ЗАКАРПАТСЬКИЙ УГОРСЬКИЙ ІНСТИТУТ ІМЕНІ ФЕРЕНЦА РАКОЦІ ІІ

II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA

Кафедра математики та інформатики
Matematika és Informatika Tanszék

МЕТОДИ ПРОЕКТУВАННЯ БАЗ ДАНИХ

ADATBÁZIS TERVEZÉSI MÓDSZEREK

МЕТОДИЧНІ ВКАЗІВКИ ДО САМОСТІЙНОЇ РОБОТИ/ MÓDSZERTANI ÚTMUTATÓ AZ ÖNÁLLÓ MUNKÁHOZ

Системи керування базами даних/ Adatbázis-kezelő rendszerek
(назва навчальної дисципліни/a tantárgy neve)
Перший (бакалаврський) / Alapképzés (BSc)
(рівень вищої освіти / felsőoktatás szintje)
01 Освіта/Педагогіка / 01 Oktatás/Pedagógia
(галузь знань / képzési ág)
Середня освіта (Інформатика) / Középiskolai oktatás (Informatika)
(освітня програма / képzési program)

Берегове / Beregszász 2025 р.

Методичні вказівки «Методи проектування баз даних» до самостійної роботи з дисципліни «Системи керування базами даних» розроблені на основі Освітньої програми підготовки бакалаврів з галузі знань «01 Освіта/Педагогіка» за напрямом «014 Середня освіта (Інформатика)» як для студентів денної, так і заочної форми навчання. Метою методичного посібника є ознайомлення студентів з основними методами проектування баз даних та з інструметальними засобами роботи з MySQL. Методичні вказівки можуть бути використані при самостійній роботі по даному курсу студентами денної та заочної форм навчання.

Затверджено до використання у навчальному процесі
на засіданні кафедри математики та інформатики ЗУІ ім. Ф.Ракоці ІІ
(протокол № 10 від «23 » червня 2025 року)

Розглянуто та рекомендовано Навчально-методичною радою
Закарпатського угорського інституту імені Ференца Ракоці ІІ
(протокол № 7 від « 26 » серпня 2025 року)

Рекомендовано до видання в електронній формі (PDF)
рішенням Вченої ради Закарпатського угорського інституту імені Ференца Ракоці ІІ
(протокол № 8 від «28» серпня 2025 року)

Підготовлено до видання в електронній формі (PDF) кафедрою математики та інформатики спільно з Видавничим відділом Закарпатського угорського інституту імені Ференца Ракоці ІІ

Розробник методичних вказівок:

Йожеф ГОЛОВАЧ – професор кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доктор технічних наук

Рецензенти:

Федір ГЕЧЕ – професор УжНУ, доктор техніічих наук, професор

Олександр ТИЛИЩАК – професор кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доктор фізико-математичних наук, доцент

Відповідальні за випуск:

Каталін КУЧІНКА – завідувач кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доцент, кандидат фізико-математичних наук

Олександр ДОБОШ – начальник Видавничого відділу ЗУІ ім. Ф.Ракоці ІІ

За зміст методичних вказівок відповідальність несе розробник.

Видавництво: Закарпатський угорський інститут імені Ференца Ракоці ІІ (адреса: пл. Кошута 6, м. Берегове, 90202. Електронна пошта: foiskola@kmf.uz.ua)

Az „Adatbázis-tervezési módszerek” című módszertani útmutató az „Adatbázis-kezelő rendszerek” tárgy területen folytatott önálló munkához a BSc „Középiskolai oktatás (Informatika)” képzési programja alapján lett kidolgozva nappali és levelező tagozatos hallgatók részére. A módszertani kézikönyv célja, hogy megismertesse a hallgatókat az adatbázis-tervezés alapvető módszereivel és a MySQL-lel való munkavégzés eszközeivel. A módszertani útmutatókat a kurzuson részt vevő nappali és levelező hallgatók önálló munkájához használhatják.

Az oktatási folyamatban történő felhasználását jóváhagyta a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszéke

(2025.06.23. , 10 számú jegyzőkönyv).

Megjelentetésre javasolta a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola
Oktatási és Módszertani Tanácsa
(2025.augusztus 26., 7. számú jegyzőkönyv).

Elektronikus formában (PDF fájlformátumban) történő kiadásra javasolta
a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Tudományos Tanácsa
(2025. augusztus 28., 8. számú jegyzőkönyv).

Kiadásra előkészítette a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola
Matematika és Informatika Tanszéke, valamint Kiadói Részlege.

A módszertani útmutató kidolgozója:

HOLOVÁCS József – professzor, műszaki tudományok doktora, a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszékének professzora

Szakmai lektorok:

GECSE Ferenc – Az UNE professzora, műszaki tudományok doktora, professzor

TILISHYAK Sandor – a fizikai és matematikai tudományok doktora, docens, a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszékének professzora

A kiadásért felelnek:

KUCSINKA Katalin – PhD, a fizikai és matematikai tudományok kandidátusa, a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszékének tanszékvezetője és docense

DOBOS Sándor – a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Kiadói Részlegének vezetője

A segédlet tartalmáért kizárólag a módszertani útmutató kidolgozója felel.

Kiadó: II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola (cím: 90 202, Beregszász, Kossuth tér 6. E-mail: foiskola@kmf.uz.ua)

ЗМІСТ

Передмова

Важливі напрямки сучасних інформаційних технологій базуються на концепції баз даних. Для ефективного функціювання баз даних створені різні системи управління базами даних (СУБД). На даний час найбільш поширені реляційні СУБД, серед яких одною з найпопулярніших є MySQL.

Одним з основних завдань, які стоять перед студентами, які вивчають дисципліну „Системи управління базами даних", є засвоєння сучасних методів проектування баз даних та володіння інструментальними засобами для їх реалізації у різних СУБД, наприклад, MySQL.

У пропонованому методичному посібнику описано метод проектування, який базується на понятті ER-моделі. Для покращення характеристик відношень, таблиць баз даних використовується метод нормалізації відношень. Методичний посібник також містить опис методу нормалізації, нормальні форми.

Для реалізації теоретичних знань студенти повинні володіти відповідними інструментальними засобами. З цією метою методичний посібник містить рекомендації по використанню MySQL Workbench, а також Dbforge Studio Devart.

Для більш поглибленого вивчення даної проблематики, рекомендується ознайомитись з наведеною в кінці посібника літературою. Наведені в посібнику приклади рекомендується виконати самостійно на комп'ютері з варіюванням вхідних даних та опціональних параметрів команд.

1. Поняття ER-моделі. Сутність. Атрибути

При проектуванні бази даних та розробці програмного продукту найбільш важливою проблемою є проблема взаємодії розробника з замовником. Задача розробника – найбільш точніше відтворити побажання замовника при розробці програмного продукту керування базою даних. Основна проблема, яку потрібно вирішити розробнику – правильна побудова бази даних, а точніше схеми (структури) бази даних.

Крім того, розробник додатково вирішує проблеми:

- пошук ефективних алгоритмів;
- підбір належних структур даних;
- відлагодження та тестування складного коду;
- дизайн та зручність інтерфейсу додатку.

У процесі розробки програмного забезпечення, що керує базою даних, розробник повинен детально вивчити вимоги замовника. База даних повинна бути побудована таким чином, щоб вона була зрозумілою, найточніше відображала вирішувану проблему і не містила надлишковості в даних.

Щоб полегшити процес розробки (проектування) бази даних, використовуються *семантичні моделі* даних. Для баз даних найбільш відомою є **ER-модель даних (Entity-Relationship model).**

ER-модель (Entity-relationship model або Entity-relationship diagram) – це семантична модель даних, яка призначена для спрощення процесу проектування бази даних. З ER-моделі можуть бути породжені всі види баз даних: реляційні, ієрархічні, мережні, об'єктні.

В основі ER-моделі лежать поняття "сутність", "зв'язок" та "атрибут".

Для великих баз даних побудова ER-моделі дозволяє уникнути помилок проектування, які надзвичайно важко виправляти, особливо, якщо база даних вже експлуатується чи на стадії тестування. Помилки в розробці структури бази даних може призвести до перебудови коду програмного забезпечення, що керує цією базою даних. У результаті час, кошти та людські ресурси будуть використані неефективно.

ER-модель – це представлення бази даних у вигляді наочних графічних діаграм. ER-модель візуалізує процес, що визначає деяку предметну область.

Діаграма "сутність-зв'язок" – це діаграма, яка представляє в графічному вигляді сутності, атрибути і зв'язки.

ER-модель – це тільки концептуальний рівень моделювання. ER-модель не містить деталей реалізації. Для тієї самої ER-моделі деталі її реалізації можуть відрізнятися.

***Сутність* в базі даних** – це будь-який об'єкт в базі даних, який можна виділити виходячи з суті *предметної області* для якої розробляється ця база даних. Розробник бази даних повинен вміти правильно визначати сутності.

Приклад. В базі даних книжного магазину можна виділити такі сутності:

- книга;
- постачальник;
- розміщення в магазині.

Приклад. В базі даних обліку навчального процесу деякого навчального закладу можна виділити такі сутності:

- студенти;
- викладачі;
- групи;
- дисципліни, що вивчаються.

У моделі “сутність”-“зв’язок” розрізняють два різновиди типів сутностей:

- *слабкий тип*. Цей тип сутності є залежним від сильної сутності;
- *сильний тип*. Це самостійний тип сутності, який ні від кого не залежить.

На рисунку 1 зображено позначення слабого та сильного типу сутності в ER-моделі.

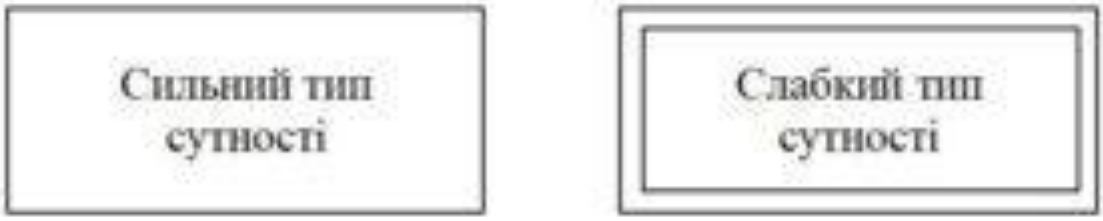


Рис. 1. Позначення сильного та слабкого типу сутності

Кожен тип сутності має певний набір атрибутів. Атрибути призначені для опису конкретної сутності.

Розрізняють наступні види атрибутів:

- ***прості*** атрибути. Це атрибути, які можуть бути частиною складених атрибутів. Ці атрибути складаються з одного компонента. Наприклад, до простих атрибутів можна віднести: код книги в бібліотеці або курс навчання студента в навчальному закладі;
- ***складені*** атрибути. Це атрибути, які складаються з декількох простих атрибутів. Наприклад, адреса проживання може містити назву країни, населеного пункту, вулиці, номеру будинку;
- ***однозначні*** атрибути. Це атрибути, які містять тільки одне єдине значення для деякої сутності. Наприклад, атрибут “Номер залікової книги” для типу сутності “Студент” є однозначним, тому що студент може мати тільки один номер залікової книги (одне значення);
- ***багатозначні*** атрибути. Це атрибути, які можуть містити декілька значень. Наприклад, багатозначний атрибут “Номер телефону” для сутності “Студент”, тому що студент може мати декілька номерів телефону (домашній та мобільний);

- ***довільні*** атрибути. Це атрибути, значення яких формуються на основі значень інших атрибутів. Наприклад, поточний курс студента можна обчислити на основі різниці поточного року навчання та року вступу студента в навчальний заклад (якщо студент не мав проблем з навчанням).

На ER-діаграмі атрибути позначаються так, як зображено на рисунку 2. Як видно з рисунку, будь-який атрибут позначається у вигляді еліпсу з назвою всередині еліпсу. Якщо *атрибут є первинним ключем, то його назву підкреслюють.*

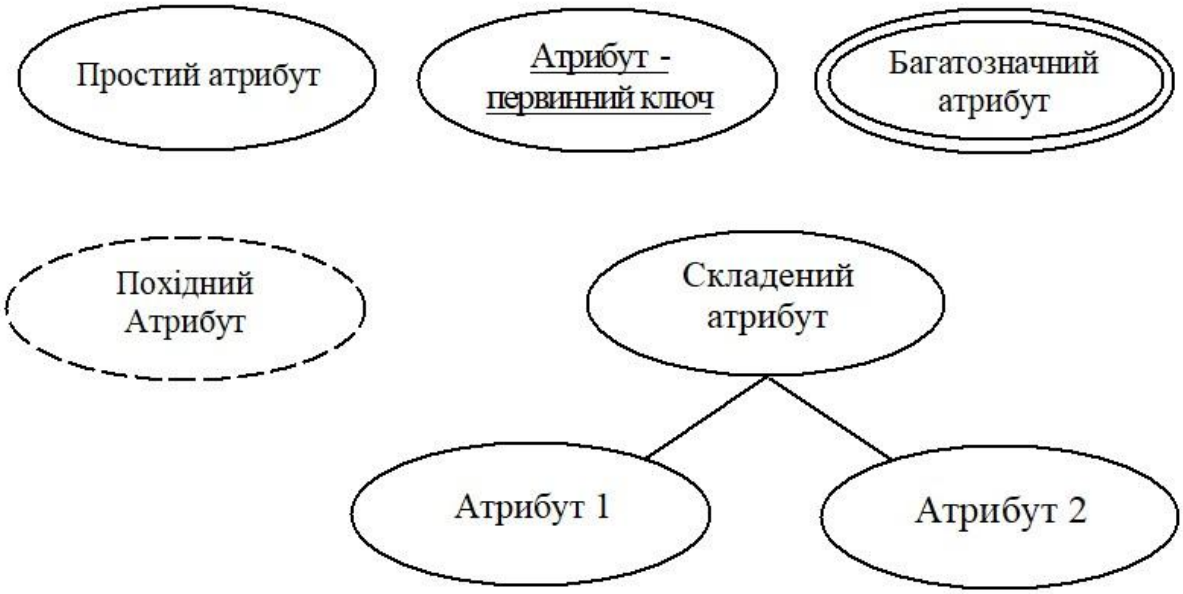


Рисунок 2. Представлення атрибутів на діаграмах ER-моделі

При розробці програм керування базами даних, типи сутностей та їх атрибути можна представляти по різному при цьому дотримуючись декількох підходів:

- вибрати в якості джерела даних відому технологію (наприклад MySQL, Oracle, Microsoft Access, Microsoft ODBC Data Source тощо), яка вже досліджена, протестована, стандартизована та має потужний набір засобів керування базою даних;

- розробити власний формат бази даних та реалізувати методи її обробки, а взаємодію з відомими джерелами даних реалізувати у вигляді спеціальних команд на зразок “Імпорт/Експорт”. У цьому випадку доведеться власноруч програмувати всю рутинну роботу з ведення та забезпечення надійної роботи бази даних;

- реалізувати поєднання двох вищенаведених підходів. Сучасні засоби розробки програмного забезпечення мають потужний набір бібліотек для обробки складних наборів та візуалізації даних в них (колекції, масиви, компоненти візуалізації тощо).

Якщо база даних реалізується у відомих реляційних СКБД, то типи сутностей представляються таблицями.

Атрибути з ER-моделі відповідають полям таблиці.

Один запис в таблиці бази даних представляє один екземпляр сутності.

Атрибути реалізується наступним чином:

- *простий атрибут* чи однозначний атрибут може бути представлений доступним набором базових типів, який є у будь-якій мові програмування. Наприклад, цілочисельні атрибути представляються типом int, integer, uint і т.д.; атрибути, що містять, дробову частину можуть бути представлені типом float, double; рядкові атрибути типом string і т.д.;
- *складений атрибут* – це об'єкт, що включає в себе декілька вкладених простих атрибутів. Наприклад, у більшості СКБД складений атрибут деякої таблиці може формуватись на основі набору простих типів (полів). У мовах програмування об'єднання полів реалізується структурами або класами;
- *багатозначний атрибут* може бути реалізований масивом або колекцією простих чи складених атрибутів;
- *довільний атрибут* реалізується додатковим полем, яке обчислюється при звертанні до таблиці. Таке поле називається *обчислювальним полем (calculated field) і* формується на основі інших полів таблиці;
- *атрибут, що є первинним ключем* може бути цілочисельним, рядковим чи іншого порядкового типу. У цьому випадку, значення кожної комірки таблиці, що відповідає первинному ключу, є унікальним. Найчастіше, в якості первинного ключа виступає цілий тип (int, integer).

Якщо база даних реалізована в унікальному форматі, то типи сутностей зручніше представляти у вигляді класів чи структур. Атрибути сутності реалізуються у вигляді полів (внутрішніх даних) класу. Методи класу реалізують необхідну обробку полів класу (атрибутів). Взаємодія (зв'язок) між класами реалізується з допомогою спеціально розроблених інтерфейсів з використанням відомих шаблонів проектування.

Приклад фрагменту ER-моделі для типу сутності "Студент"

Наведений приклад демонструє фрагмент ER-моделі для типу сутності "Студент".

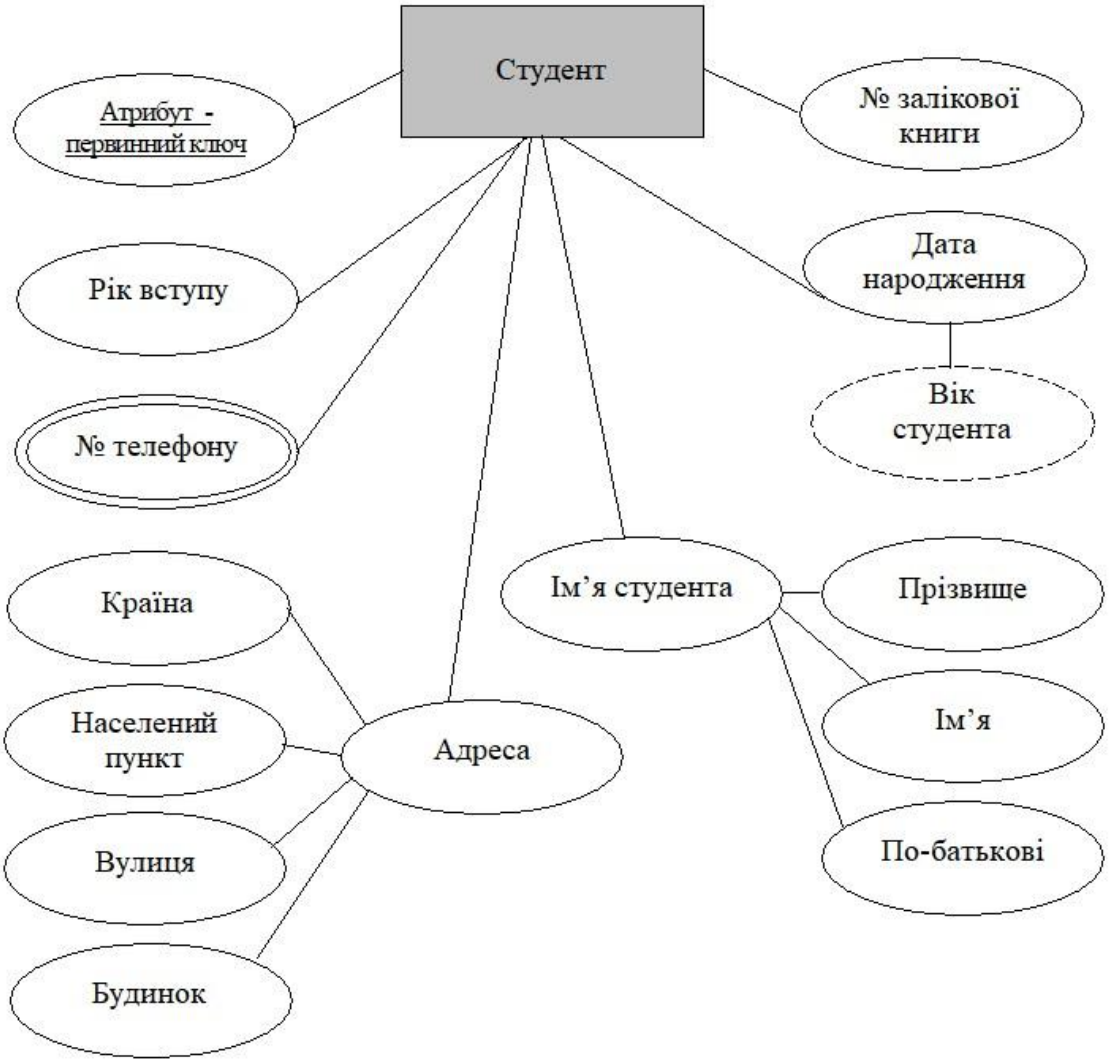


Рис 3. Фрагмент ER-моделі для типу сутності "Студент"

На вищенаведеному рисунку оголошуються наступні атрибути, які в СКБД можуть мати наступні типи:

- Атрибут Первинний ключ – є унікальним цілочисельним значенням яке формується автоматично. В СКБД це є поле-лічильник;
- атрибут Рік вступу – простий атрибут, який можна реалізувати цілочисельним значенням (int, integer);
- атрибут Номер телефону – багатозначний атрибут, який може бути реалізований як масив чи колекція тощо;
- атрибут Номер залікової – простий атрибут, який можна реалізувати як рядок символів, оскільки номер залікової крім цифр може містити і літери;
- атрибут Країна, Місто, Вулиця, Номер будинку – це атрибути, які утворюють складений атрибут Адреса. Всі ці атрибути можуть бути рядкового (текстового) типу (string, Text);
- атрибут Прізвище, Ім'я, По-батькові – це прості атрибути, які є частиною складеного атрибуту Ім'я студента. Всі ці атрибути можуть бути рядкового (текстового) типу (string, Text);
- атрибут День народження – простий атрибут типу Дата (DateTime);

- атрибут Вік студента – розрахункове поле, яке визначається як різниця поточної (системної) дати та значення атрибуту День народження

1.1. Підтипи сутностей

Підтипи сутностей вводяться в ER-модель з метою зменшення загальної кількості атрибутів кожної сутності. Кожна сутність має набір унікальних атрибутів. Однак, атрибути різних сутностей можуть повторюватись. Тому, потрібно так побудувати ER-модель, щоб кількість повторюваних атрибутів в різних сутностях була мінімальною або зведеною до нуля.

