Рейтинг* на вказаний вище запит.

Аби оновлення даних форми відбувалося автоматично, слід вказати в якості обробника події *After Update (Після оновлення)* наступний макрос:



Кнопка *Повернутися* повинна закривати поточну форму та відкривати форму входу, а тому обробник події *On Click (Після клацання)* для неї матиме наступний вигляд:

Нарешті, залишається налаштувати відкриття форми *Вхід* при відкритті бази даних. Для цього слід перейти до меню *File (Файл)* — *Options (Параметри)* — *Current Database (Поточна база даних)* та обрати цю форму у розкривному списку параметру *Display Form (Форма відображення)*:



В цьому ж меню за допомогою параметру *Document Window Options (Параметри вікна)* можна налаштувати перекриття вікон (*Overlapping Windows*) замість відображення у вигляді вкладок (*Tabbed Document*).

Розділ 5

Огляд принципів створення промислових інформаційних систем

Під час кожної нашої подорожі мережею Інтернет ми маємо справу з масою інформаційних систем: пошуковою системою Google, соціальними мережами Facebook та Instagram, відеопорталом Youtube тощо. Якщо раніше доступ до мережі Інтернет був радше предметом розкоші, то в сучасному світі Інтернет є елементом нашого повсякденного життя.

Так, за даними спільного досліження компаній *We Are Social* та *Hootsuite* більше 4 мільярдів населення планети є користувачами мережі Інтернет, а пересічний Інтернет-користувач наразі проводить близько шести годин, використовуючи під'єднанні до глобальної мережі пристрої та сервіси, — практично третину часу, який ми проводимо не уві сні.

Така популярність цього засобу зв'язку призвела до зростання кількості сервісів глобальної мережі. Свої послуги масово переводять в електронний формат магазини, банки та навіть державні установи, розширюючи таким чином потенційну частину населення, яку такі послуги можуть охопити.

Оскільки кількість користувачів мережі щорічно зростає, зростає й навантаження, яке доводиться приймати на себе великим інформаційним системам, а отже й посилюються вимоги до технічних характеристик цих ресурсів. Наразі практично кожна інформаційна система повинна швидко реагувати на запити користувачів, надійно зберігати їх дані, в тому числі зберігаючи резервні копії в різних точках світу, а також захищаючи інформацію

від несанкціонованого доступу, дозволяти оперативне доповнення функціоналу тощо.

Оскільки ядром інформаційної системи є її база даних, то аналогічні вимоги стосуються і систем управління базами даних та власне самих баз даних, адже робота з даними на жорсткому диску наразі є найвужчим місцем практично будь-якого програмного продукту. Що швидше на запит до бази даних буде отримано відповідь, тим швидше користувач отримає потрібну йому інформацію. А тому коректна організація даних є надважливою частиною подальшого успіху інформаційної системи, що розробляється.

Проте не тільки організація даних впливає на якість програмного продукту. З огляду на те, що сфера комп'ютерної розробки стає дедалі більш конкурентною, важливим ресурсом розробника є час, що минає від появи ідеї системи до повної її імплементації: що довшою є розробка інформаційної системи, тим більшою є ймовірність, що обрану нішу за цей час займуть конкуренти.

Саме тому важливим етапом у розробці програмного забезпечення є проектування його архітектури.

5.1 Клієнт-серверна архітектура

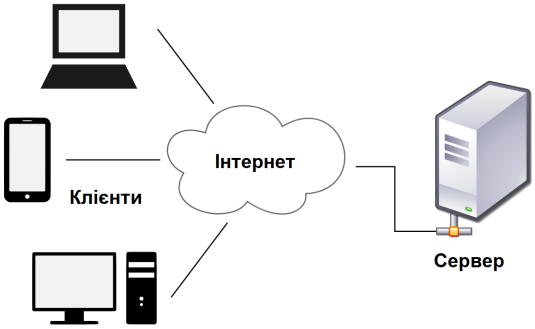


Рис. 5.1: Клієнт-серверна архітектура

Найбільш поширеним архітектурним шаблоном розподілених програмних систем є архітектура «клієнт-сервер», яка передбачає набір серверів, що зберігають та надають певну інформацію або послуги за запитом, набір клієнтів, що передають інформацію серверам та отримують її від них за допомогою запитів, а також мережу, яка забезпечує зв'язок між цими ком-

понентами та надійну передачу інформації (Рисунок 5.1). Тут під поняттями сервера та клієнта маються на увазі програмні модулі, що відповідають за ту чи іншу роль.