Повторення атрибутів несе надлишковість в базі даних. Розмір бази даних стає необгрунтовано великий, тому цю проблему потрібно виправляти. Для вирішення цієї проблеми використовуються підтипи сутностей.

Ідея використання підтипу сутності полягає в тому, що для усього різноманітного набору сутностей *виділяється супертип*, який містить загальну для усіх типів сутностей інформацію. Деталі кожного типу сутності виносяться окремо в декілька *спеціалізованих підтипів.*

Приклад. Нехай потрібно розробити базу даних працівників навчального закладу. У навчальному закладі виділяються 3 сутності, кожна з яких представляє професійну групу працівників:
- сутність Адміністрація;
- сутність Викладач;
- сутність Допоміжний персонал.

Якщо для кожної сутності описати власний набір атрибутів, то можна помітити, що деякі атрибути в різних сутностях будуть повторюватись.

Нижченаведені атрибути є спільними для усіх сутностей:

- “Прізвище”;
- “Ім’я”;
- “По-батькові”;
- “Ідентифікаційний номер”.

Також можна виділити деякі унікальні атрибути:

- тип сутності “Адміністрація” має адміністративну ставку, назву займаної посади тощо;
- тип сутності “Викладач” має кількість вичитаних годин, посаду тощо;
- тип сутності “Допоміжний персонал” має ставку персоналу, коефіцієнт вихідного чи невихідного дня (якщо працівник працював у вихідні дні) тощо.

Щоб вирішити проблему уникнення повторюваності, в ER-моделі вносяться зміни, як показано на рисунку 1, а саме:

- вводиться супертип сутності “Працівник”. Цей супертип містить спільні атрибути для усіх типів (підтипів) сутностей;
- вводяться підтипи сутностей “Адміністрація”, “Викладач”, “Допоміжний персонал”. Кожен з підтипів сутностей має свої унікальні атрибути.

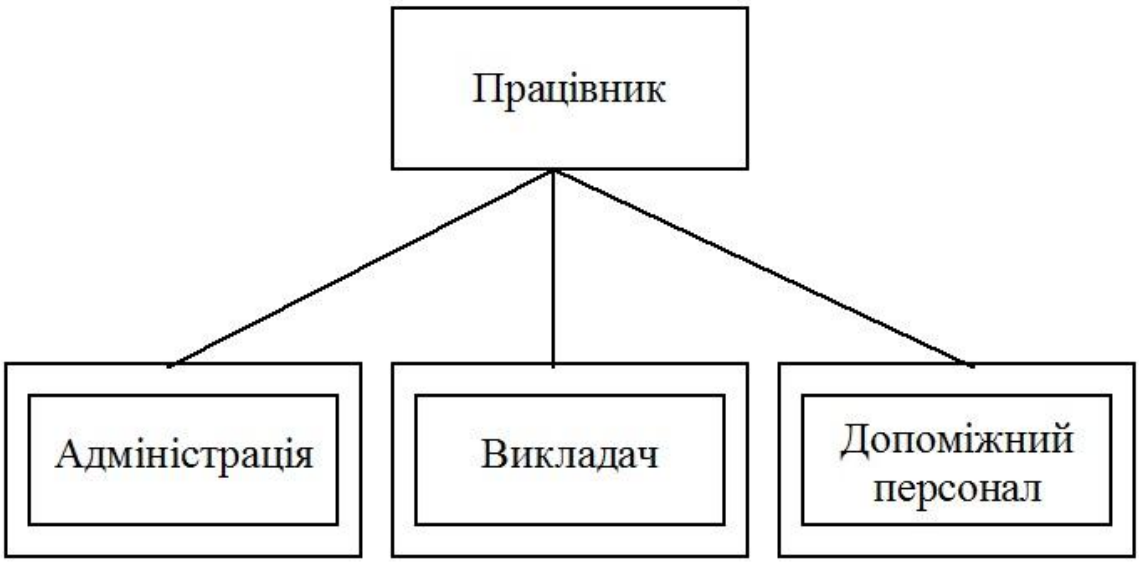


Рис.4. Зображення супертипу та підтипів сутностей на діаграмі

Супертип сутності – це такий тип сутності, в якому реалізуються (описуються) тільки спільні атрибути для підтипів сутностей, що використовують цей супертип.

Можливі зв'язки між підтипами сутностей

Підтипи сутностей можуть перетинатися або не перетинатися. Якщо підтип однієї сутності може підходити для підтипу іншої сутності, то це означає що підтипи сутностей *перетинаються* між собою. В іншому випадку підтипи сутностей *не перетинаються*.

Приклад.

У навчальному закладі працівник, що займає адміністративну посаду (підтип Адміністрація) може бути викладачем. Тобто, підтип сутності Адміністрація перетинається з підтипом сутності Викладач.
Однак, працівник з підтипу сутності Допоміжний персонал не може бути викладачем. Тому підтип сутності Допоміжний персонал не перетинається з підтипом Викладач.
З врахуванням вищесказаного, зображення супертипу та підтипів на діаграмі має вигляд, як показано на рисунку.

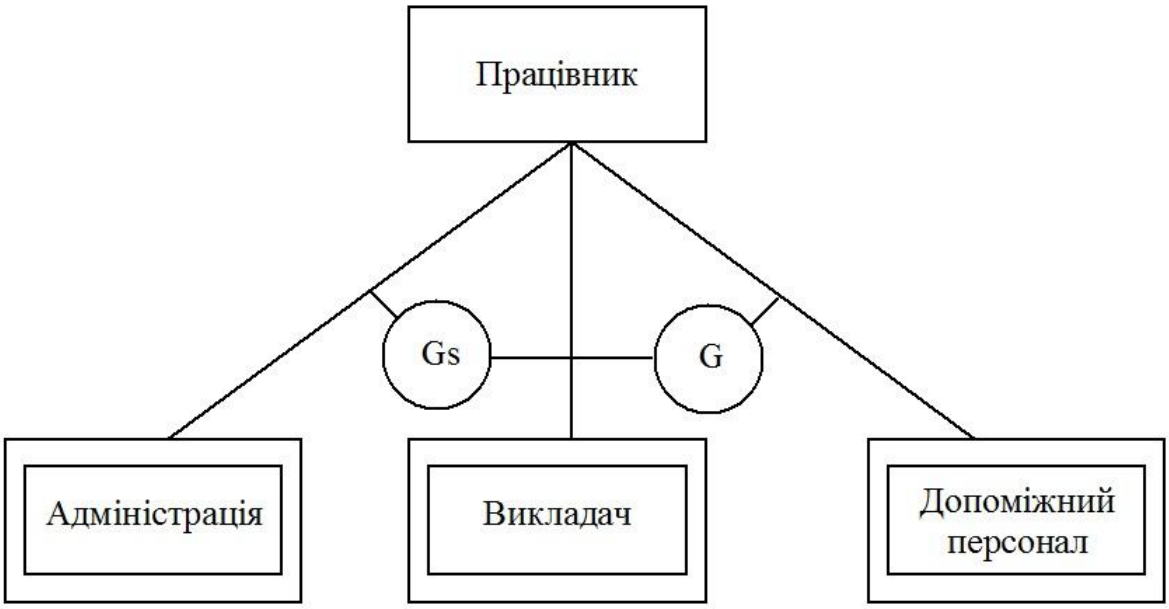


Рис.5. Супертип та підтипи сутностей з врахування перетину між підтипами сутностей

На рисунку між підтипами Адміністрація та Викладач є позначення з символом Gs. Це означає, що підтипи можуть перетинатись. Між підтипами Викладач та Допоміжний персонал є позначення з символом G. Це означає, що підтипи сутностей не перетинаються.

Недоліки використання підтипів сутностей при проектуванні баз даних

Кожен супертип та підтип сутностей реалізується відповідною таблицею. Поля таблиці є атрибутами даного типу сутностей.

Недоліки використання підтипів сутностей полягають в тому, що в цілому ускладняється робота з розробки бази даних. Труднощі полягають в наступному:

- складність в організації взаємодії між головною та підлеглою таблицями для реляційних баз даних;
- в таблицях підтипів потрібно додатково вводити первинні ключі для зв'язування з таблицею супертипу;
- запити на мові SQL стають більш складнішими, оскільки потрібно обробляти дані в зв'язаних таблицях;
- складність в забезпеченні безпечного доступу користувача до стовпців таблиці.

Переваги використання підтипів сутностей при проектуванні баз даних

Переваги використання підтипів сутностей:

- уникнення надлишковості даних, оскільки кожен підтип містить тільки унікальну інформацію. Це в свою чергу призводить до зменшення розміру самої бази даних;
- спрощення задавання обмежень для кожного атрибуту, який відповідає стовпцю таблиці яка реалізує конкретний підтип сутності;
- зменшення помилок при програмуванні операцій над таблицею що відповідає конкретному підтипу сутності;

- гнучкість в модифікації структури бази даних. Не потрібно модифікувати таблицю супертипу у випадку додавання/зміни структури таблиці підтипу. Це, в свою чергу, призводить до того, що не потрібно переробляти всю програму керування базою даних.

Тип відношення між супертипом та підтипом

Між супертипом і підтипом встановлюється відношення “один до одного” 1:1.

1.2. Поняття зв’язку у ER-моделі

Зв’язок в ER-моделі

Дві сутності можуть мати між собою зв’язок. Взаємовідносини між сутностями характеризуються дієсловом, яке можна застосувати для взаємодії між ними. Зв’язок – це деяке відношення між двома типами сутностей.

Наприклад. Наведений приклад демонструє визначення зв’язку між сутностями Студент, Дисципліна, Група.
Студент вивчає дисципліну – виділяється дієслово “вивчає”.

Студент навчається у групі – виділяється дієслово “навчається”.

Позначення зв’язків в ER-моделі

В ER-моделі зв’язки позначаються у вигляді ромба. Всередині ромба вказується дієслово, яке визначає характер взаємодії між сутностями.

Типи зв’язків в ER-моделі

Розрізняють 3 типи зв’язків між сутностями:

- “один до одного” або **1:1**. Це означає, що одному екземпляру сутності може відповідати тільки один екземпляр іншої сутності;
- “один до багатьох” або **1:M**. Це означає, що одному екземпляру сутності може відповідати будь-яка кількість (M) екземплярів іншої сутності. Якщо відоме значення максимальної кількості екземплярів, то це значення вказується замість символу M;
- “багато до багатьох” або **M:N**. Це означає, що декільком екземплярам однієї сутності може відповідати декілька екземплярів іншої сутності.

Приклад зв’язку типу “один до багатьох” – 1:M

На рисунку 1 зображено фрагмент ER-моделі, що демонструє зв’язок між типами сутностей Студент і Група.

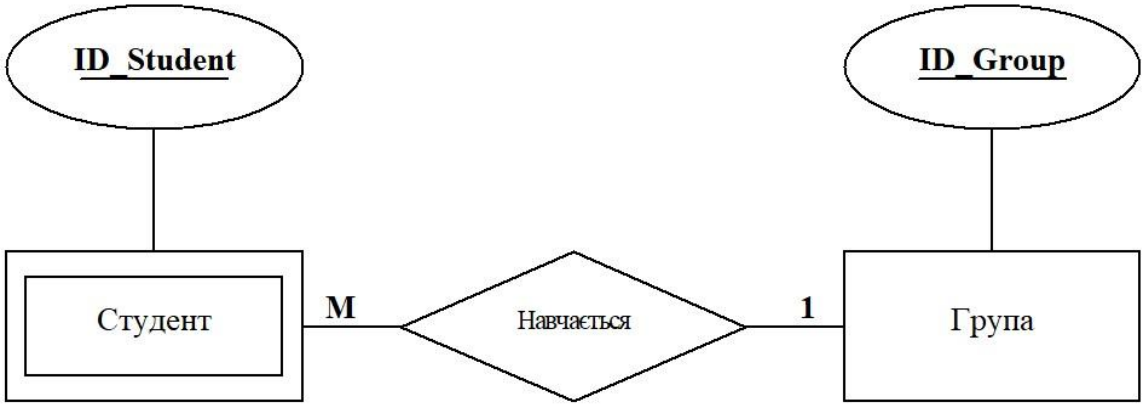


Рис.6. Зв'язок між типами сутностей Студент та Група

На вищенаведеному рисунку зображено два типи сутностей Студент та Група.

Між типами сутностей Студент та Група є зв'язок, тому що студент навчається в групі (ромб з текстом "навчається").

У групі може бути багато студентів. Студент може навчатись тільки в одній групі, у двох і більше групах студент навчатись не може. Тому, на рисунку тип зв'язку позначено як 1:M, що означає "один до багатьох". Позначення M знаходиться біля типу сутності Студент (багато студентів), позначення 1 знаходиться біля сутності Група (одна група).

Тип сутності Студент позначено як "слабка" сутність. Тип сутності Група позначено як сильна сутність. Це означає, що студент знаходиться в підлеглому стані до групи, в якій він навчається.

Потужність зв'язку

Потужність зв'язку – це значення максимальної кількості конкретних екземплярів сутностей, які можуть використовуватись для даного зв'язку. Потужність зв'язку 5 говорить про те, що у даному зв'язку може бути використано не більше 5 різних екземплярів сутностей. Або іншими словами, не більше 5 відмінних між собою значень.

Приклад. Студент навчається у групі. Між типами сутностей Студент та Група можна встановити зв'язок 1:30, як показано на рисунку 2. Число 30 означає, що у групі може навчатись не більше 30 студентів.

Рис.7. Зв'язок між сутностями Студент та Група

Якщо кількість значень екземплярів сутностей довільна, то потужність зв'язку найчастіше представляється символом **M** або знаком ∞ (нескінченість).

Використання конкретних числових значень в потужності зв'язку дає зручність при розробці програмного забезпечення. Оскільки можна більш якісно реалізувати структури даних.

Приклад зв'язку типу "багато до багатьох" – M:N

Нехай задано два типа сутностей: Студент та Дисципліна. Якщо студент (тип сутності Студент) вивчає деяку дисципліну (тип сутності Дисципліна), то на ER-моделі зв'язок може бути відображений так як показано на рисунку 3.

Рис.8. Відображення зв'язку типу "багато до багатьох"

На рисунку символами M та N позначено тип зв'язку "багато до багатьох". Цей тип зв'язку вибрано, тому що один студент може вивчати декілька (багато) дисциплін, і, навпаки, одну дисципліну може вивчати декілька студентів.

Характерна відмінність типу зв'язку "один до одного" 1:1 порівняно з іншими типами

Тип зв'язку "один до одного" 1:1 відрізняється від інших типів зв'язку тим, що він може бути замінений атрибутом. При проектуванні бази даних, розробник на власний розсуд повинен визначити, чи потрібно використовувати тип зв'язку 1:1, чи достатньо замінити його атрибутом. Все це залежить від специфіки вирішуваної задачі.

Особливості реалізації зв'язку типу "багато до багатьох" M:N

Зв'язок "багато до багатьох" складно реалізувати програмно, тому що реляційні бази даних підтримують зв'язок "один до багатьох". Щоб вирішити цю проблему, розробник бази даних створює штучний тип сутності, який виконує функції комутатора між двома основними сутностями.

Приклад. Для двох типів сутностей Студент та Дисципліна можна реалізувати штучний тип сутності, як показано на рисунку 4.

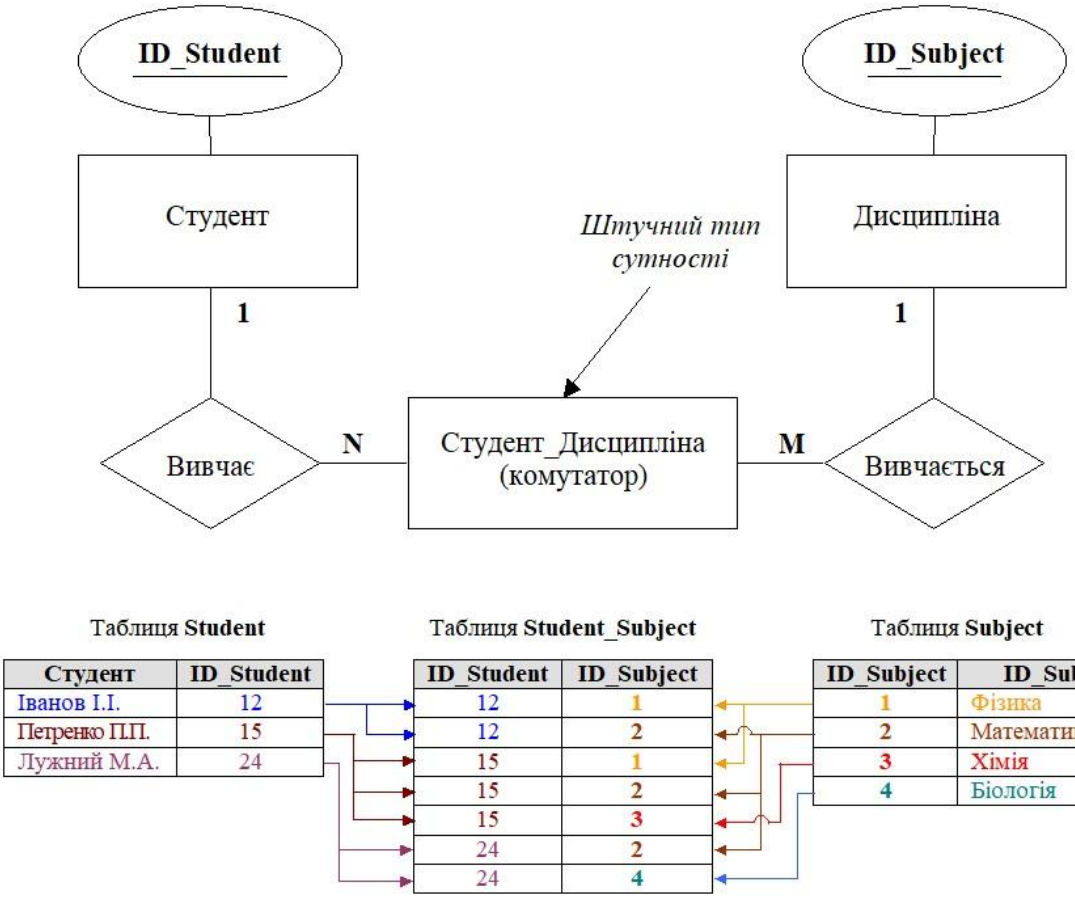


Рис.9. Реалізація зв’язку “багато до багатьох” або M:N

На вищенаведеному рисунку сутність-комутатор містить два атрибути, що є зовнішніми ключами. Один атрибут зв’язаний з підтипом сутності Студент. Другий атрибут зв’язаний з підтипом сутності Дисципліна. Також відображено наближену реалізацію відповідних реляційних таблиць. Таблиця Subject_Student є додатково створеною таблицею, яка відображає штучний тип сутності, який виконує функції комутатора.

Для забезпечення кращої наочності вищенаведеної схеми, часто реалізується спрощений варіант, який зображено на рисунку 5. У цьому випадку штучний тип сутності зображують у вигляді ромбу, вписаного в прямокутник.

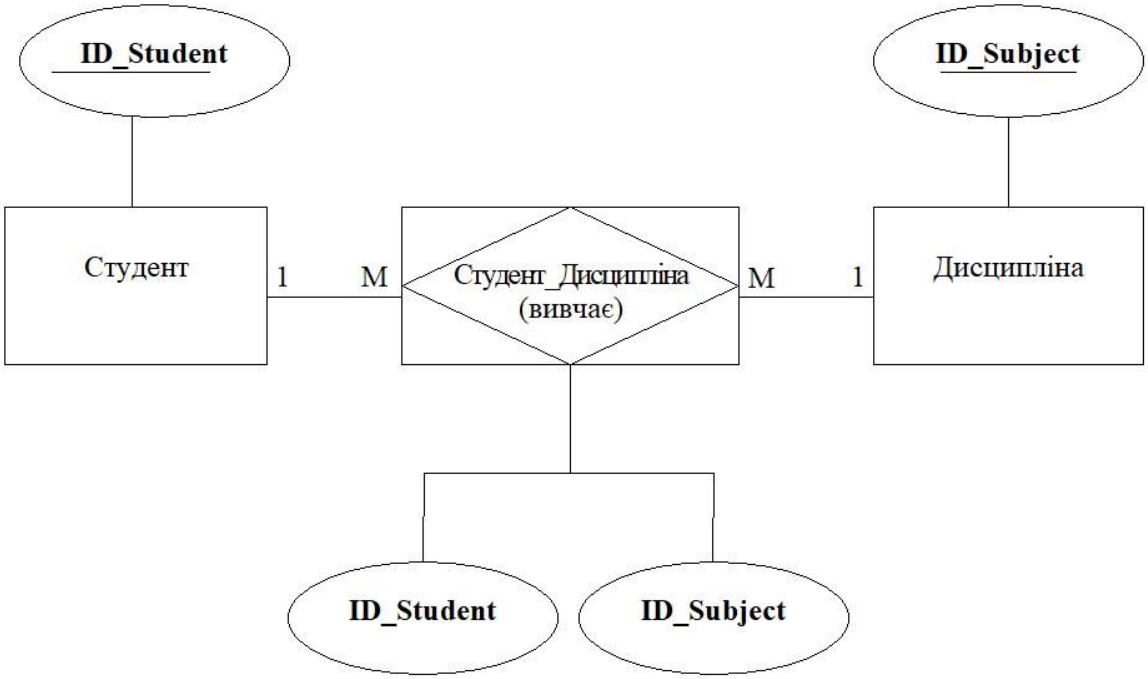


Рис.10. Реалізація спрощеного варіанту штучного типу сутності на ER-діаграмі

2.Нормалізація бази даних

Нормалізація це техніка розробки бази даних, яка зменшує надмірність даних і усуває небажані характеристики, такі як аномалії вставки, оновлення та видалення. Правила нормалізації розділяють великі таблиці на менші та зв'язують їх за допомогою зв'язків. Метою нормалізації в SQL є усунення надлишкових (повторюваних) даних і забезпечення логічного зберігання даних.

Винахідник **реляційної моделі** Едгар Кодд запропонував теорію нормалізації даних із введенням *першої нормальної форми,* і він продовжував розширювати теорію за допомогою *другої та третьої нормальної форми.* Пізніше він приєднався до Реймонда Ф. Бойса для розробки теорії *нормальної форми Бойса-Кодда.*

Нормалізація даних – це багатокроковий процес, що включає застосування набору правил для забезпечення правильної організації та структурування даних у базі даних. Процес нормалізації спрямований на усунення надмірності та залежності даних, що може підвищити цілісність даних і зробити маніпуляцію даними більш ефективною. Нижче наведені основні кроки в нормалізації даних.