В межах даної архітектури клієнти та сервери працюють незалежно одне від одного: клієнт може цілком адекватно працювати навіть якщо всі сервери будуть неактивними, хоча й не отримуватиме від них жодної інформації.

Найяскравішим прикладом систем клієнт-серверної архітектури є усі веб-сайти мережі Інтернет. Клієнтами в даному випадку виступають спеціальні програми — браузери, які здійснюють HTTP-запити до веб-серверів за допомогою спеціального рядка — уніфікованого локатора ресурсів (URL), а сервери, після відповідної обробки отриманого запиту, надсилають HTTP-відповіді, що зазвичай містять HTML-сторінку, зображення, файл, медіапотік чи інші дані.

Серверну частину таких систем зазвичай створюють за допомогою об'єктно-орієнтованих та скриптових мов програмування на кшталт PHP, Python, Ruby, C++, Java тощо. Її основна мета — робота з інформацією. Клієнтська ж частина представляє собою набір гіпертекстових документів (HTML) разом з каскадною таблицею стилів (CSS) та скриптами, написаними мовою програмування JavaScript. Основна ціль цієї частини — представлення даних, збережених на сервері, та надання інтерфейсу до маніпуляції ними.

Загалом існує два підходи до побудови веб-орієнтованих систем: коли сервер повертає у відповідь на запит готову HTML-сторінку, а також WebAPI-підхід (Application Programming Interface з англійської — прикладний програмний інтерфейс), коли сервер передає клієнту лише дані у одному зі стандартних форматів (наприклад, XML чи JSON), а клієнт вже формує відповідне їх подання у вигляді HTML-сторінок.

5.2 Розробка веб-орієнтованих інформаційних систем на прикладі мови програмування Python

Python є високорівневою динамічно типізованою мовою програмування з відкритим кодом, що використовується для побудови як автономних програм, так і скриптових застосунків для широкого кола сфер використання. Наразі є лідером серед мов програмування за версією *IEEE Spectrum*.

Python є мовою, свідомо оптимізованою для швидкої розробки: простий синтаксис, динамічна типізація, відсутність процесу компіляції та вбудована

бібліотека стандартних рішень — все це дозволяє значно зменшити час розробки програмного продукту, а, отже, залишити більше часу для його відлагодження та вдосконалення. Також слід зазначити, що більшість програм, написаних цією мовою, виконуються однаково на усіх основних комп'ютерних платформах.

До значних переваг мови програмування Python належить використання системи управління пакетами pip для встановлення та деінсталяції зовнішніх залежностей. Цей засіб надає можливість для встановлення великої кількості сторонніх бібліотек, які дозволяють сконцентрувати процес розробки на елементах, що є особливими для конкретного розроблюваного продукту.

Ми зупинимо увагу оглядово на основних елементах процесу розробки інформаційних систем цією мовою. В межах цієї книги ми ставимо завдання не ґрунтовно розглянути процес розробки систем цією мовою програмування, проте дати читачеві розуміння щодо поточного стану справ для більш продуктивного вивчення цієї області знань в майбутньому. З синтаксисом мови Python, за потреби, ви можете познайомитися в довідці, доступній на її офіційному сайті.

5.2.1 Робота з базою даних

На відміну від інформаційних систем в середовищі Microsoft Access, промислові системи відділені від бази даних, часто навіть фізично розташовані на різних пристроях, а тому виникає необхідність встановлення з'єднання з базою даних.