2.1. Типи нормальних форм у СУБД

- **1NF (перша нормальна форма).**

Перший крок – це забезпечити, щоб кожен стовпець у таблиці містив атомарні значення, що означає, що кожне значення є неділимим. Це усуває зберігання множини значень в одному атрибуті.

Перша нормальна форма гарантує, що таблиця бази даних організована таким чином, що кожен стовпець містить атомарні (неподільні) значення, а кожен запис є унікальним. Це усуває повторювані групи, тим самим структуруючи дані в таблиці та стовпці.

- **2NF (друга нормальна форма)**

Після досягнення 1NF, *друга нормальна форма* забезпечує, щоб усі неключові атрибути повністю функціонально залежали від первинного ключа. Це означає, що якщо у таблиці є складений первинний ключ, кожен неключовий атрибут повинен залежати від усього складеного ключа, а не тільки від його частини.

На основі 1NF нам потрібно видалити зайві дані з таблиці, які застосовуються до кількох рядків і розмістити їх в окремих таблицях. Вимагається, щоб **усі неключові атрибути функціонально повно залежали від первинного ключа.**

3NF (Третя нормальна форма)

Розширює 2NF, гарантуючи, що всі неключові атрибути те тільки функціонально повно залежали від первинного ключа, але й **не залежали один від одного**. Спираючись на попередні рівні нормалізації, **третя нормальна форма усуває транзитивні**

залежності. Транзитивна залежність виникає тоді, коли неключовий атрибут залежить від іншого неключового атрибута, а не безпосередньо від первинного ключа.

Інші нормальні форми: Крім трьох вищезазначених нормальних форм, існують вищі нормальні форми, такі як четверта нормальна форма (4NF) і п'ята нормальна форма (5NF), які можуть застосовуватися до більш складних наборів даних. Ці вищі нормальні форми спрямовані на подальше зменшення надмірності та залежності в базі даних.

- **4NF (четверта нормальна форма)**

Вирішує багатозначні залежності. Це гарантує відсутність кількох незалежних багатозначних фактів про сутність у записі.

- **5NF (п'ята нормальна форма)**

Також відома як «звичайна форма проекції-об’єднання» (PJNF), стосується реконструкції інформації з менших, по-різному впорядкованих фрагментів даних.

Практичні переваги нормалізації таблиць

Нормалізація даних дає кілька практичних переваг для управління базами даних і цілісності даних. Деякі з основних переваг включають:

- **Послідовність даних.** Зменшуючи надмірність, нормалізація даних допомагає підтримувати єдину, точну версію кожного фрагмента даних. Це забезпечує, що оновлення та зміни даних відображаються послідовно в усій базі даних.
- **Ефективні зміни бази даних.** Нормалізовані дані легше змінювати та обновлювати, оскільки зміни потрібно застосовувати лише в одному місці. Це спрощує процес внесення змін до структури бази даних або даних, зменшуючи ризик невідповідностей або помилок.
- **Спрощені запити.** Нормалізовані бази даних сприяють ефективним і керованим запитам. З добре структурованими даними менше необхідності в складних об'єднаннях і перетвореннях, що призводить до поліпшення продуктивності запитів.
- **Покращена цілісність даних.** Нормалізація даних зменшує ймовірність виникнення **аномалій** під час маніпуляцій даними, таких як операції вставки, оновлення та видалення. Це допомагає підтримувати цілісність і надійність даних, що зберігаються в базі даних.

Поради для ефективного використання нормалізації таблиць

1. **Зрозумійте рівні нормалізації.** Ознайомтеся з різними рівнями нормалізації, такими як 1NF, 2NF, 3NF та вищі нормальні форми. Зрозумійте принципи кожного рівня та застосовуйте їх відповідно до конкретних вимог бази даних.
2. **Консолідуйте та рефакторіть.** Регулярно перевіряйте та рефакторіть структури бази даних для забезпечення дотримання принципів нормалізації. Консолідуючи та рефакторячи таблиці, ви можете усунути надмірність і підвищити цілісність даних.
3. **Оцінюйте вплив на продуктивність.** Хоча нормалізація є важливою для збереження цілісності даних, важливо збалансувати це з впливом на продуктивність у разі надмірної

нормалізації. Оцінюйте продуктивність бази даних і враховуйте компроміс між цілісністю даних і оптимізацією продуктивності.

Висновки. Нормалізація даних – це процес, що включає структурування бази даних для мінімізації надмірності і залежності, що призводить до покращення цілісності даних і більш ефективної маніпуляції даними. Дотримуючись набору правил нормалізації, таких як перша, друга і третя нормальні форми, бази даних можуть бути організовані таким чином, щоб зменшити надмірність даних і забезпечити правильне управління залежностями даних. Практичні переваги нормалізації даних включають послідовність даних, спрощені зміни бази даних, спрощені запити і покращену цілісність даних. Для ефективного впровадження нормалізації даних важливо мати чітке розуміння рівнів нормалізації, регулярно переглядати і рефакторити структури бази даних та враховувати вплив надмірної нормалізації на продуктивність.

2.2. Денормалізація

Денормалізація – це техніка оптимізації бази даних, яка передбачає свідоме введення надмірності в дизайн бази даних. Цей процес спрямований на покращення продуктивності операцій отримання даних шляхом зменшення складності схеми бази даних.

Традиційно, у нормалізованій базі даних, дані організовані в декілька пов'язаних таблиць, щоб мінімізувати надмірність і залежність. Однак це може призвести до уповільнення виконання запитів, особливо при обробці складних приєднань і агрегування.

Денормалізація вирішує це шляхом консолідації даних з кількох таблиць в одну таблицю, зменшуючи потребу в складних приєднаннях і прискорюючи обробку запитів. Подвоюючи певні елементи даних, денормалізація спрямована на досягнення балансу між ефективністю зберігання та продуктивністю запитів.

Переваги денормалізації

Включивши денормалізацію в дизайн бази даних, можна досягти кількох переваг:

1. **Покращена продуктивність запитів**. Денормалізовані бази даних зазвичай пропонують швидший час обробки запитів через мінімізовану потребу в складних приєднаннях і агрегуваннях. Це може призвести до покращення часу відповідей для кінцевих користувачів та додатків, які залежать від бази даних.

2. **Спрощене отримання даних**. Завдяки денормалізації, операції отримання даних можуть бути спрощені, оскільки інформація з декількох таблиць консолідується в одну таблицю. Це підвищує зручність запитів та зменшує складність, яка пов'язана з отриманням даних з нормалізованої бази даних.

3. **Зменшена складність.** Денормалізація зменшує складність схеми бази даних, усуваючи потребу у надмірних приєднаннях та зв'язках між таблицями. Це може зробити базу даних легшою для розуміння, підтримки та модифікації.

4. **Покращена продуктивність для часто використовуваних даних.** Завдяки вибірковій денормалізації таблиць, які часто запитуються або потребують покращеної продуктивності, адміністратори баз даних можуть оптимізувати систему для специфічних випадків використання. Це може призвести до швидшого доступу до даних для критичних областей додатка.

Недоліки денормалізації

Хоча денормалізація має кілька переваг, важливо враховувати потенційні недоліки перед впровадженням цієї техніки:

1. **Збільшені вимоги до зберігання**. Денормалізація вводить надмірність шляхом подвоєння певних елементів даних, що може призвести до збільшення вимог до зберігання. Це може вплинути на загальне використання дискового простору, особливо при роботі з великими базами даних або наборами даних.

2. **Непослідовність даних**. Введення надмірності через денормалізацію може призвести до непослідовності даних, якщо не управляти належним чином. Оскільки дубльовані дані зберігаються в кількох місцях, будь-які оновлення дубльованих даних повинні бути ретельно синхронізовані для збереження послідовності по всій базі даних.

3. **Важкість модифікації схеми**. Денормалізовані бази даних можуть бути складнішими для модифікації та підтримки порівняно з нормалізованими базами даних. Зміни у схемі бази даних вимагають оновлення у кількох місцях, що може збільшити складність і потенційний ризик помилок.

Найкращі практики для денормалізації

Щоб забезпечити успішне впровадження денормалізації, зверніть увагу на наступні найкращі практики:

1. **Проведіть оцінку продуктивності**. Перед денормалізацією бази даних важливо оцінити конкретні потреби системи у продуктивності. Не всі бази даних потребують денормалізації, і рішення про денормалізацію повинно базуватися на ретельному аналізі проблем продуктивності.

2. **Стратегічна денормалізація**. Використовуйте денормалізацію помірно і стратегічно. Сконцентруйтесь на таблицях, які часто запитуються і потребують покращеної продуктивності. Цілком націлюючись на конкретні області бази даних, ви можете мінімізувати потенційні недоліки, при цьому максимізуючи переваги денормалізації.

3. **Моніторинг послідовності даних**. Встановіть надійний процес для підтримки послідовності даних у денормалізованій базі даних. Це включає впровадження відповідних механізмів для синхронізації оновлень та змін дубльованих елементів даних. Регулярні аудити та валідації можуть допомогти забезпечити цілісність даних по всій системі.

4. **Розгляньте індексацію.** Поряд з денормалізацією, розгляньте можливість впровадження технік індексації бази даних для додаткової оптимізації продуктивності запитів. Індексація може прискорити отримання даних, створюючи структуровані індекси в базі даних, що дозволяє швидший доступ до конкретних елементів даних.

   Індексація бази даних. Техніка, що використовується для покращення швидкості операцій отримання даних шляхом створення структурованих індексів в базі даних.

Індексація дозволяє швидший доступ до конкретних елементів даних, зменшуючи час, необхідний для обробки запитів.

2.3. Аномалії в базах даних

Аномалія в базах даних означає будь-яке відхилення від очікуваного шаблону або поведінки даних, що часто вказує на помилки, шахрайство або загрози безпеці. Ці аномалії можуть виникати в різних формах, таких як непослідовності, аномальні значення або неочікувані зміни значень даних.

Аномалії в базах даних можуть мати суттєві наслідки, починаючи від пошкодження й втрати даних до порушення безпеки й фінансового шахрайства. Важливо для організацій розуміти, як виникають аномалії, та запроваджувати превентивні заходи для мінімізації їх впливу.

Як працюють аномалії в базах даних

Аномалії в базах даних можуть проявлятися різними способами і класифікуються на три основні категорії: непослідовності, аномальні значення та неочікувані зміни.

Непослідовності

Непослідовності стосуються записів даних, які не відповідають встановленим правилам або обмеженням. Наприклад, якщо поле повинно містити лише позитивні числа, наявність негативного значення вважається непослідовністю. Непослідовності можуть виникати через помилки вводу даних, програмні помилки або помилкові процеси інтеграції. Вони можуть викликати проблеми, такі як пошкодження даних, неправильні обчислення та неточні звіти.

Аномальні значення

Аномальні значення – це точки даних, які значно відрізняються від більшої частини набору даних. Ці значення зазвичай вважаються незвичайними або ненормальними порівняно з рештою даних. Аномальні значення можуть бути наслідком помилок вводу даних, збоїв датчиків або навмисної підробки. Виявлення аномальних значень важливе, оскільки вони можуть спотворити статистичний аналіз, впливати на процес прийняття рішень та призводити до неправильних висновків чи прогнозів.

Неочікувані зміни

Неочікувані зміни в даних стосуються раптових або невідомих змін у базі даних. Ці зміни можуть включати несанкціоновані оновлення, видалення або додавання даних. Такі модифікації можуть вказувати на порушення безпеки, зловмисні дії або пошкодження бази даних. Важливо, щоб організації мали надійні механізми для виявлення та запобігання несанкціонованим змінам у своїх базах даних.

Поради щодо запобігання аномалій

Щоб захиститися від аномалій у базах даних, організаціям слід впроваджувати проактивні заходи, які включають наступні:

Регулярний моніторинг

Впроваджуйте автоматизовані системи для безперервного моніторингу активності баз даних і виявлення аномалій у реальному часі. Ці системи можуть використовувати такі

техніки, як алгоритми машинного навчання, моделі виявлення аномалій та статистичний аналіз для виявлення ненормальної поведінки або шаблонів даних. Швидке виявлення аномалій дозволяє організаціям вживати негайних заходів для розслідування та мінімізації потенційних ризиків.

Встановлення базових значень

Визначайте нормальні шаблони даних та встановлюйте пороги для прийнятних відхилень, щоб швидко виявляти потенційні аномалії. Встановлення базових значень дозволяє організаціям виявляти відхилення від очікуваної поведінки, незалежно від того, чи є це непослідовностями, аномальними значеннями або неочікуваними змінами. Це дозволяє здійснювати своєчасне втручання, знижуючи вплив аномалій і мінімізуючи ризик втрати даних чи шахрайства.

Контроль доступу

Обмежуйте доступ до баз даних для авторизованого персоналу та запроваджуйте заходи для виявлення несанкціонованих спроб модифікації даних. Надійні механізми контролю доступу, такі як рольовий контроль доступу та протоколи сильної аутентифікації, є важливими для запобігання несанкціонованому доступу до баз даних. Крім того, впровадження механізмів аудиту та моніторинг привілеїв можуть допомогти ідентифікувати підозрілі дії та потенційні загрози.

Валідація даних

Використовуйте сильні методи валідації та перевірки введених даних, щоб забезпечити точність та цілісність вхідних даних. Механізми валідації даних, такі як перевірка типу даних, діапазону та формату, можуть допомогти виявляти непослідовності та запобігати введенню недійсних даних у базу даних. Регулярні оцінки якості даних і процеси валідації сприяють підтримці точності та надійності даних.

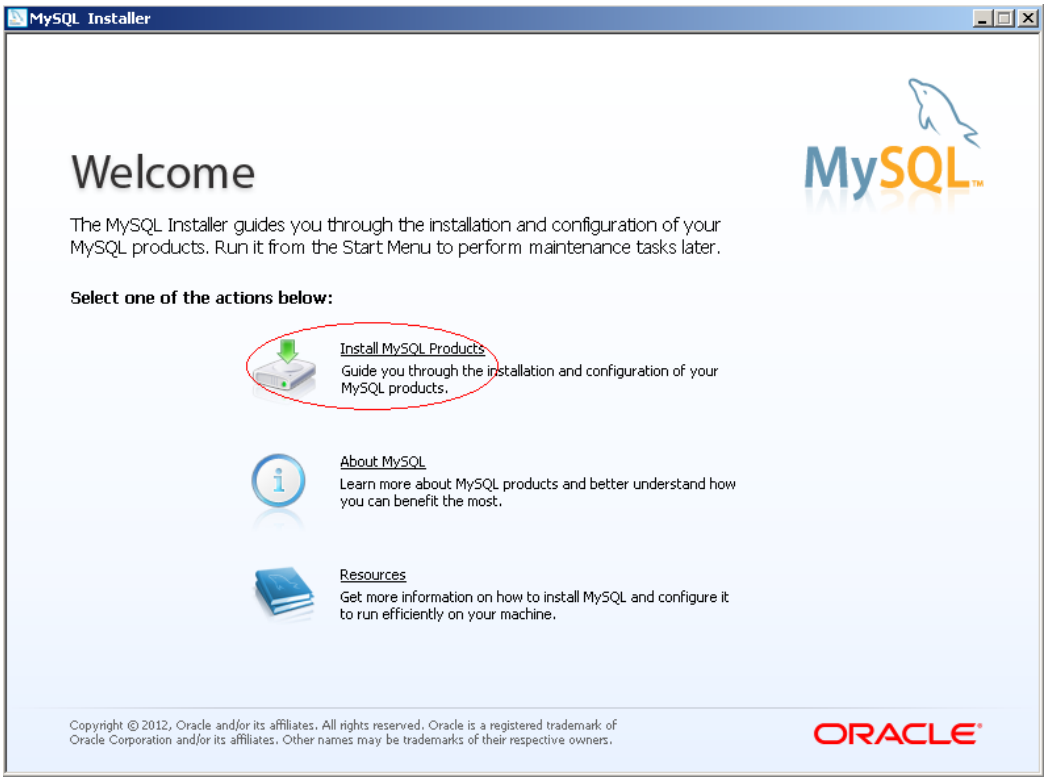
3. Установка MySQL

Розглянемо встановлення пакету програм для Windows, що містить MySQL server, середовище для розробки та адміністрування MySQL Workbench та багато інших корисних компонентів. (Джерело - http://www.mysql.com/downloads/installer/)

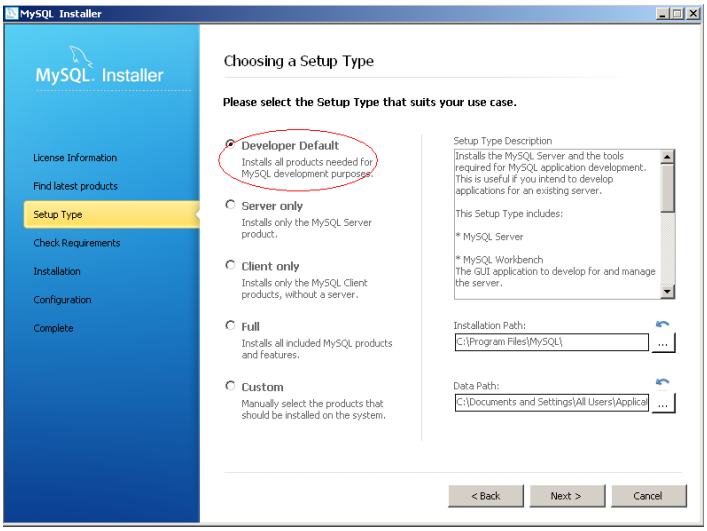
1. Для початку потрібно встановити Microsoft .NET 4.0 Framework, якщо його немає.

2. Встановити Visual C++ Redistributable Packages for Visual Studio 2013.

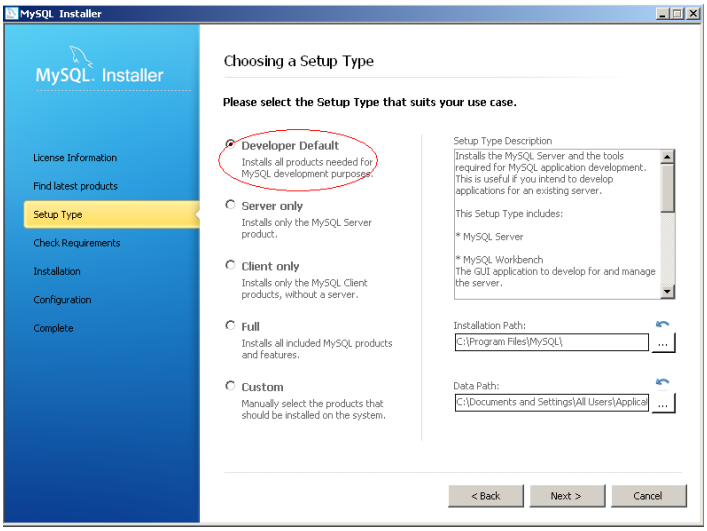
Тепер можна встановлювати MySQL - mysql-installer-community-5.5.28.3.msi або новішу версію:



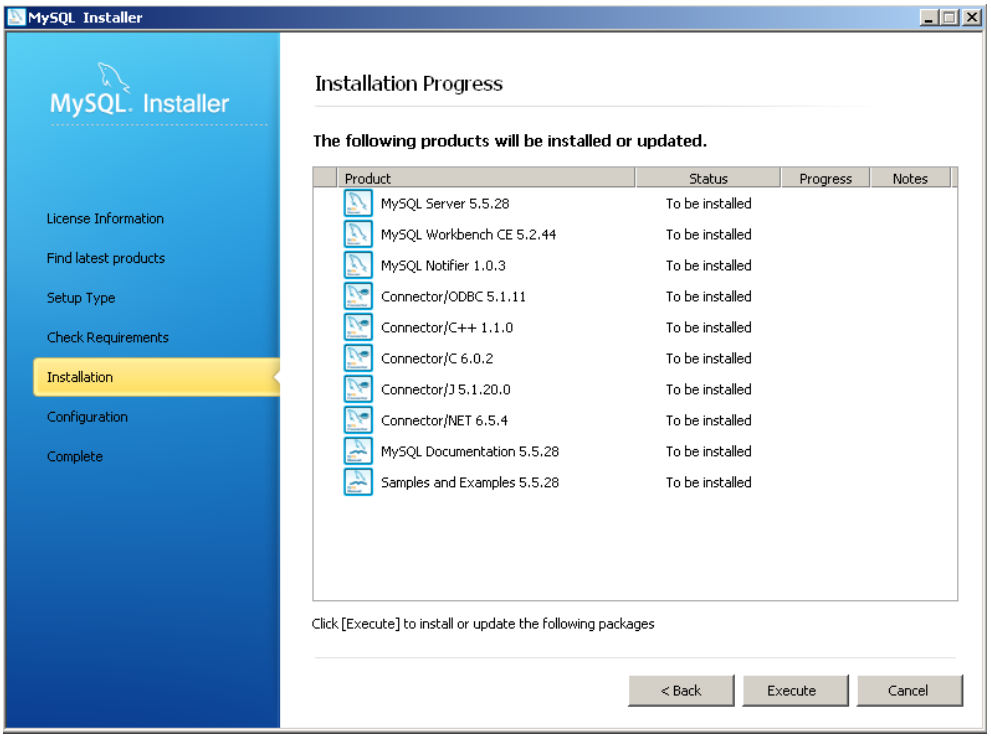
На 2 сторінці приймаємо ліцензію. На 3 сторінці не перевіряємо поновлення (skip the check for updates).