Для з'єднання з базою даних за допомогою бібліотеки *MySQLdb* необхідно виконати наступний код:

```
import MySQLdb
db = MySQLdb.connect(host='<хост>',
                     user='<ім'я користувача>',
                     passwd='<пароль>',
                     db='<назва бази даних>',
                     charset='utf8')
```

Тут хост — це IP-адреса чи домен пристрою, на якому розміщено систему управління базами даних.

У разі успішного з'єднання, ми зможемо отримати курсор — поіменовану область пам'яті для здійснення запитів до бази даних та збереження результату за допомогою команди:

```
cursor = db.cursor()
```

Після цього будь-який запит до бази даних матиме наступний вигляд:

```
cursor.execute('<SQL-запит на вибірку>')
```

Після виконання цієї операції в курсорі зберігатиметься результат виконання запиту, який можна порядково відобразити за допомогою методу *fetchone()*, або ж отримати усі записи результату за допомогою методу *fetchall()*.

Для закриття з'єднання з базою даних слід скористатися командою:

```
db.close()
```

Наприклад, для відображення всіх товарів з бази даних супермаркетів, слід виконати наступний скрипт:

```
import MySQLdb
if __name__ == '__main__':
    db = MySQLdb.connect(host='localhost',
                         user='root',
                         passwd='',
                         db='supermarkets',
                         charset='utf8')
    cursor = db.cursor()
    cursor.execute('SELECT * FROM goods')
    for row in cursor.fetchall():
        print(row)
    db.close()
```

Результатом виконання цього коду буде наступний вивід:

```
(245558543, 'Батон білий', 14.3, 3)
(654646878, 'Молоко українське', 28.3, 2)
(654687269, 'Сир голандський', 231.1, 2)
(821556286, 'Хліб білий', 12.8, 1)
(863543436, 'Сирок плавлений', 13.2, 2)
(989651445, 'Хліб житній', 10.3, 1)
```

Параметризовані запити, як-от запит для визначення товарів за кодом їх виробника, *не варто* здійснювати за допомогою конкатенації рядків, як це показано нижче:

```
cursor.execute(
    'SELECT * FROM goods ' +
    'WHERE manufacturer_code = ' +
    code
)
```

Уявімо, що значення змінної *code* заповнюється з певної форми користувачем системи. Користувач може ввести в це поле текст «1 UNION SELECT user(), NULL, NULL, NULL» і отримати у останньому записі виводу ім'я користувача бази даних:

```
('821556286', 'Хліб білий', 12.8, 1)
('989651445', 'Хліб житній', 10.3, 1)
('root@localhost', None, None, None)
```

Адже буде виконано такий запит до бази даних:

```
SELECT *
FROM goods
WHERE manufacturer_code = 1
UNION
SELECT user(), NULL, NULL, NULL
```

Це лише один із варіантів отримання несанкціонованого доступу при такому записі запитів, один із найменш небезпечних. Загалом клас таких способів злому систем має назву SQL-ін'єкції.

Означення 52. *SQL-ін'єкція — це один із найпоширеніших способів злому програмних систем, що базується на впровадженні в запит до бази даних довільного SQL-коду.*

Аби запобігти SQL-ін'єкціям, слід усі подібні запити оформлювати як параметризовані наступним чином:

```
cursor.execute(
    'SELECT * FROM goods ' +
    'WHERE manufacturer_code = %s',
    (code,)
)
```

Тут усі входження параметрів замінюються позначкою *%s*, а згодом передаються кортежем як другий параметр методу *execute()*. В цьому випадку бібліотека власноруч відфільтровуватиме подібні ін'єкції.

Робота з базами даних в подібному ключі є зручною для невеликих програм, проте для промислових систем написання SQL-коду таким чином є джерелом помилок. А оскільки запитів в таких системах доволі багато, то й кількість помилок теж зростає значним чином, що впливає на час розробки програмного продукту.

5.2.2 Об'єктно-реляційне відображення

У об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Наприклад, виробника можна представити як список товарів з даними про них: назвами, цінами, штрих-кодами тощо. В термінах ООП товари представлятимуться об'єктами класу «Товар», що складатиметься з вищевказаних полів. З іншого боку, реляційні бази даних можуть зберігати і обробляти лише скалярні значення, збережені в таблицях.

Оскільки використання виключно скалярних даних є очевидно незручним та обмежуючим рішенням, необхідно вирішити проблему перетворення об'єктів у реляційну форму та навпаки зі збереженням їх властивостей та відношень між ними.

ORM (об'єктно-реляційне відображення) — це технологія програмування для перетворення даних між несумісними системами типів в об'єктно-орієнтованих мовах програмування.

ORM позбавляє розробника програмного забезпечення від необхідності власноруч реалізовувати роботу з базою даних. Оскільки ця робота є громіздкою та концентрує зазвичай велику кількість помилок, розробка систем значно пришвидшується, а їх код є більш інтуїтивно зрозумілим та підлягає подальшому використанню, коригуванню та доповненню. Крім того, переважна більшість бібліотек з підтримкою ORM дозволяє здійснювати запити за користувацьким SQL-кодом, а отже, не обмежують розробника у використанні переваг цієї мови запитів.

5.2.3 Фреймворк Django

Побудова веб-орієнтованої системи завжди потребує схожого набору компонент, як-от управління користувачами, додавання, оновлення та видалення об'єктів, доступ до них з панелі керування тощо. Оскільки ці елементи функціональності програмних систем є зазвичай стандартними, необхідність їх імплементації вимагає велику кількість часу, яку насправді можна спрямувати на створення особливих функціональних можливостей, що стосуються конкретної обраної предметної області. Саме тому в таких ситуаціях є доцільним використання програмних каркасів – фреймворків.

Означення 53. *Django — високорівневий Python-фреймворк, призначений для розробки веб-орієнтованих систем.*

Огляд фреймворку

Структура Django-систем побудована відповідно до популярного шаблону проектування MVC («Модель — Представлення — Контролер»). В основі цього шаблону лежить поділ на три взаємопов'язані компоненти: модель даних, представлення (або інтерфейс користувача) та керування. Основна його ідея в тому, аби модифікація елементів інтерфейсу користувача практично не впливала на роботу з даними предметної області системи, а зміни в даних призводили до мінімальних змін у представленнях.

Контролер в Django реалізований самим фреймворком за допомогою диспетчера URL, який делегує обробку запиту відповідному для даного URL обробнику.

Оскільки частина шаблону вже реалізована, Django називають MTV-фреймворком:

- *модель (M)* є рівнем доступу до даних та представляє сутності предметної області;
- *шаблон (T)* є рівнем відображення; на цьому рівні приймаються рішення, що стосуються відображення даних на веб-сторінці чи в іншому документі;
- *представлення (V)* є рівнем бізнес-логіки; на цьому рівні розміщена логіка доступу до моделі та вибору потрібного шаблону; в деякому наближенні це міст між моделями та шаблонами, проте представлення тут не є в повній мірі аналогом контролера в MVC.

Об'єктно-реляційне відображення. Моделі. Міграції

Частиною фреймворку Django є також об'єктно-реляційне відображення, яке дозволяє перетворювати моделі в таблиці бази даних та виконувати запити за допомогою спеціальних методів моделі, а не шляхом написання SQL-запитів.