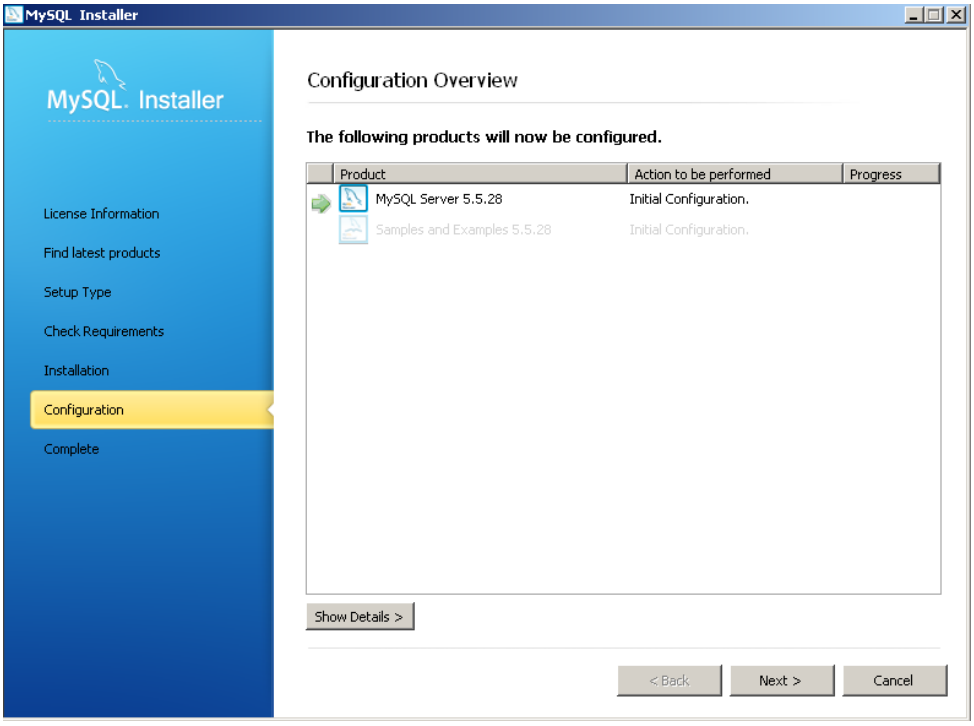
Перевіряються необхідні умови для встановлення:



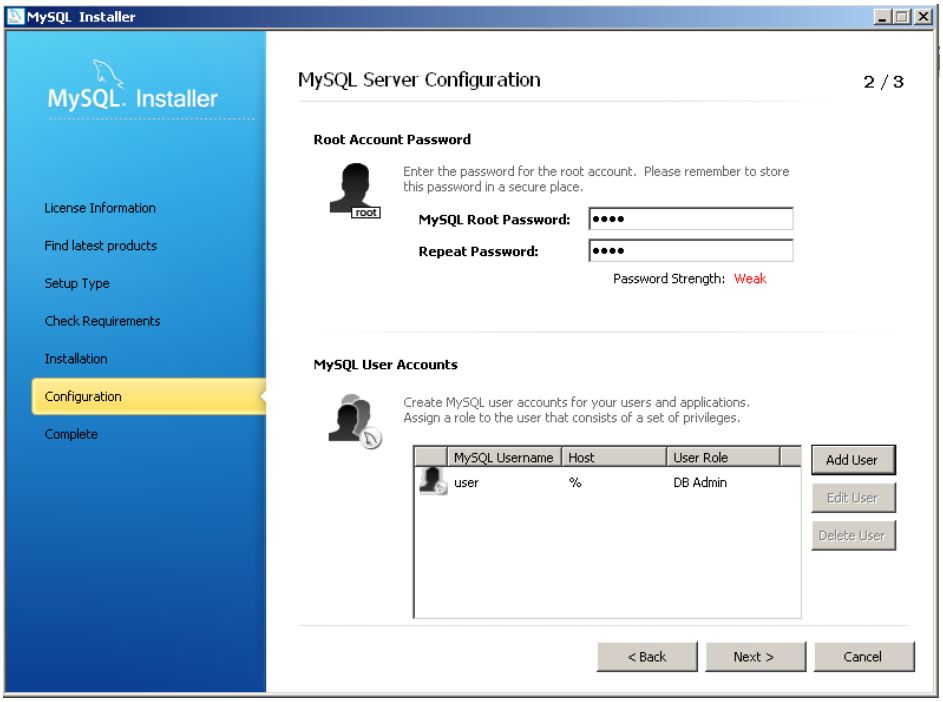
Встановлюємо такі продукти: (перший пункт – це сервер, другий пункт – середовище розробника)



Конфігурування сервера:



Для адміністратора на ім'я root задамо пароль (запам'ятайте його!) А також створимо користувача на ім'я user з паролем (теж запам'ятайте його!) (можна зробити паролі, що збігаються з ім'ям користувача: root і user, хоча в сенсі безпеки це поганий варіант, але для навчальних цілей годиться).



На наступних сторінках нічого не змінюємо. Ім'я сервера за промовчанням MySQL55. Встановлення успішно завершено.

4. Використання MySQL Workbench для роботи з БД

На даний момент існує багато систем управління реляційними базами даних. Приклади реляційних баз даних: MySQL, Microsoft SQL Server, Microsoft Access, Oracle, DB2 тощо.

MySQL є найбільш популярною СУБД. MySQL це реляційна база даних з відкритим кодом. MySQL є кросплатформним, що означає, що він працює на кількох різних платформах, таких як Windows, Linux і Mac OS.

MySQL підтримує кілька механізмів зберігання, кожен зі своїми специфікаціями, тоді як інші системи, такі як SQL Server, підтримують лише один механізм зберігання. Розглянемо два механізми зберігання, які підтримуються MySQL.

InnoDB: – його система зберігання даних за умовчанням MySQL починаючи з версії 5.5. InnoDB підтримує зовнішні ключі для посилальної цілісності, а також підтримує стандартні транзакції ACID.

MyISAM: – це був механізм зберігання за замовчуванням для MySQL до версії 5.5. MyISAM не підтримує транзакції. Його переваги перед InnoDB включають простоту та високу продуктивність.

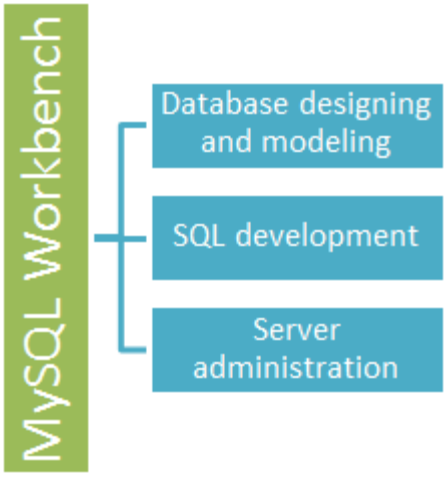
MySQL має високу продуктивність порівняно з іншими системами реляційних баз даних. Це пов'язано з простотою конструкції та підтримкою механізмів зберігання даних.

Економічно ефективний, відносно дешевший у порівнянні з іншими реляційними базами даних. Фактично видання спільноти є безкоштовним. Комерційне видання має ліцензійну плату, яка також економічно ефективна порівняно з ліцензійною платою за такі продукти, як Microsoft SQL Server.

Кросплатфор ма – MySQL працює на багатьох платформах, що означає, що його можна розгорнути на більшості машин. Інші системи, такі як MS SQL Server, працюють лише на платформі Windows.

MySQL Workbench

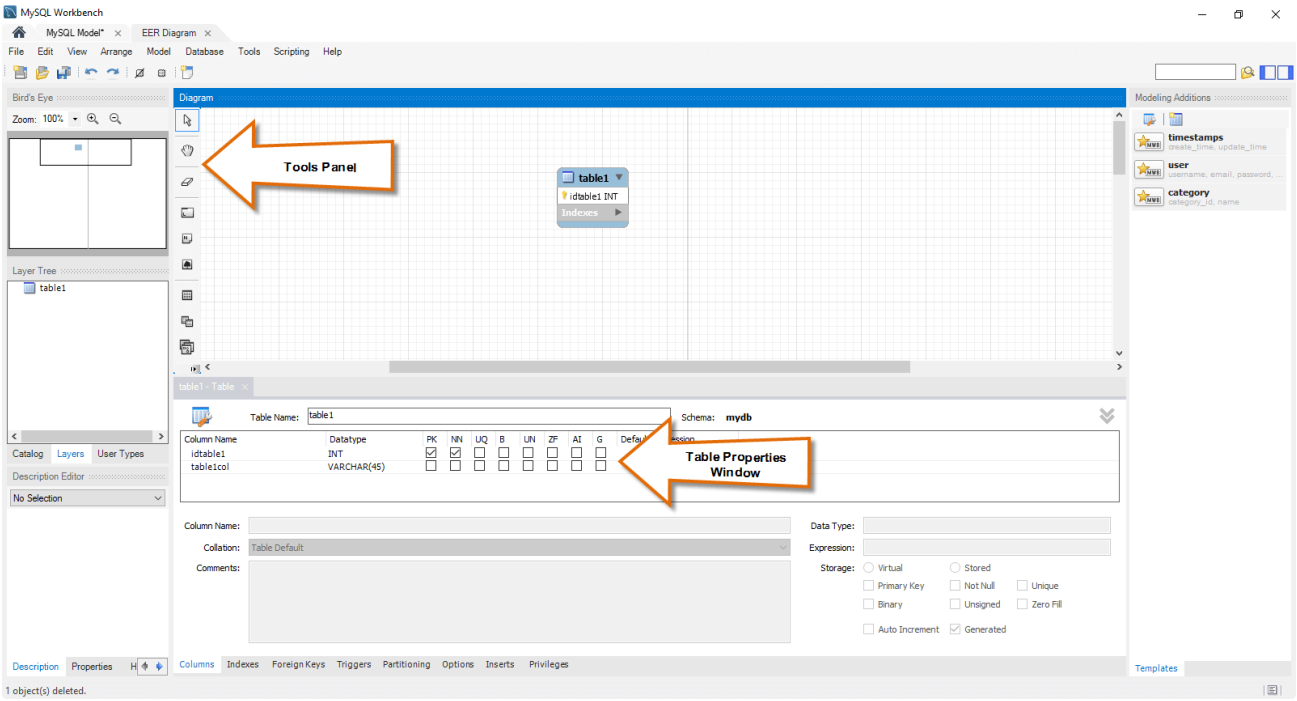
Мета MySQL workbench — надати інтерфейс для більш легкої та структурованої роботи з базами даних- Workbench дає можливість для в**ізуального проектування та моделювання баз даних,** полегшує створення нових фізичних моделей даних і модифікацію існуючих MySQL бази даних із зворотним/прямим проектуванням і функціями керування змінами.



MySQL Workbench – Інструмент моделювання та проектування

- Моделі є основою більшості дійсних і високопродуктивних баз даних. MySQLWorkbench має інструменти, які дозволяють розробникам і адміністраторам баз даних візуально створювати фізичні моделі дизайну бази даних, які можна легко перевести на MySQL бази даних з використанням передової інженерії.
- MySQL Workbench підтримує створення кількох моделей в одному середовищі.
- Він підтримує всі об'єкти, такі як таблиці, подання, збережені процедури, тригери тощо, які складають базу даних.
- MySQL Workbench має вбудовану утиліту перевірки моделі, яка повідомляє розробнику даних про будь-які проблеми, які можуть бути виявлені.

На малюнку нижче показано вікно моделювання для MySQL Workbench.

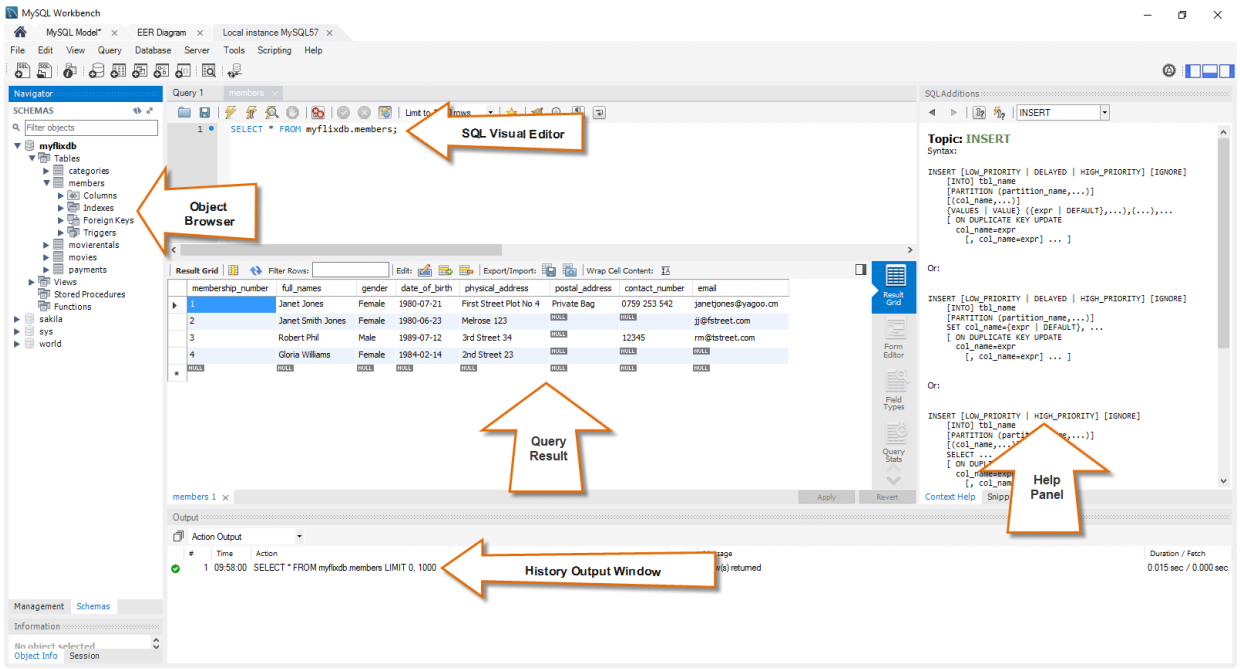


MySQL Workbench – засіб розробки SQL-кодів

Мова структурованих запитів (SQL) дозволяє нам маніпулювати нашими реляційними базами даних. SQL лежить в основі всіх реляційних баз даних.

- MySQL Workbench, має вбудований візуальний редактор SQL.
- Редактор Visual SQL дозволяє розробникам створювати, редагувати та запускати запити MySQL серверні бази даних. Він має утиліти для перегляду даних та їх експорту.
- Підсвічувачі кольорів синтаксису допомагають розробникам легко писати та налагоджувати оператори SQL.
- Можна виконувати кілька запитів, а результати автоматично відображати на різних вкладках.
- Запити також зберігаються на панелі історії для подальшого пошуку та запуску.

На малюнку нижче показано вікно розробки SQL для MySQL Workbench.



MySQL Workbench, як засіб адміністрування

Адміністрування сервера відіграє вирішальну роль у захисті даних компанії. Основними питаннями, що стосуються адміністрування сервера, є керування користувачами, конфігурація сервера, журнали сервера та багато іншого. Верстак MySQL має такі функції, які спрощують процес MySQL адміністрування сервера;

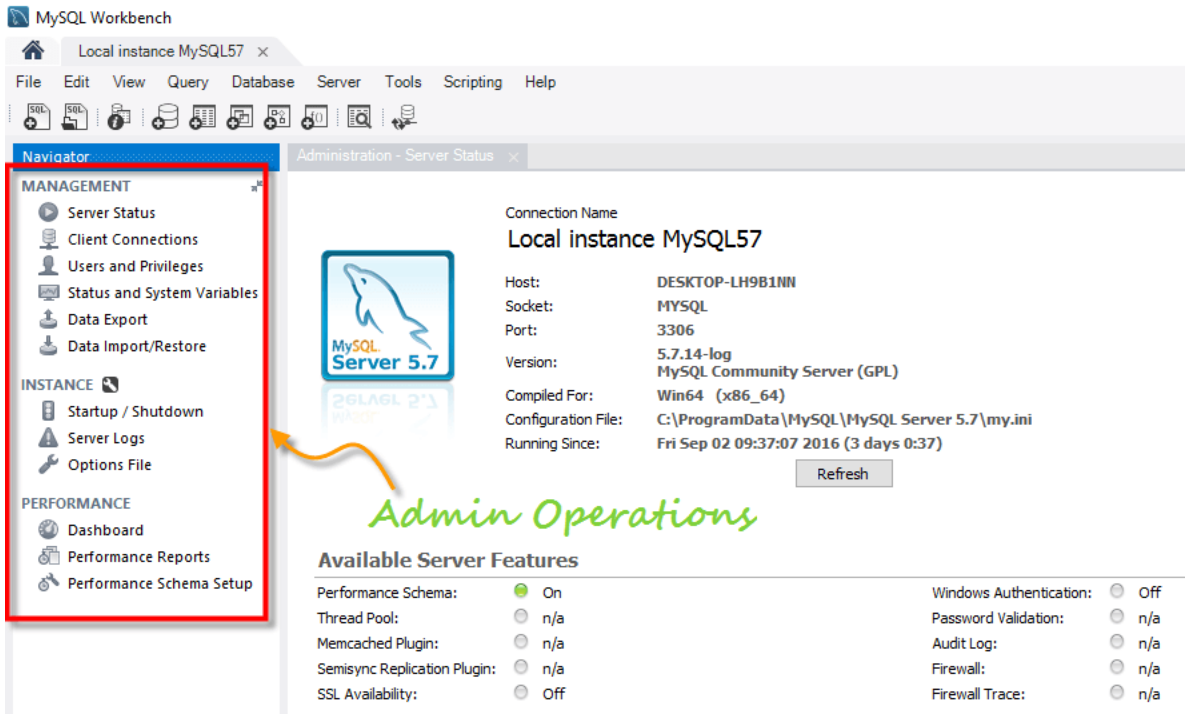
- **Адміністрація користувача** – візуальна утиліта для керування користувачами, яка дозволяє адміністраторам баз даних легко додавати нових і видаляти існуючих користувачів, якщо виникне потреба, надавати та скидати привілеї та переглядати профілі користувачів.

- **Конфігурація сервера** – дозволяє розширену конфігурацію сервера та тонке налаштування для оптимальної продуктивності.

- **Резервне копіювання та відновлення баз даних** – візуальний інструмент для експорту/імпорту MySQL дамп файлів. MySQL файли дампа містять сценарії SQL для створення баз даних, таблиць, представлень, збережених процедур і вставки даних.

- **Журнали сервера** – візуальний інструмент для перегляду MySQL журнали сервера. Журнали включають журнали помилок, двійкові журнали та журнали InnodDB. Ці журнали стануть в нагоді під час діагностики на сервері. На малюнку нижче показано вікно моделювання для MySQL Workbench.

  На малюнку нижче показано панель адміністратора для Workbench MySQL.



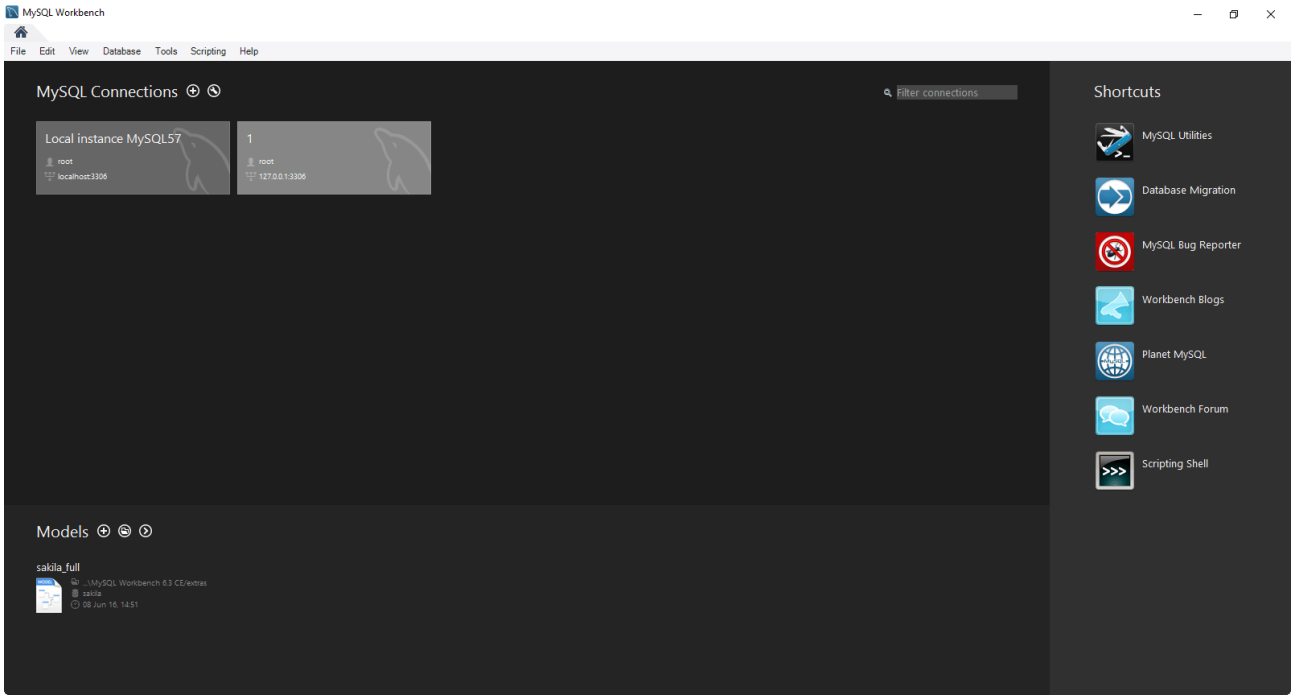
Встановлення MySQL Workbench для Windows (2-етапний процес).

1) Встановити MySQL Community Server

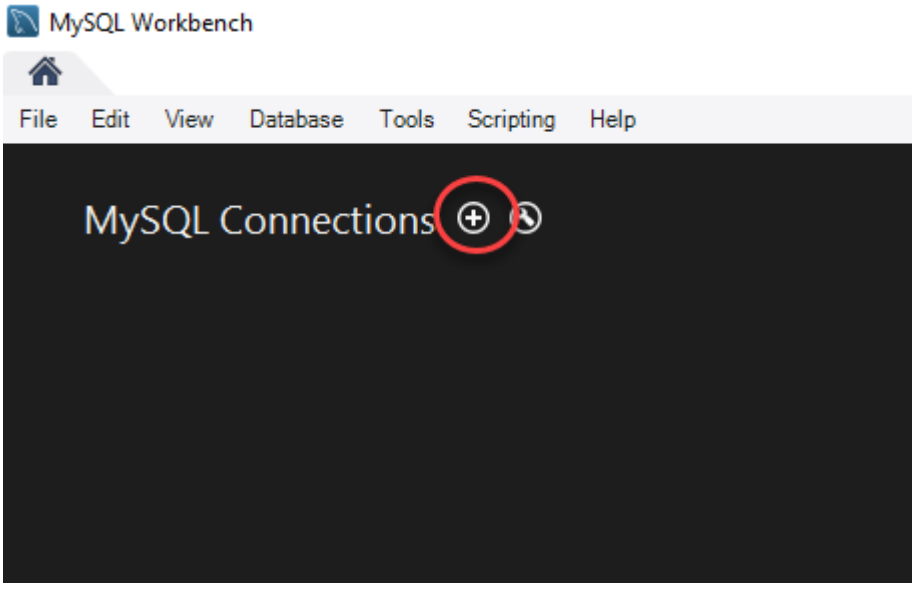
2) Встановити MySQL Workbench – Ви можете встановити Workbench за допомогою файлу zip або інсталятора msi (рекомендовано). Для встановлення вам знадобляться права адміністратора.

Після цього потрібно буде налаштувати MySQL Workbench. Нижче наведено крок за кроком процес встановлення MySQL Workbench.

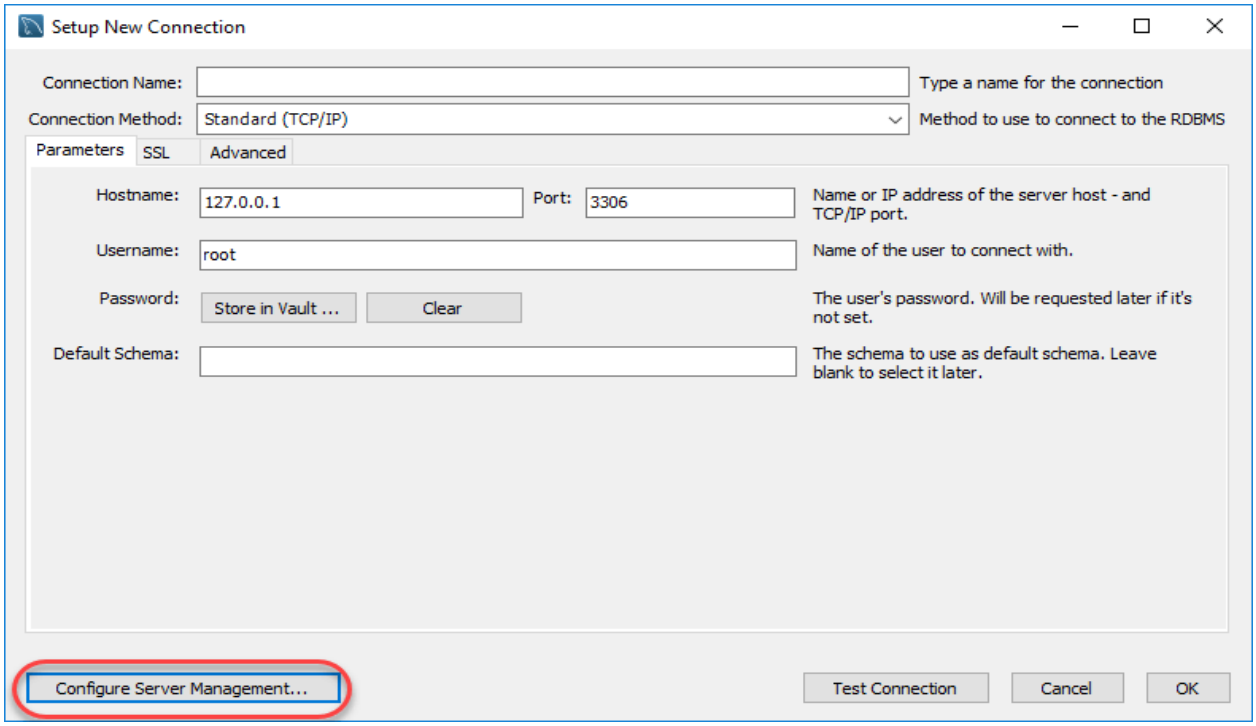
Крок 1) Відкрити головне вікно.Першим кроком є запуск Workbench MySQL. Те, що ви бачите, називається **Домашнє вікно**



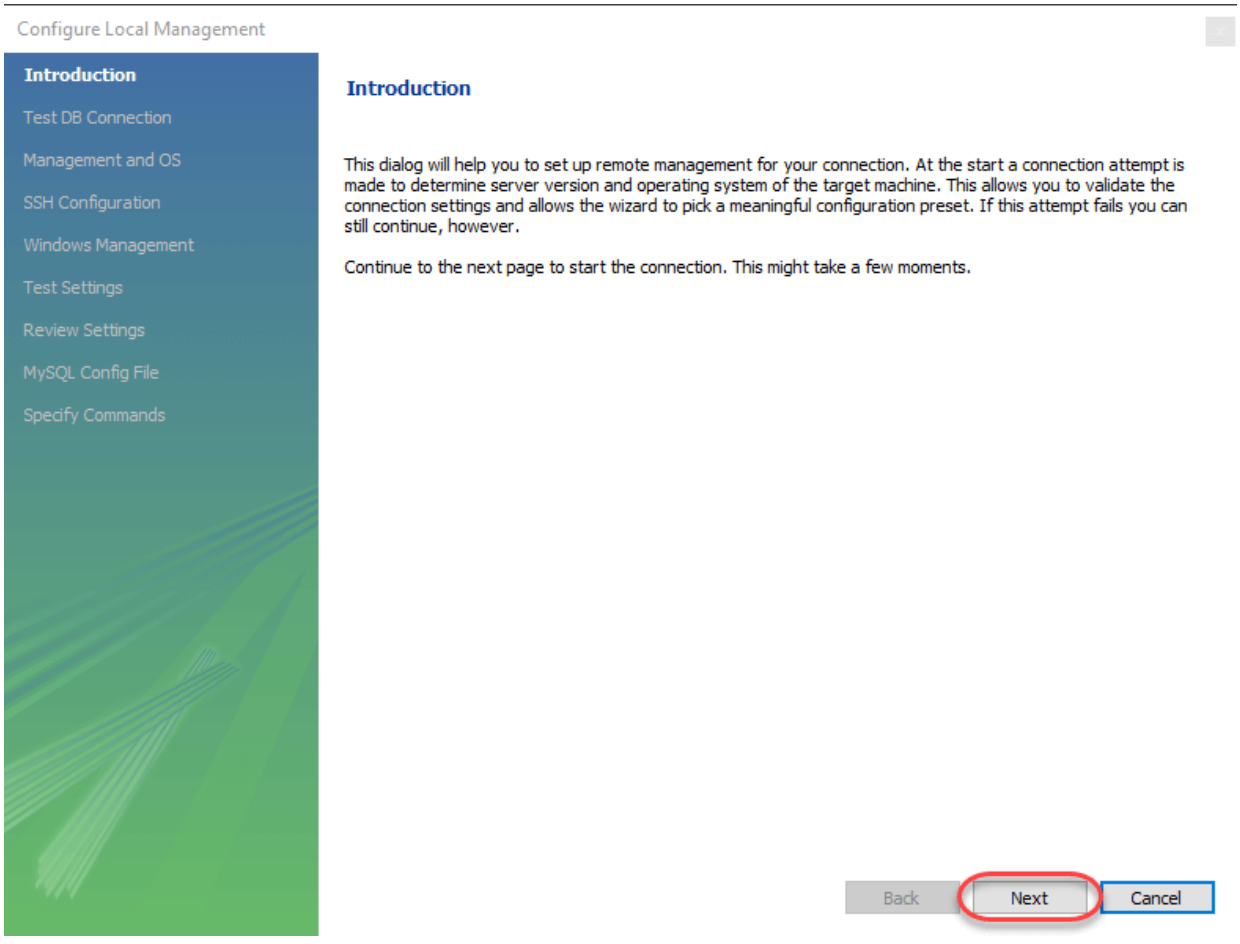
Крок 2) Відкрийте майстер нових підключень. Далі потрібно створити свій MySQL Підключення до сервера, яке містить відомості про цільовий сервер бази даних, включаючи те, як підключитися до нього. Натисніть ” + “ in MySQL Workbench Домашнє вікно. Це відкриється майстер **Налаштувати нове підключення**.



Крок 3) Натисніть кнопку «Налаштувати керування сервером». Можете створити підключення до локально встановленого сервера. Натисніть **Налаштувати керування сервером** кнопка в **Налаштувати нове підключення** вікно для перевірки конфігурації MySQL сервера.

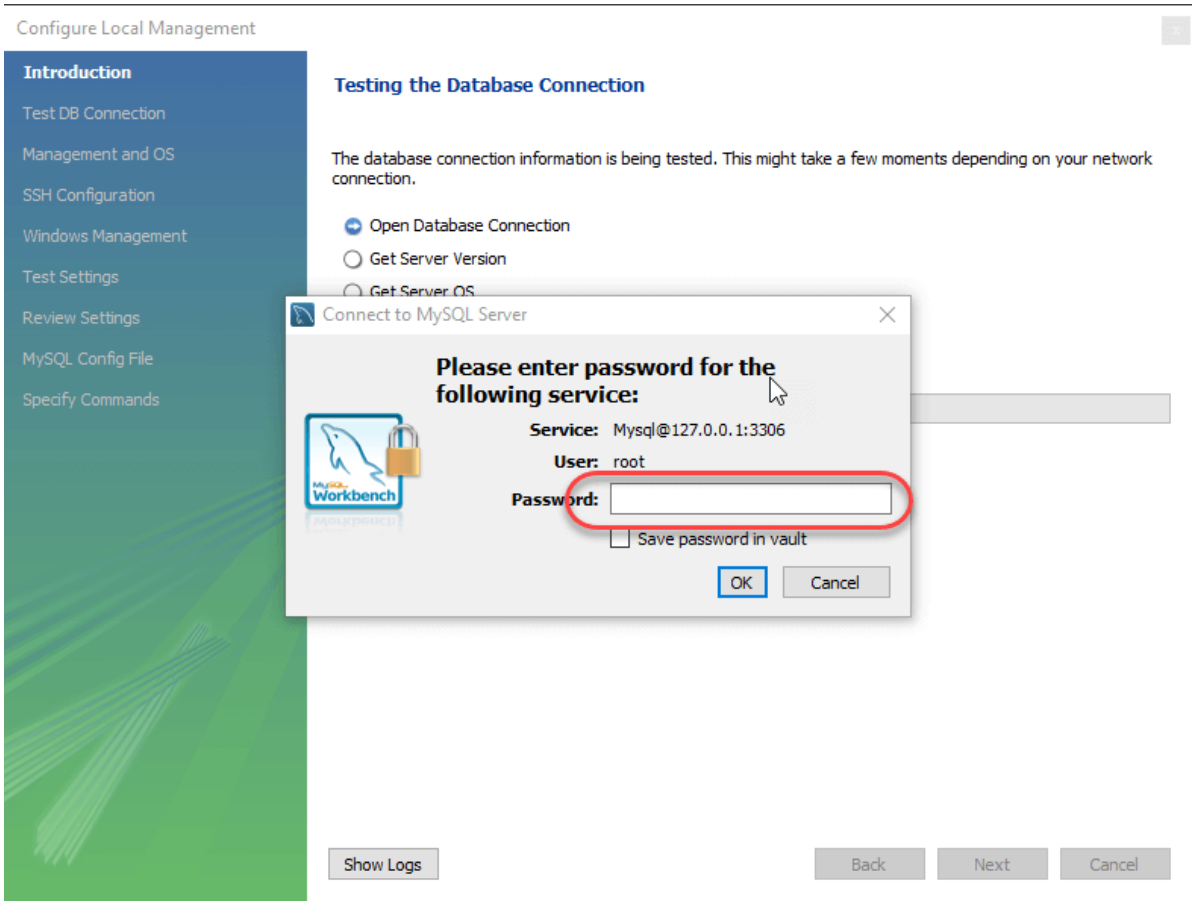


Крок 4) Натисніть кнопку **Next**, щоб продовжити
Відкриється нове вікно з назвою **Налаштувати локальне керування**. Натисніть кнопку **Next**, щоб продовжити.

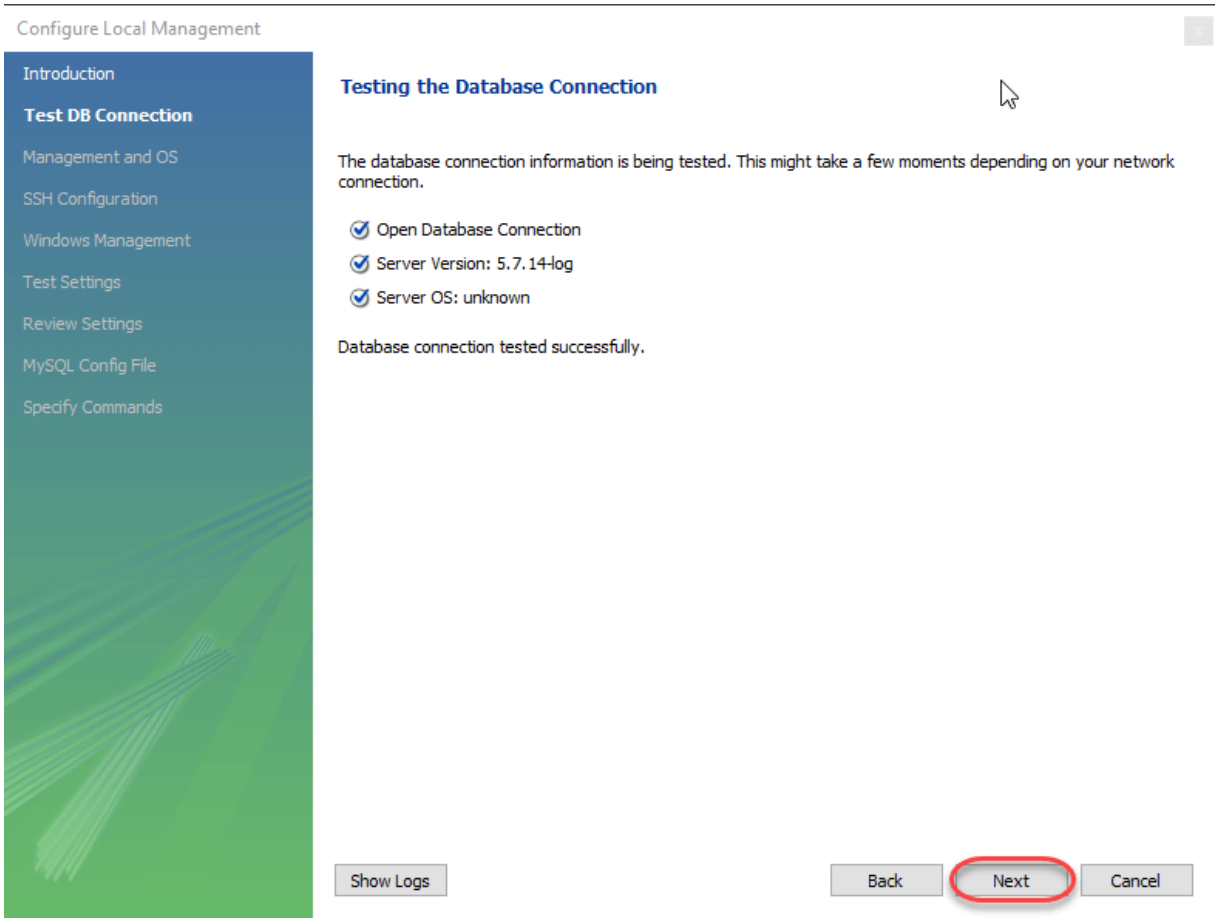


Крок 5) Введіть свій пароль і натисніть **OK**

Далі майстер перевірить підключення до бази даних. Якщо перевірка не вдається, поверніться та виправте параметри підключення до бази даних. Далі відкриється спливаюче вікно із запитом на введення пароля користувача root, щоб перевірити ваше з'єднання з екземпляром локального сервера mysql. Пароль той, який ви встановили під час встановлення MySQL Workbench. Введіть свій пароль і натисніть **OK**

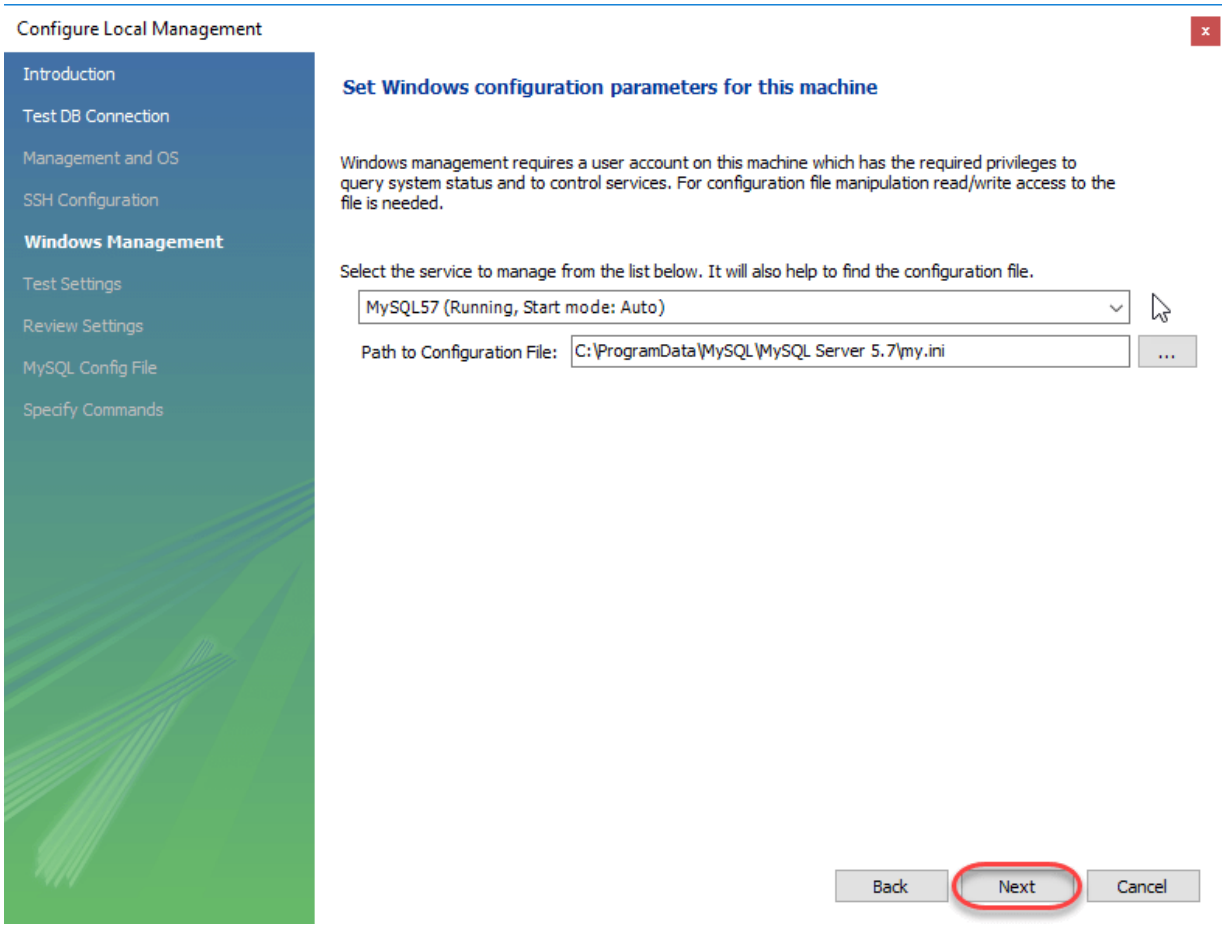


Крок 6) Натисніть «**Next**», щоб продовжити
Далі майстер перевірить підключення до бази даних. Якщо тест не вдається, поверніться та виправте параметри підключення до бази даних. Інакше, якщо всі тести пройшли успішно, натисніть «**Next**», щоб продовжити.