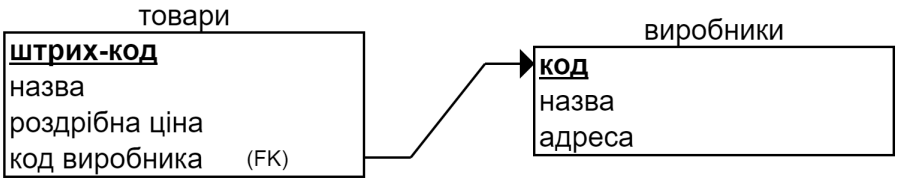


Рис. 5.2: Зв'язок «виробник виготовляє товар»

Наприклад, зв'язок множинністю «один до багатьох» між виробниками та товарами, табличну модель якого подано на Рисунку 5.2, реалізується в моделях Django дуже схоже до табличного представлення наступним способом:

```
class Manufacturer(models.Model):
    code = models.IntegerField(primary_key=True)
    name = models.TextField()
    address = models.TextField()

class Goods(models.Model):
    code = models.IntegerField(primary_key=True)
    name = models.TextField()
    price = models.DecimalField()
    manufacturer = models.ForeignKey(Manufacturer,
        on_delete=models.CASCADE)
```

В цьому випадку при отриманні зі сховища інформації про всі товари за допомогою наступної команди:

```
Goods.objects.all().select_related()
```

ми отримаємо в полі *manufacturer* кожного товару не код виробника, а об'єкт відповідного класу з усіма його властивостями.

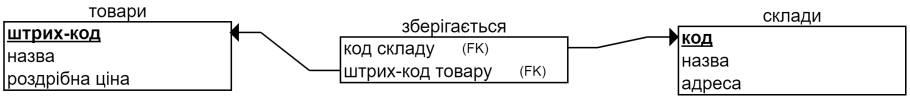


Рис. 5.3: Зв'язок «товар зберігається на складі»

Зв'язок множинністю «багато до багатьох» теж подається в моделях доволі просто та не потребує створення допоміжних класів. Наприклад, зв'язок між складами та товарами, табличну модель якого подано на Рисунку 5.3, реалізується в моделях Django наступним способом:

```
class Goods(models.Model):
    code = models.IntegerField(primary_key=True)
    name = models.TextField()
    price = models.DecimalField()
    manufacturer = models.ForeignKey(Manufacturer,
        on_delete=models.CASCADE)
```

```
class Warehouse(models.Model):
    code = models.IntegerField(primary_key=True)
    name = models.TextField()
    address = models.TextField()
    goods = models.ManyToManyField(Goods)
```

Іншими словами, кожен склад характеризується певним переліком товарів, що в ньому зберігаються. При цьому отримати дані про товари для, наприклад, складу з кодом *1* можна за допомогою вбудованих методів, які надаються фреймворком:

```
Warehouse.objects.get(code=1).goods.all()
```

Доступним є і перелік складів для кожного з товарів, хоча явно для нього жодного поля задано вище не було. Аби отримати перелік складів, на яких зберігається, наприклад, товар зі штрих-кодом *245558543* можна за допомогою наступної команди:

```
Goods.objects.get(code=245558543).warehouse_set.all()
```

Додавання елемента до переліку є також доволі простим та може бути здійснений за допомогою наступних команд:

```
warehouse = Warehouse.objects.get(code=1)
item = Goods.objects.get(code=245558543)
warehouse.goods.add(item)
```

За допомогою інструментів міграції Django дозволяє автоматично відтворювати вищенаведені зв'язки між моделями в пов'язаній з проектом реляційній базі даних.

Міграції — це засіб перетворення змін, внесених до моделей проекту, у відповідні зміни до реляційної схеми бази даних.

Міграції зберігаються на диску в спеціальних файлах міграції, які можна записувати власноруч, або ж генерувати за допомогою наступної команди автоматично:

```
python manage.py makemigrations
```

Після того, як всі нові файли міграції готові, виконання відповідних дій над структурою бази даних відбувається за допомогою команди:

```
python manage.py migrate
```

Слід зазначити, що проекти на базі Django саме завдяки апарату міграцій є незалежними від системи управління базами даних. Переведення

проекту з однієї СУБД на іншу потребує лише внесення змін до конфігурації проекту, пов'язаних зі з'єднанням з базою даних, а також запуску процесу міграції. Решту роботи виконає сам фреймворк відповідно до зробленого вами вибору.

Запити до бази даних

Ми вже розглянули кілька запитів до бази даних, що здійснюються за посередництвом моделей.