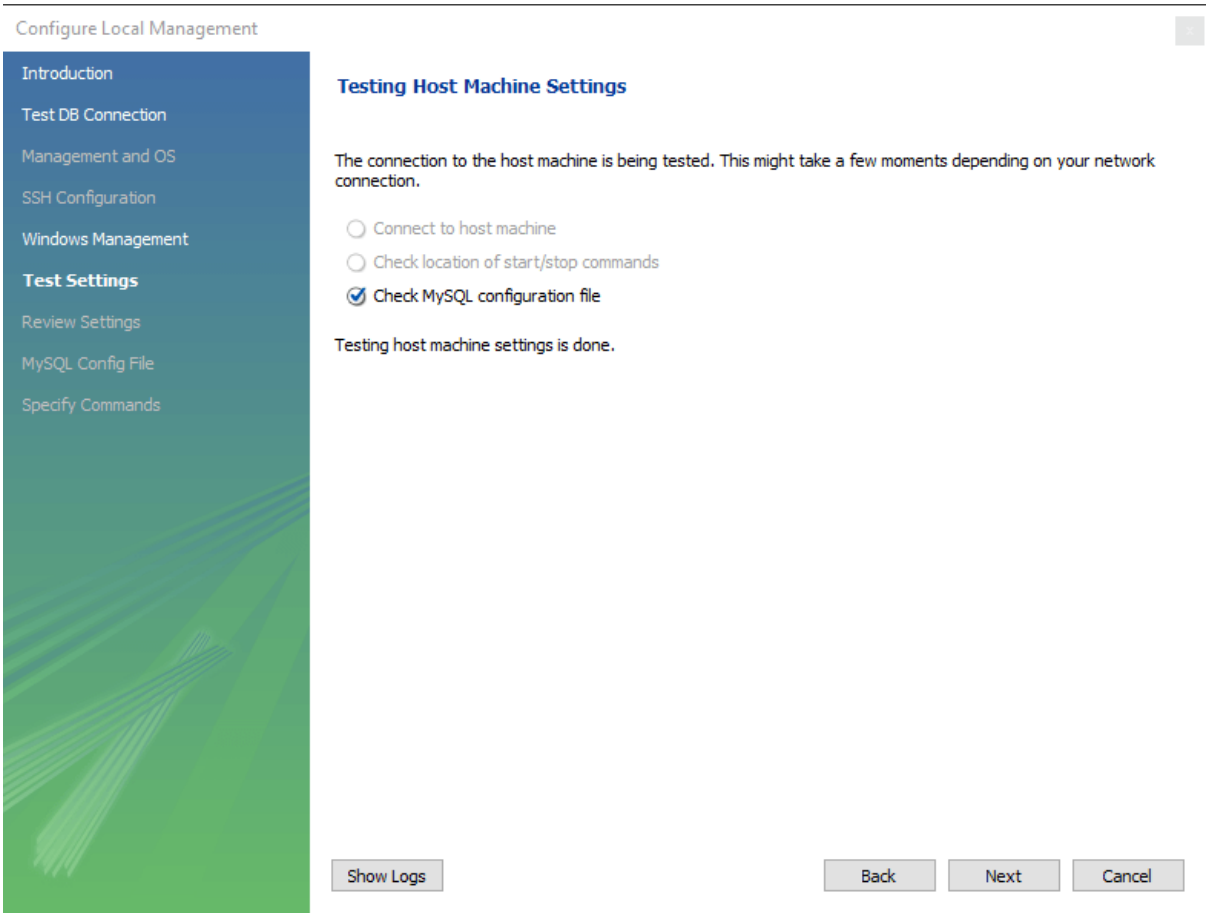


Крок 7) Натисніть **Next**

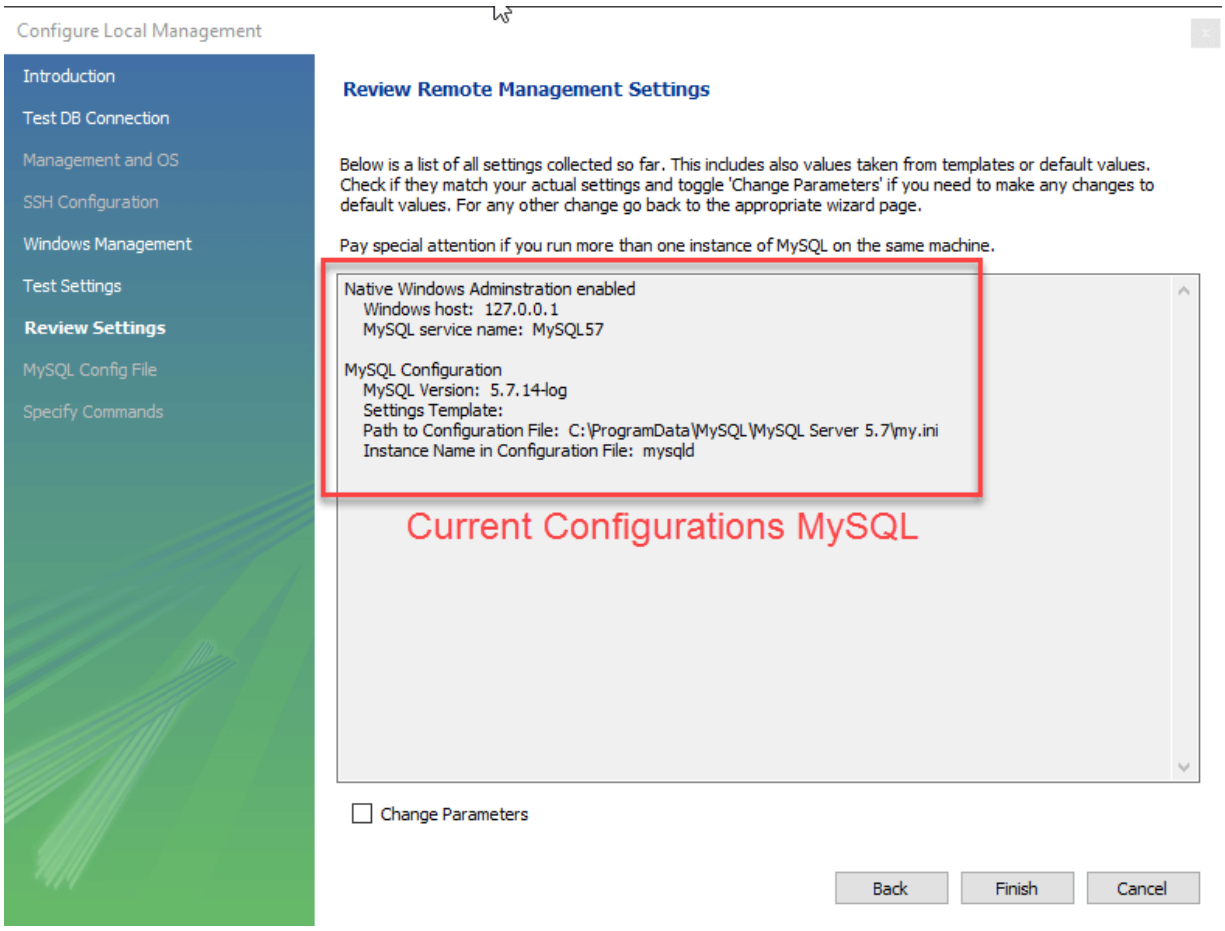
Після цього відкриється новий майстер керування локальними службами – він дозволяє перемикатися між декількома серверами mysql, встановленими на одній машині. Як новачок, ви можете обійти це й натиснути **Next** продовжувати.



Крок 8) Виберіть MySQL Файл конфігурації сервера
Потім майстер перевірить можливість доступу MySQL Файл конфігурації сервера та тестові команди запуску/зупинки.

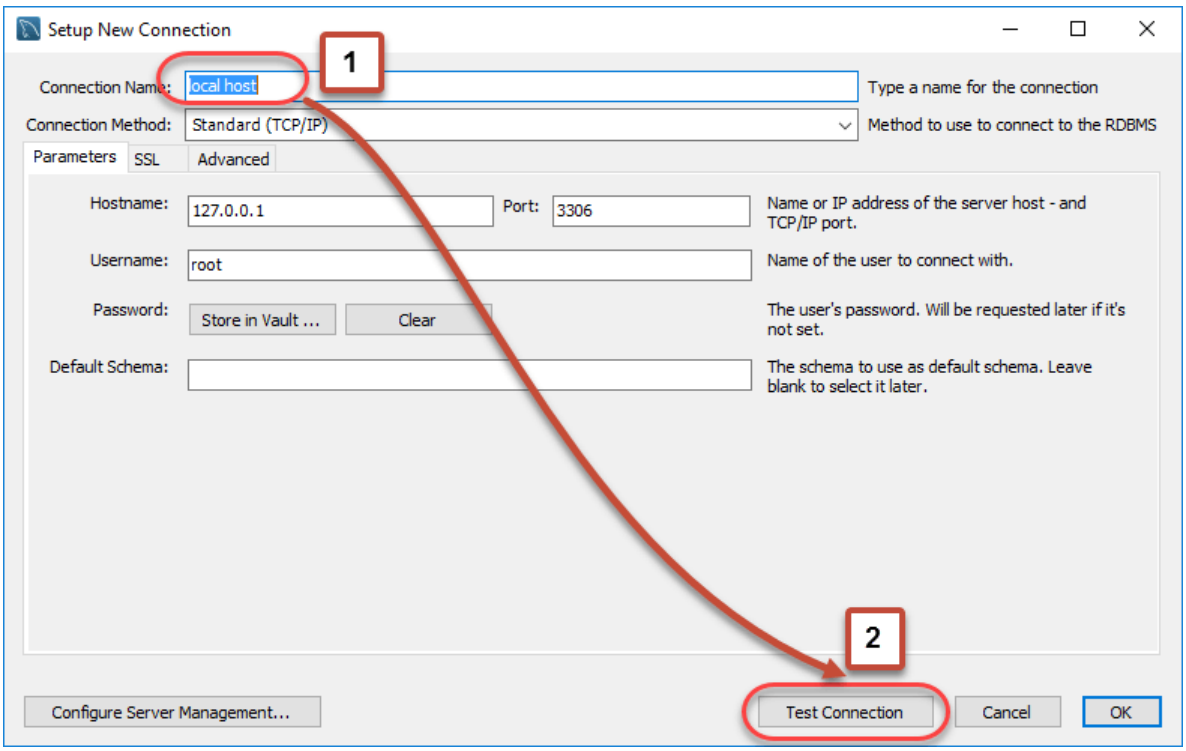


Крок 9) Натисніть «**Finish**», щоб завершити налаштування сервера
Далі ви можете переглянути поточні конфігурації. Після перегляду конфігурацій натисніть «Готово», щоб завершити налаштування сервера



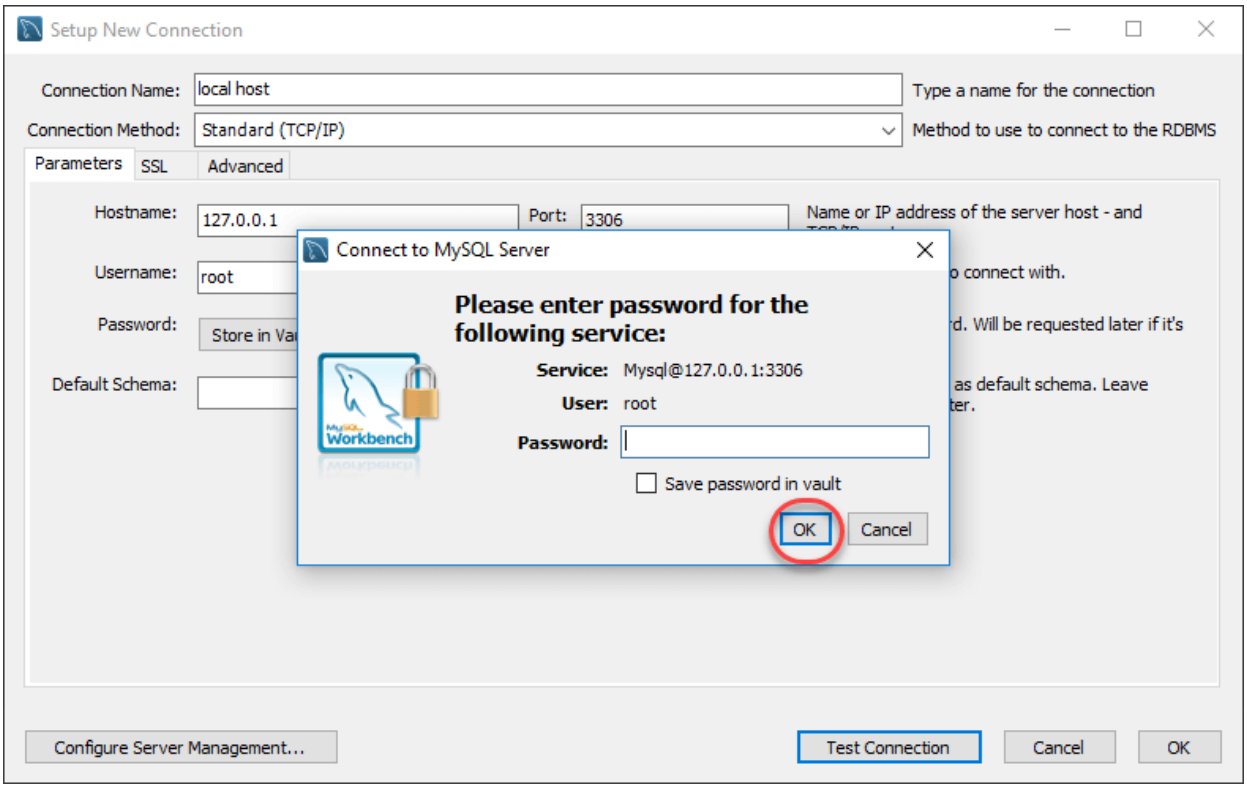
Крок 10) Натисніть **Test Connection**

Наступним кроком є налаштування підключення, яке можна використовувати для підключення до сервера. Якщо ви ще не створили підключення, ви можете використовувати надані значення за замовчуванням. Клацніть Перевірити з'єднання [2] після введення імені з'єднання [1].

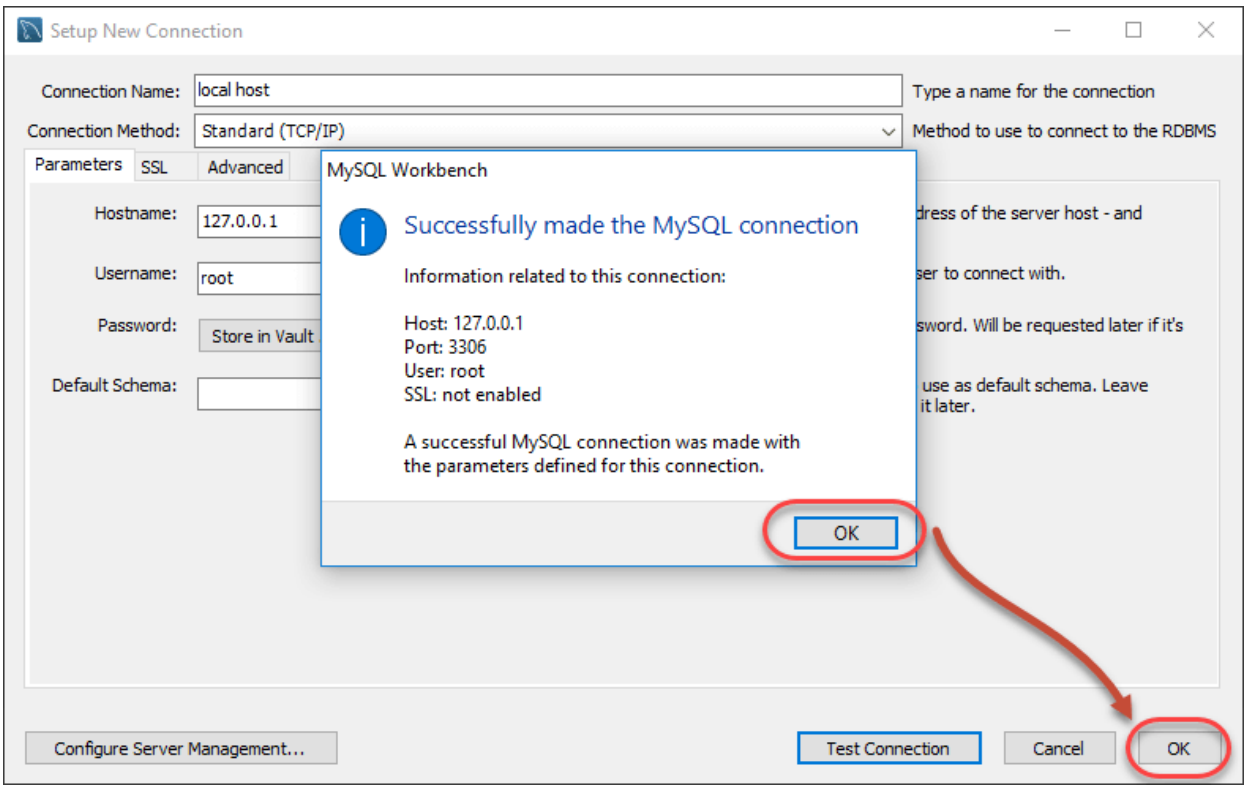


Крок 11) натисніть **OK**

Відкриється нове діалогове вікно із запитом пароля для root/вибраного користувача. Якщо ваш MySQL користувач root має пароль, ви можете ввести його за допомогою Store in Vault функція. Натисніть OK.



Якщо введений пароль для користувача правильний, відобразиться наступний екран. Натисніть на **обидві OK** кнопки, і все буде готово.

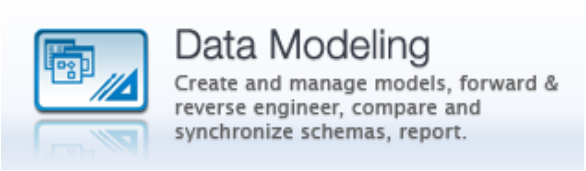


Новий екземпляр буде показано на домашній сторінці.

4.1. Робота в mySQL Workbench - Створення EER-діаграми

Середовище mySQL Workbench призначене для візуального проектування баз даних та управління сервером mySQL.

Для побудови моделей призначено секцію Data Modeling:



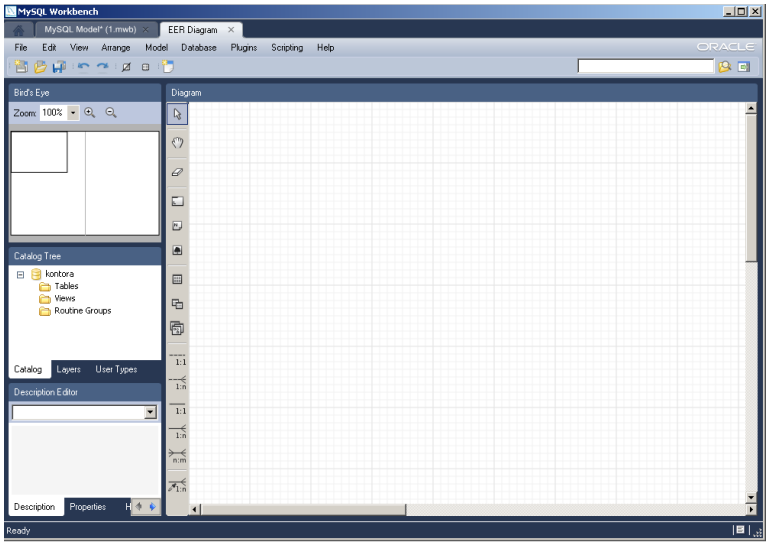
Виберемо пункт Create new EER Model.

EER model розшифровується як Extended Entity-Relationship Model та перекладається як Розширена модель сутностей-зв'язків.

За замовчанням ім'я створеної моделі myDB. Клацніть правою кнопкою миші на ім'я моделі та виберіть у меню меню Edit schema. У вікні можна змінити ім'я моделі. Назвемо її, наприклад, kontora. У іменах таблиць і шпальт не можна використовувати російські букви.

У цьому вікні також потрібно налаштувати так звану кодову сторінку. Для цього виберіть зі списку пункт "cp1251-cp1251_general_ci". Вікно властивостей можна зачинити.

Діаграму будуватимемо за допомогою візуальних засобів. Клацніть по Add diagram, завантажиться порожнє вікно діаграми:



Створити нову таблицю можна за допомогою піктограми. Потрібно клацнути по цій піктограмі, а потім клацнути в робочій ділянці діаграми. На цьому місці з'явиться таблиця під назвою table1. Подвійне клацання по цій таблиці відкриває вікно редагування, в якому можна змінити ім'я таблиці та налаштувати її структуру.

Будемо створювати таблицю Відділи з наступними стовпцями: номер_відділу, повна_назва_відділу, коротка_назва_відділу. Перейменуємо table1 на k_dept і почнемо створювати стовпці.

Кожен стовпець має:

• ім'я (не використовуйте російські літери в імені!),

• тип даних. Найпоширеніші типи даних:

o INT - ціле число;

o VARCHAR(розмір) – символьні дані змінної довжини, у дужках вказується максимальний розмір;

o DECIMAL (розмір, десяткові_знаки) - десяткове число;

o DATE – дата:

o DATETIME – дата та час.

Далі розташовуються стовпці, в яких можна налаштувати додаткові властивості поля, увімкнувши відповідний прапорець:

• PK (primary key) – первинний ключ;

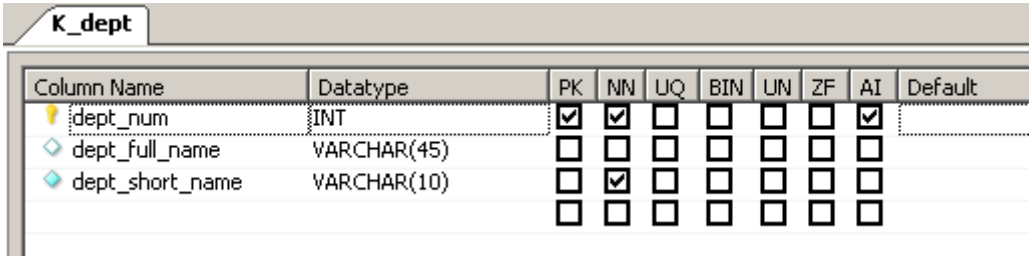
• NN (not null) – осередок не допускає порожніх значень;

• UQ (unique) – значення має бути унікальним у межах стовпця;

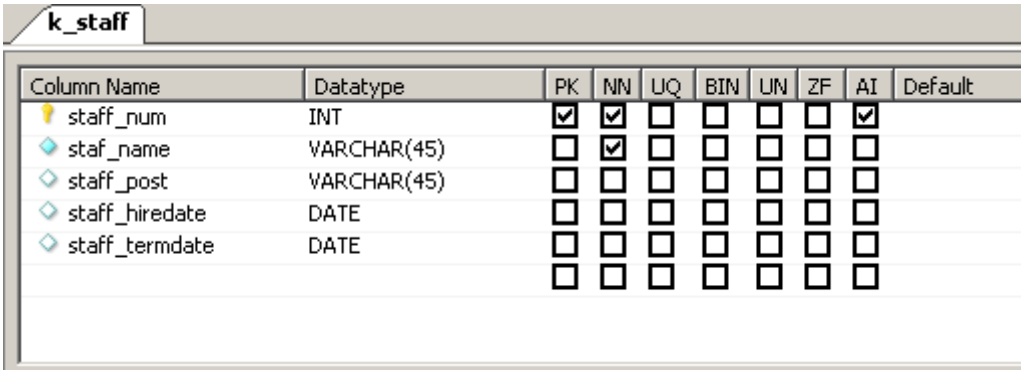
• AI (auto incremental) – це властивість корисна для простого первинного ключа, воно означає, що первинний ключ автоматично заповнюватиметься натуральними числами: 1, 2, 3, тощо;

• DEFAULT – значення за промовчанням, тобто значення, яке при додаванні нового рядка в таблицю автоматично вставляється в комірку сервером, якщо користувач залишив комірку порожньою.