Особливістю запитів цього фрейворку є так званий лінивий тип обчислення. Записаний вами запит є екземпляром класу *QuerySet*, що накопичує вказані вами інструкції в SQL-запиті, а виконує його лише тоді, коли буде затребувано виведення результату. Це дозволяє добудовувати до запиту ланцюжки умов, як показано в прикладі:

```
Goods.objects.filter(
    price__gt=20
).exclude(
    manufacturer__name='Кулиничі'
).order_by('-price')
```

Вищевказана команда акумулює товари з ціною вище *20 гривень*, що не виготовляється виробником *Кулиничі*, та впорядковує їх за спаданням ціни. Отриманий за допомогою цієї команди об'єкт *QuerySet* згенерує SQL-запит до бази даних, еквівалетний наступному:

```
SELECT goods.*
FROM goods INNER JOIN manufacturer
    ON goods.manufacturer_code = manufacturer.code
WHERE goods.price > 20 AND
    NOT (manufacturer.name = "Кулиничі")
ORDER BY goods.price DESC;
```

Слід звернути особливу увагу на те, що відсутність явного SQL-коду дозволяє уникнути помилок при формуванні умов об'єднань та уникати залежності інформаційної системи від конкретної системи управління базами даних. З іншого боку, фреймворк надає можливості для написання запитів мовою SQL, а тому жодним чином не обмежує можливості вибірки інформації. Але засоби фреймворку достатньо потужні і включають також групування, агрегатні функції, обчислювані поля та багато інших можливостей, а тому розробники фрейморку вкрай не рекомендують вдаватися до написання чистого SQL-коду в продуктах на базі Django.

Представлення та шаблони

Після побудови відповідних моделей, для кожної дії (наприклад, відображення сторінки з таблицею всіх товарів та відповідних їм виробників) створюється відповідне їй представлення на кшталт наступного:

```
def display_goods(request):
    goods = Goods.objects.all().select_related()
    return render(request, 'goods.html', {
        'goods': goods
    })
```

Це представлення вказує шаблон, який використовується для відображення інформації, та передає в цей шаблон відповідні дані, які далі використовуються для подання даних в табличному форматі:

```
<h1>Перелік товарів</h1>
<table>
{% for item in goods %}
    <tr>
        <td>{{ item.code }}</td>
        <td>{{ item.name }}</td>
        <td>{{ item.address }}</td>
        <td>{{ item.manufacturer.name }}</td>
    </tr>
{% endfor %}
</table>
```

В даному випадку готова HTML-сторінка готується вже на сервері і після цього відправляється клієнту для відображення сторінки в браузері.

API та Django REST Framework

У випадку API-підходу, відповідне представлення повертає не HTML-сторінку, а дані в одному з популярних форматів — зазвичай JSON.

Для реалізації цього підходу варто використовувати окремий Django REST Framework, який значно полегшує створення відповідних представлень. Для найпростішого API в додаток до створених моделей необхідно створити їх серіалізатори — транслятори структури даних у певний збережуваний формат (в даному випадку — JSON). Наприклад, для побудованих нами моделей для складів, виробників та товарів серіалізатори матимуть наступний вигляд:

```
class ManufacturerSerializer(
    serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Manufacturer
        fields = ('code', 'name', 'address')

class WarehouseSerializer(
    serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Warehouse
        fields = ('code', 'name', 'address')

class GoodsSerializer(
    serializers.HyperlinkedModelSerializer):
    manufacturer = ManufacturerSerializer()
    class Meta:
        model = Goods
        fields = ('code', 'name', 'price',
            'manufacturer')
```

Доступ до даних отримується так само за допомогою представлень, що ідентифікуються відповідно до URL отриманого запиту. Типові представлення для створених вище серіалізаторів матимуть наступний вигляд:

```
class ManufacturerViewSet(viewsets.ModelViewSet):
    queryset = Manufacturer.objects.all()
    serializer_class = ManufacturerSerializer

class WarehouseViewSet(viewsets.ModelViewSet):
    queryset = Warehouse.objects.all()
    serializer_class = WarehouseSerializer

class GoodsViewSet(viewsets.ModelViewSet):
    queryset = Goods.objects.all()
    serializer_class = GoodsSerializer
```

Як легко помітити, в найпростішому представленні вказуються джерело даних та клас серіалізатора, за допомогою якого об'єкти джерела даних можуть бути перетворені у формат JSON.

В результаті буде отримано найпростіший API, що дозволятиме перегляд об'єктів.

Типовий результат запиту переліку товарів у форматі JSON в даному випадку матиме наступний вигляд:

```
[
    {
        "code": 245558543,
        "name": "Батон білий",
        "price": 14.3,
        "manufacturer": {
            "code": 3,
            "name": "Кулиничі",
            "address": "м. Харків, вул. Грищенка, 17"
        }
    },
    ...
]
```

Результат такого запиту може оброблятися застосунком-клієнтом за допомогою засобів мови JavaScript.

Такий підхід до побудови інформаційних систем є популярним з кількох причин. По-перше, така організація системи дозволяє відділити розробку серверної (backend) частини програмного засобу, що відповідає переважно за логіку обробки даних та їх збереження, від його клієнтської частини, яка переважно зосереджена на поданні інформації у зручному для користувача форматі. Таким чином, розробники не обов'язково повинні володіти засобами розробки обох частин додатку і можуть працювати практично незалежно один від одного. По-друге, API є чудовим способом ділитися наявними даними у мережі. Дані, отримані в стандартному форматі за допомогою API, можуть зручно оброблятися стороннім програмним забезпеченням інших розробників.

Детально з фреймворками Django та Django REST Framework, в тому числі з більш детальними прикладами, ви можете познайомитися в навчальному посібнику Django Girls [9] та офіційній документації [7] в мережі Інтернет.

Предметний покажчик

Посилання

[1] Badia A., Lemire D. Functional dependencies with null markers. — Computer Journal 58 (5), 2015. — 1160-1168 P.

[2] Connolly T. M., Begg C. E. Database Systems. A Practical Approach to Design, Implementation, and Management. Sixth Edition. — Pearson Education, 2014. — 1440 P.

[3] Lambert J. Microsoft Access 2016 Step by Step. — Microsoft Press, 2016. — 448 P.

[4] DIGITAL IN 2018: WORLD'S INTERNET USERS PASS THE 4 BILLION MARK [Електронний ресурс]. — Режим доступу: https://wearesocial.com/blog/2018/01/global-digital-report-2018.

[5] Interactive: The Top Programming Languages 2018 [Електронний ресурс]. — Режим доступу: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018.

[6] Буй Д.Б., Сільвейструк Л.М. Формалізація моделі "Сутність-зв'язок— Київ: Видавничо-поліграфічний центр "Київський університет 2011. — 175 с.

[7] Документация Django REST framework [Електронний ресурс]. — Режим доступу: https://ilyachch.gitbooks.io/django-rest-framework-russian-documentation/content/.

[8] Завадський І. О. Основи баз даних: [Навч. посіб.] — Київ: Видавець І. О. Завадський, 2011. — 192 С.

[9] Навчальний посібник Django Girls [Електронний ресурс]. — Режим доступу: https://tutorial.djangogirls.org/uk/django/.

Офіційний веб-сайт підтримки:

isdb.itolymp.com

Кімната на порталі **sql.itolymp.com**:

Книга «Інформаційні системи та бази даних»

Пароль: **isdbbook2018**

Навчальний посібник

Григорій Іванович Гогерчак

ІНФОРМАЦІЙНІ СИСТЕМИ ТА БАЗИ ДАНИХ

Комп'ютерна верстка автора
Коректори: Н.В. Речич, Є.А. Варзар

Видавництво «Лікей»
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготівників
і розповсюджувачів видавничої продукції
серія ДК № 1371 від 28 травня 2003 року

Підписано до друку 8 травня 2019 року