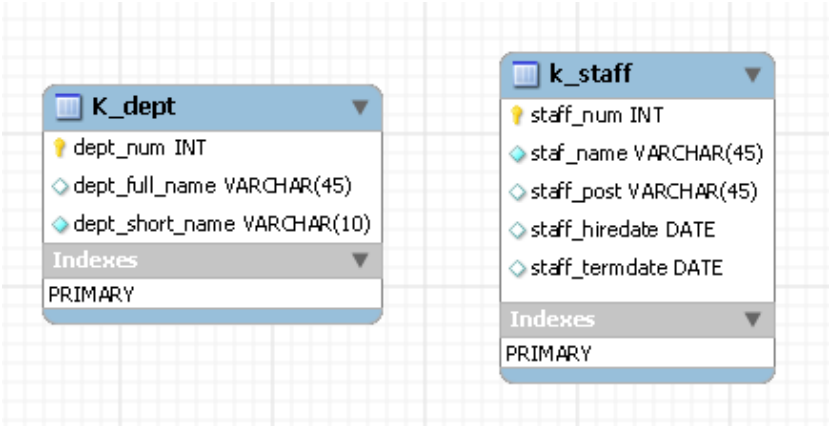
Таблиця Відділи має такий вигляд:



Потім створимо таблицю Співробітники з наступними стовпцями: номер_співробітника, ім'я_співробітника, посада, дата_початку_контракту, дата_закінчення_контракту



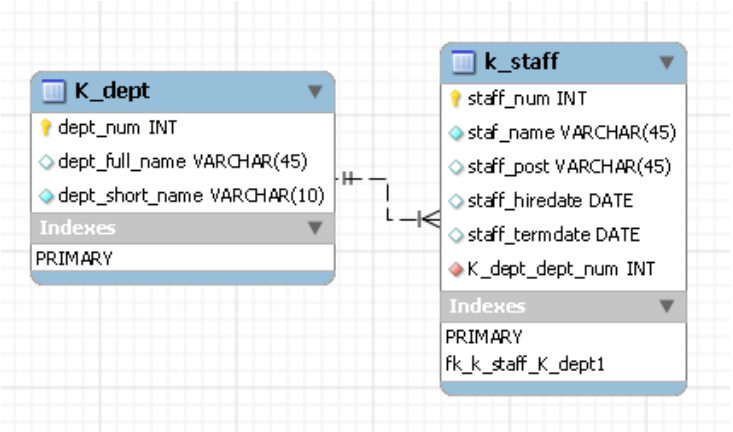
На діаграмі створені таблиці виглядають так:



Зверніть увагу, що при створенні первинного ключа автоматично створюється індекс цього первинного ключа. Індекс є допоміжною структурою, яка служить, перш за все, для прискорення пошуку та швидкого доступу до даних.

Тепер зв'яжемо ці таблиці. Спочатку створимо зв'язок «Працює» між Співробітником (дочірня таблиця) та Відділом (батьківська таблиця), ступінь зв'язку M:1. Для створення зв'язків M:1 є піктограма на панелі інструментів (з пунктирною лінією). З її допомогою створюється так званий «неідентифікуючий зв'язок», тобто. звичайний зовнішній ключ, причому первинний ключ батьківської таблиці додається до списку стовпців дочірньої таблиці.

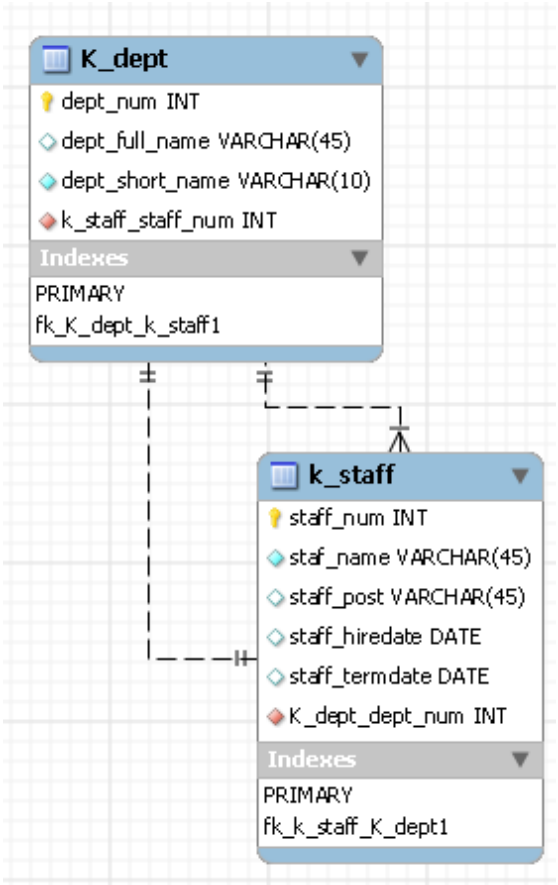
Отже, натисніть на піктограму, потім натисніть на дочірній таблиці Співробітники, потім на батьківській таблиці Відділи:



Зверніть увагу, що при цьому сталося. Між таблицями утворилася пунктирна лінія; у бік «до одного» вона відзначена двома рисками, у бік «до багатьох» - «курячою лапкою». Крім того, у таблиці Співробітники утворився додатковий стовпець, якому автоматично присвоєно ім'я k_dept_dept_num (тобто ім'я батьківської таблиці плюс ім'я первинного ключа батьківської таблиці). А в групі Індекси створено індекс із зовнішнього ключа.

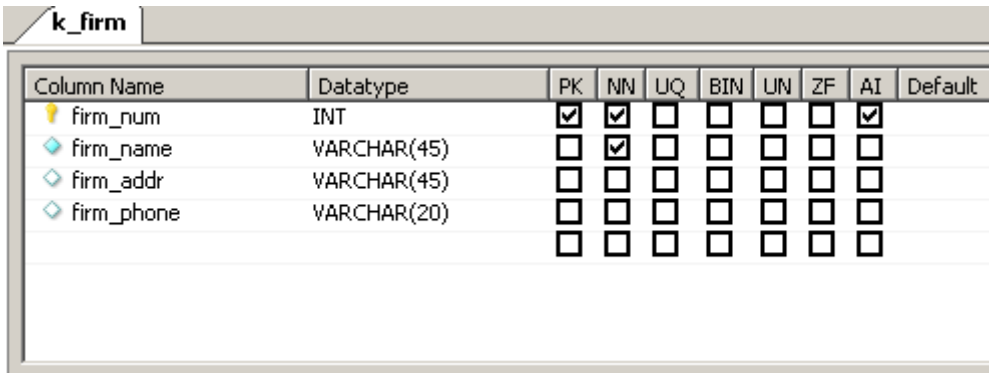
Тепер додамо зв'язок 1:1 між тими самими таблицями. Виберемо піктограму , потім натисніть Відділи, потім по Співробітникам.

Щоб два зв'язку на картинці не «зав'язувалися вузлом», ми їх розмістили один під одним.

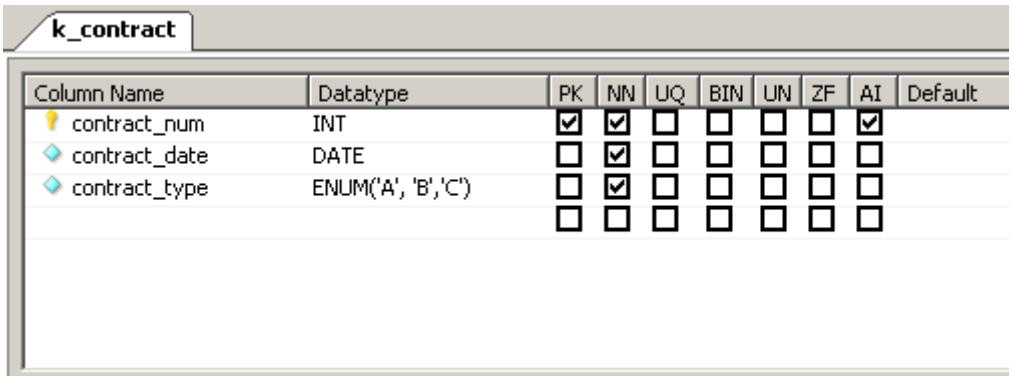


Зверніть увагу, що до таблиці Відділів було автоматично додано стовпець k_staff_staff_num, а також індекс із зовнішнього ключа.

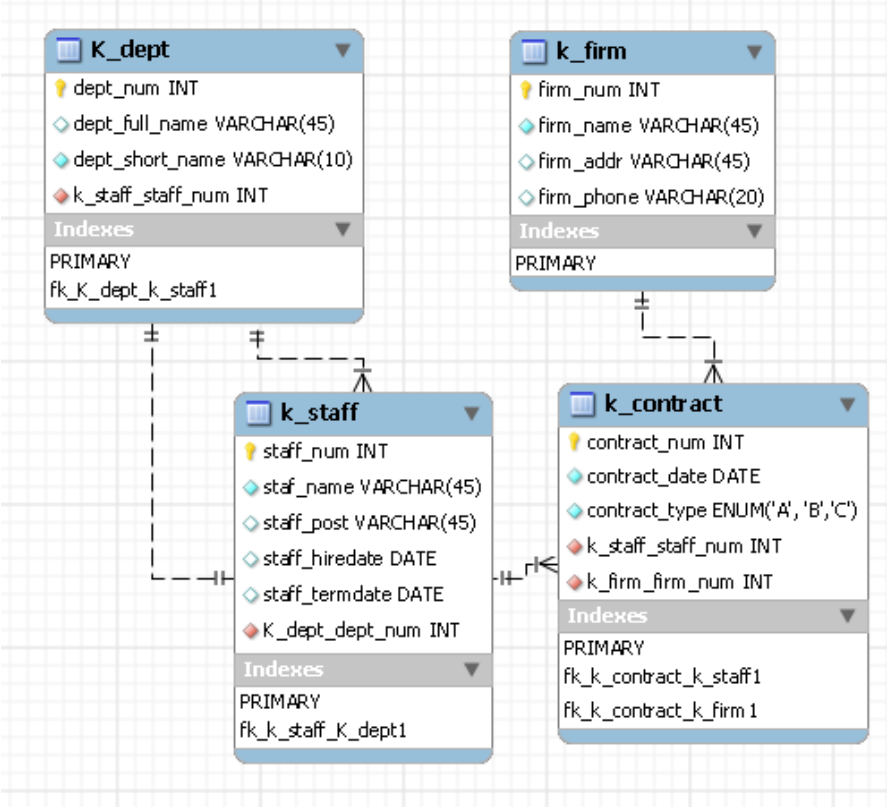
Створимо таблицю Підприємства:



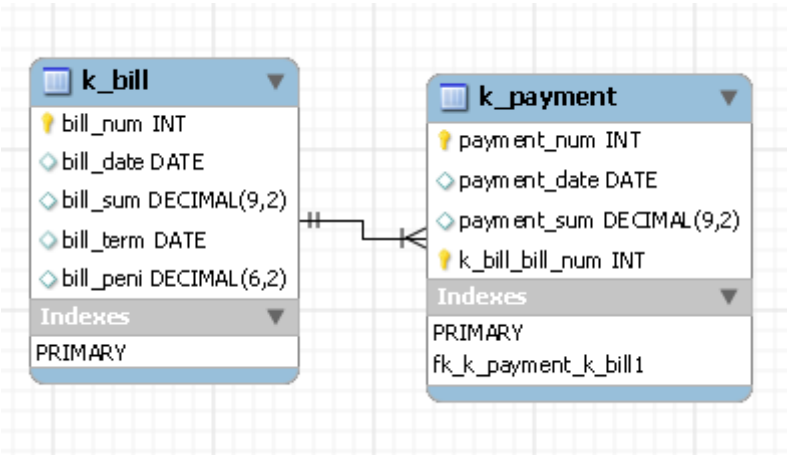
Створимо таблицю Договори. У стовпця Тип_договора задамо наступний формат: це буква зі списку 'A', 'B', 'C'.



Зв'яжемо Договори із Співробітниками та Підприємствами зв'язками M:1.



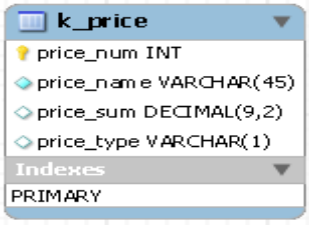
Потім створимо Рахунки та Платежі:

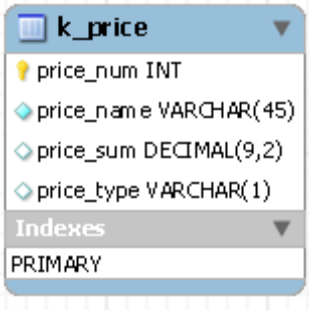


Оскільки сутність Платіж була «слабкою», у неї немає повноцінного первинного ключа, і кожен платіж однозначно ідентифікується групою атрибутів (номер_рахунку, номер платежу). Відзначимо як ключове поле payment_num, а потім створимо ідентифікуючий зв'язок між Рахунком і Платежем.

Ідентифікуючий зв'язок створюється за допомогою піктограми (суцільною лінією). При цьому новий стовпець k_bill_bill_num стає не лише зовнішнім ключем у таблиці Платіж, а й частиною первинного ключа.

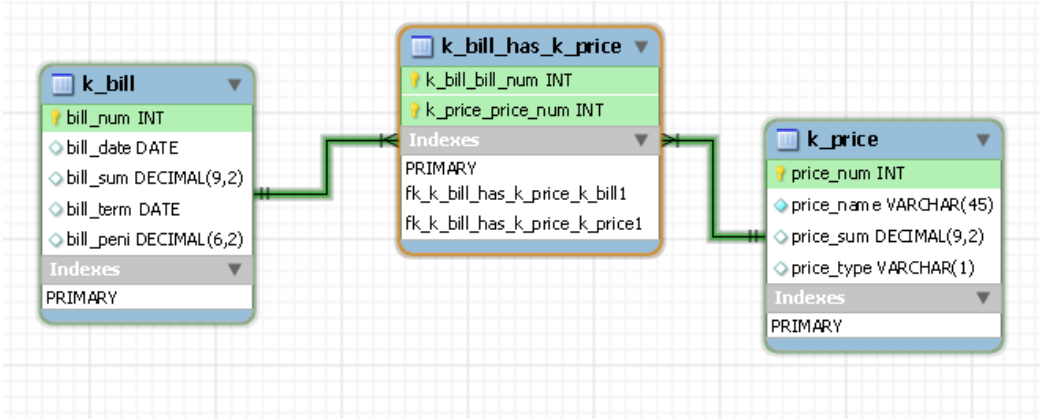
Далі створимо таблицю Прайс-лист зі стовпцями (номер_товару, назва_товару, ціна_товару та тип_товару).



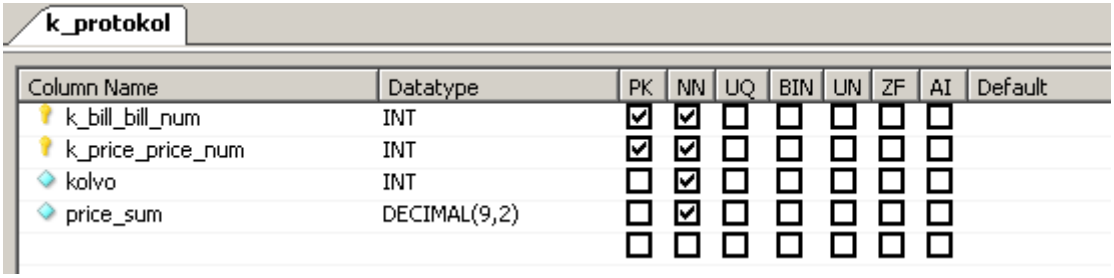


Між об'єктами Рахунок та Прайс-лист є зв'язок «багато - до багатьох». Для створення зв'язку потрібно використовувати піктограму 

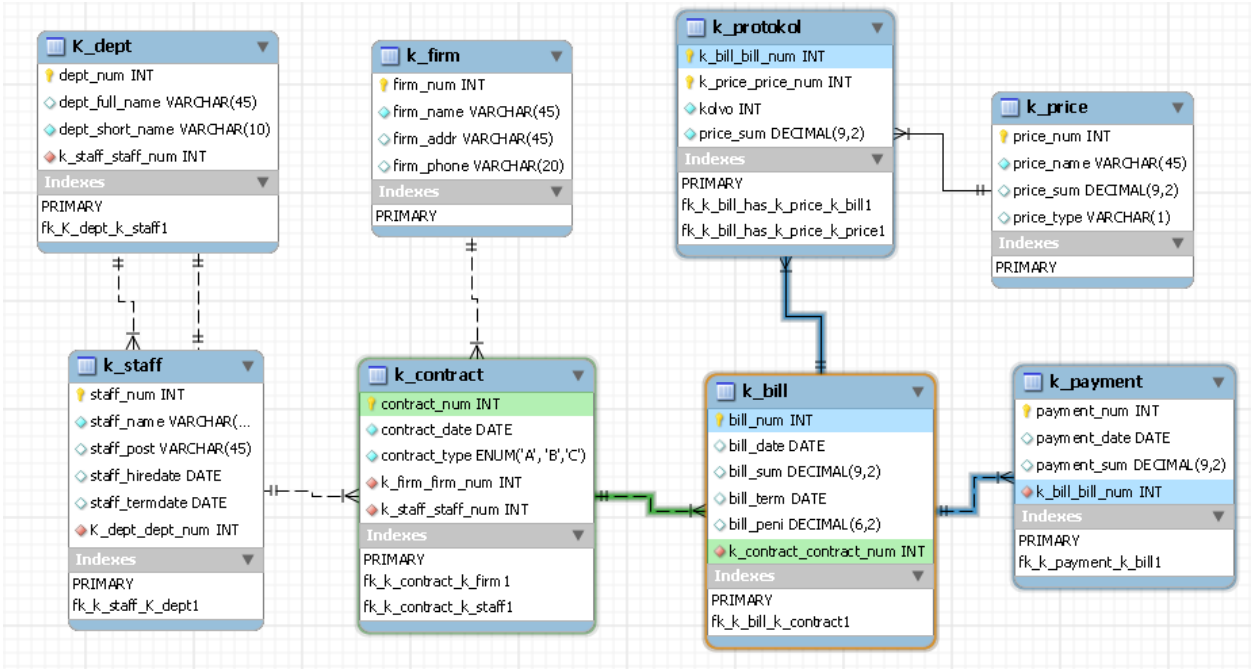
. Слід клацнути мишею по цій піктогр амі, а потім послідовно клацнути по таблицях, що зв'язуються. Між ними з'явиться нова таблиця, зверніть увагу на її стовпці, первинний ключ та зовнішні ключі:



Для зручності перейменуємо цю таблицю в k_protokol (ПротоколРахунки), додамо стовпці kolvo і price_sum.



Тепер EER-діаграма має такий вигляд:

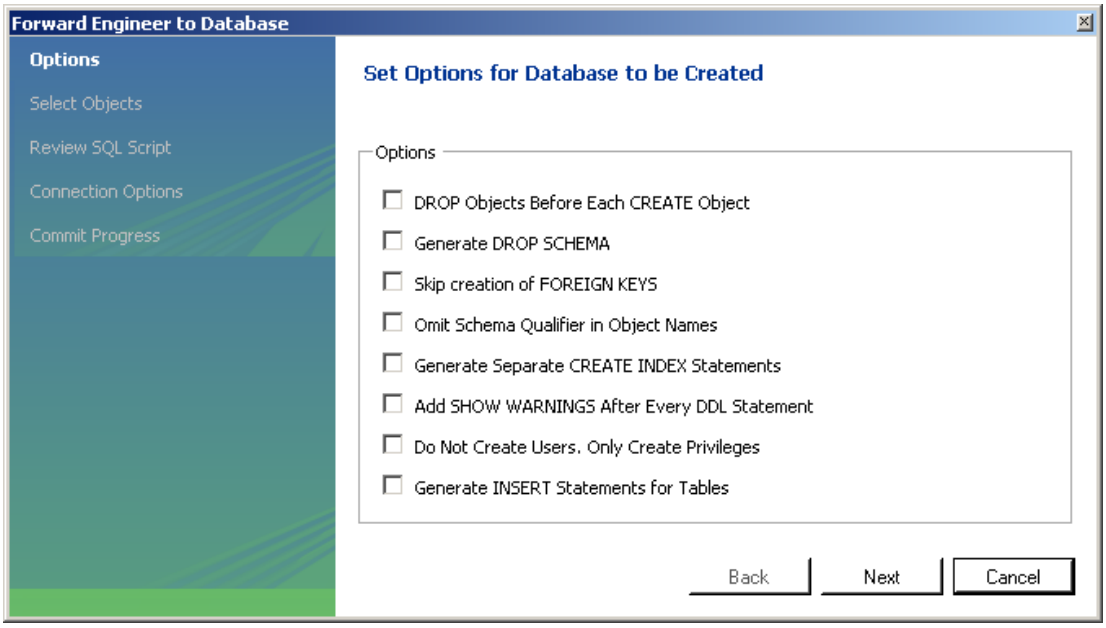


4.2. Створення бази даних з EER-діаграми

На попередньому етапі ми розробили EER-діаграму для нашої предметної області.

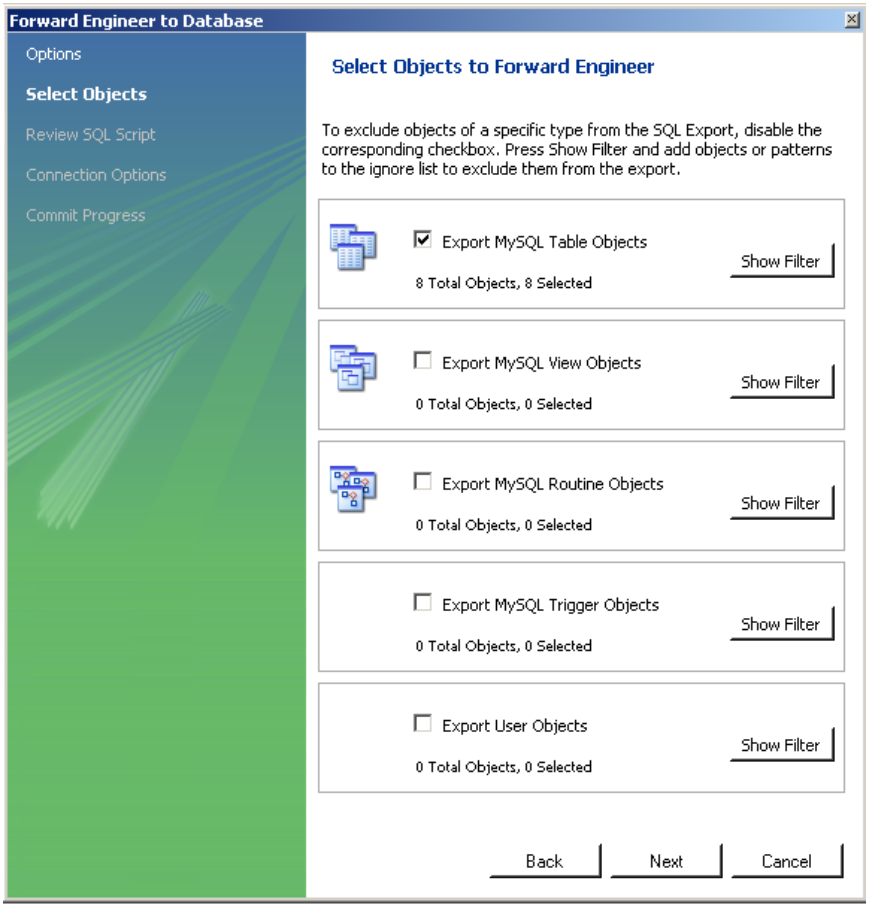
Тепер на основі цієї діаграми створимо фізичну базу даних. Виберемо пункт меню Database – Forward Engeneer. Запуститься майстер побудови бази даних.

На першому кроці можна встановити деякі додаткові дії. Спочатку нічого на цій сторінці не вибираємо, натискаємо Next.

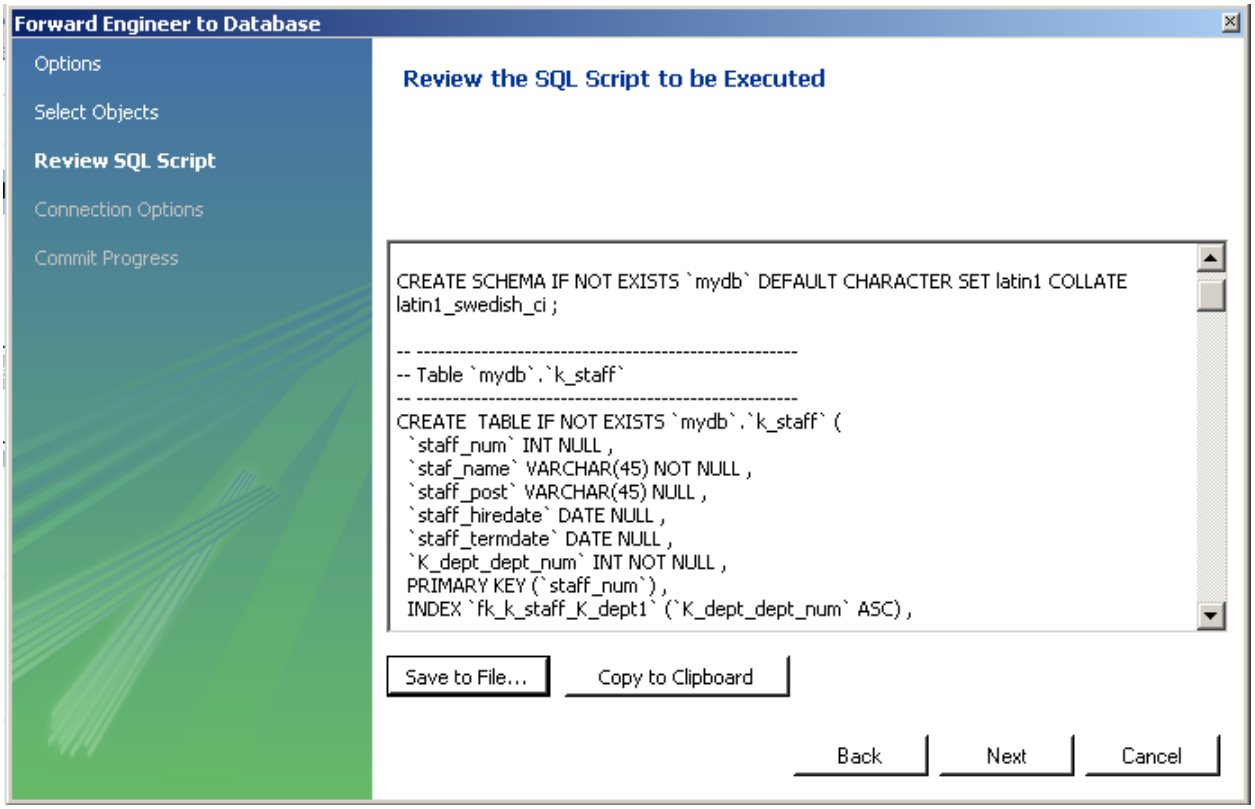


(Примітка: при повторному створенні бази даних потрібно буде увімкнути перші два прапорці для видалення старих таблиць.)

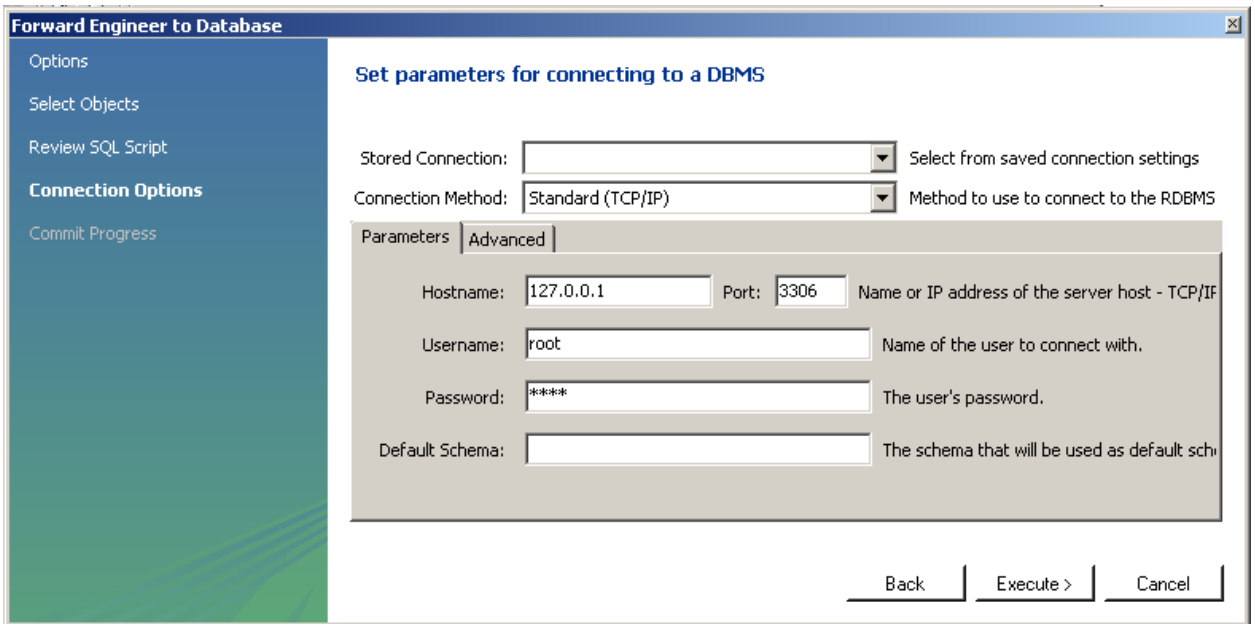
На другій сторінці включено прапорець, що вказує, що ми створюємо таблиці (загалом 8 шт.). Інших об'єктів поки що у нас немає.



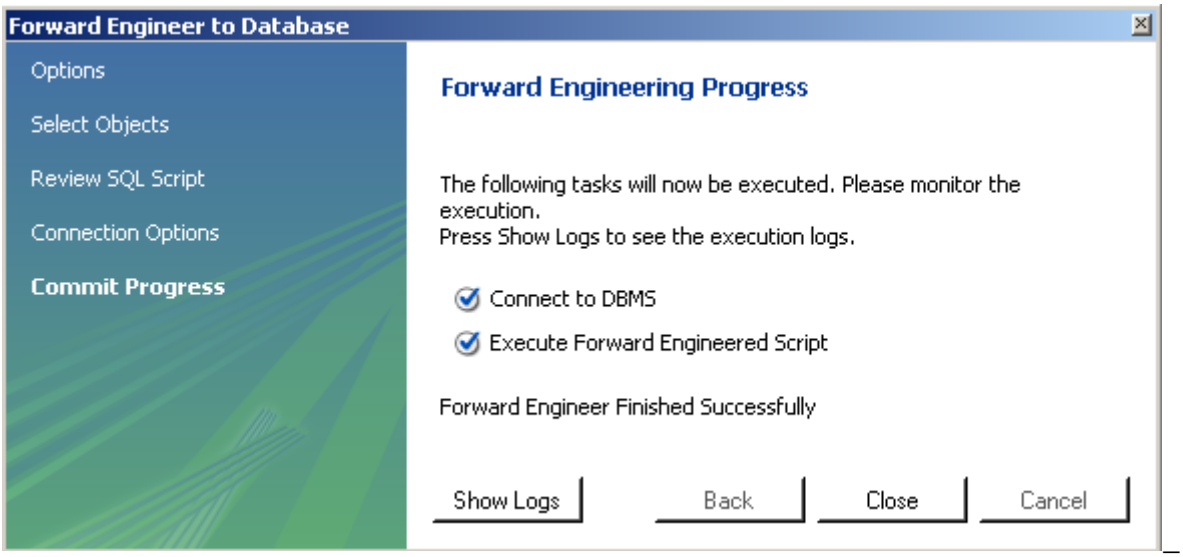
На наступній сторінці відображається текст сценарію для створення бази даних.



Далі запитується логін та пароль для підключення до сервера:

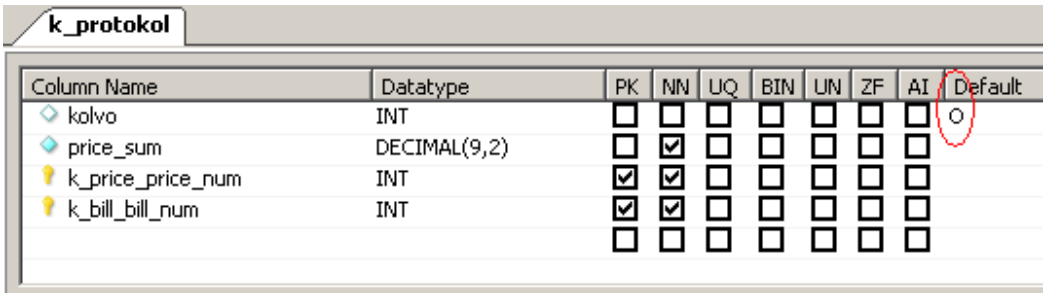


Якщо немає жодних помилок, то отримаємо вікно з повідомленням про успішний результат:

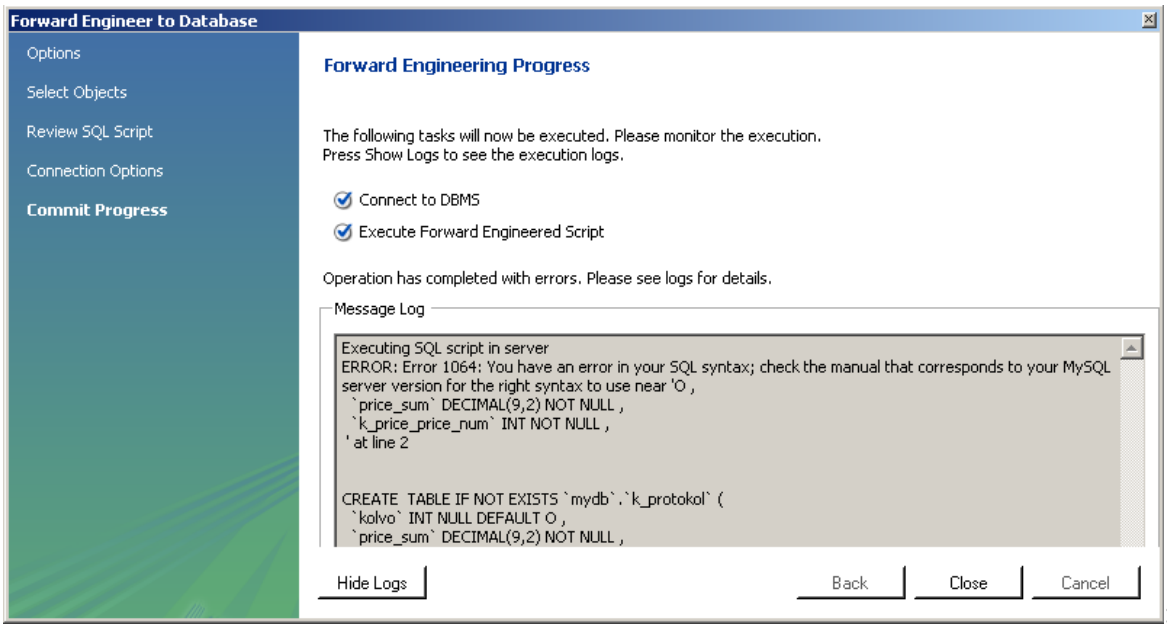


В іншому випадку можна натиснути кнопку Show logs і подивитися протокол помилок.

Які можуть бути помилки? Наприклад, у таблиці Протокол_iРахунки ми захотіли вказати для кількості значення за промовчанням «нуль», а натомість набрали букву О:



Отримаємо помилку



Завдання. Створіть у MySQL WORKBENCH базу даних із EER-діаграми.

Додаткова інформація. Під час створення таблиць використовується команда CREATE TABLE.

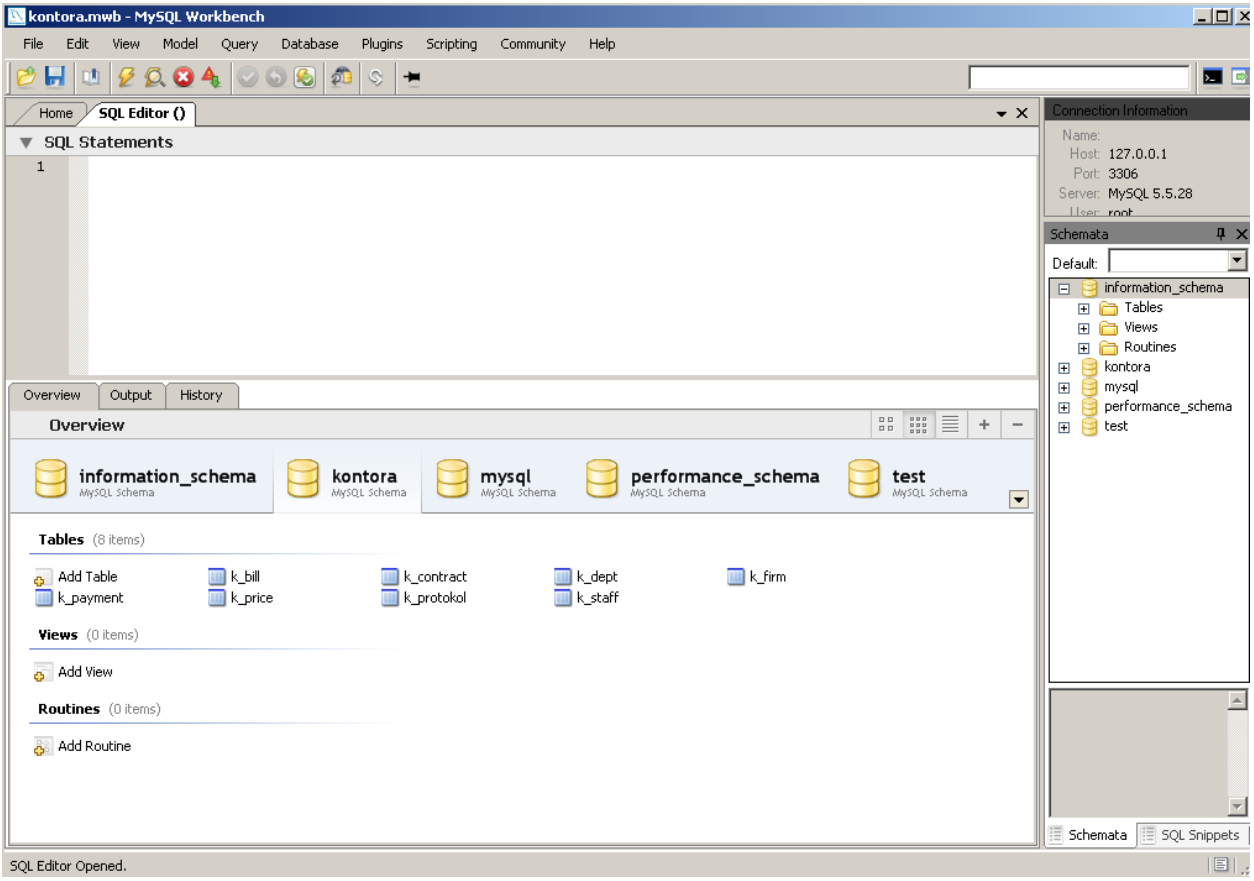
4.3. Заповнення бази даних, модифікація даних

На попередньому етапі ми створили базу даних. Тепер її заповнюватимемо даними. Підключимося до сервера, у секції

клацніть на посилання Open Connection to Start Querying. У вікні потрібно задати username і password, і натиснути на кнопку «ОК».

Ми підключилися до MySQL Server.

У цьому режимі робоча область MySQL Workbench розділена на 3 вікна:



• Верхнє вікно SQL Statements призначене для введення та виконання команд SQL. Увага! У OS Windows XP текст, набраний у цьому вікні, автоматично не зберігається. Якщо ви перемістилися з цього вікна в інший режим роботи, текст може бути втрачений.

• Нижнє вікно з кількома вкладками показує структуру наявних баз даних та дозволяє ними керувати. Наприклад, якщо "двічі натиснути" по якій-небудь таблиці, то відкриється додаткове вікно зі структурою цієї таблиці в нижній частині робочої області.

• Праве вікно містить ієрархію об'єктів сервера.

Для заповнення бази даних MySQL Workbench є кілька можливостей. Розглянемо три з них.

Перший метод заповнення бази даних - використовуємо команду INSERT

Найбільш універсальний та гнучкий спосіб створення даних полягає у використанні SQL-команди INSERT. Формат у неї такий:

INSERT INTO Ім'яТаблиці [(СписокСтовпцівТаблиці)]

VALUES (СписокЗначень);

У квадратних дужках зазначаються необов'язкові елементи команди. Якщо в цій команді пропустити Список Стовпців Таблиці, то маються на увазі ВСІ стовпці, і саме в такому порядку, як вони були визначені при створенні таблиці.

SQL-команди потрібно набирати у вікні SQL statement. Для виконання команд потрібно вибрати меню Query - Execute або кнопку на панелі інструментів або натиснути Ctrl + Enter.

Можна набрати кілька команд і виконати їх усі разом, або виділити окрему команду (як копіювання) і виконати лише її. Текст SQL команд, який також називають SQL сценарієм, можна (і потрібно!) зберігати у файл. Типовий тип файлу SQL.

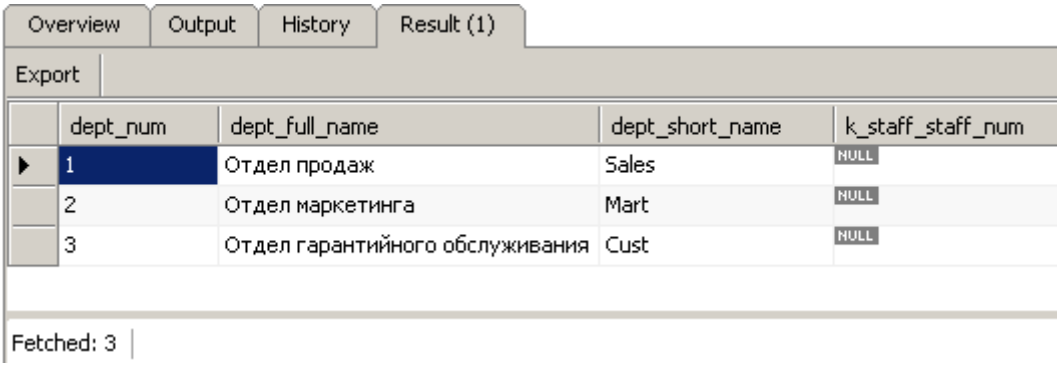
Заповнимо таблицю Підприємства:

```
# Виберемо базу даних

USE kontora;

# додамо рядки

INSERT INTO k_firm (firm_name, firm_addr)

VALUES('Альфа', 'Осло');

INSERT INTO k_firm (firm_name, firm_addr)

VALUES('Бета', 'Рим');

INSERT INTO k_firm (firm_name, firm_addr)

VALUES('Гамма', 'Париж');

INSERT INTO k_firm (firm_name, firm_addr)

VALUES('Дельта', 'Лондон');

INSERT INTO k_firm (firm_name, firm_addr)

VALUES('Омега', 'Токіо');


# подивимося результат

SELECT * FROM k_firm;
```

Зверніть увагу, що ми не задавали значення для стовпця firm_num, оскільки цей стовпець має властивість Auto increment, і сервер його заповнює сам натуральними числами.

Заповнимо Відділ

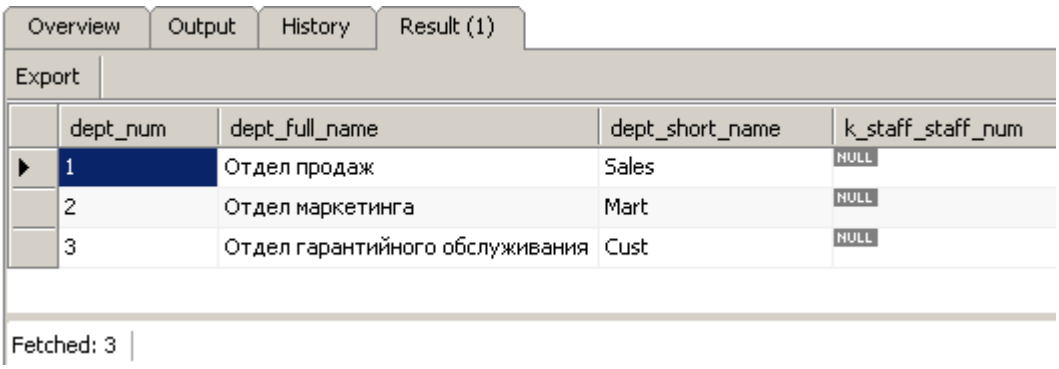
```
INSERT INTO k_dept (dept_short    _name, dept_full_name)

VALUES('Sales', 'Відділ продажів');

INSERT INTO k_dept (dept_short_name, dept_full_name)

VALUES('Mart', 'Відділ маркетингу');

INSERT INTO k_dept (dept_short_name, dept_full_name)

VALUES('Cust', 'Відділ гарантійного обслуговування');


SELECT * FROM k_dept;
```



Заповнимо таблицю Співробітник.

Зверніть увагу, що в цій таблиці можна вказувати лише такий номер відділу, який існує у таблиці Відділ! (Залишити це поле порожнім теж можна.)

```
INSERT INTO k_staff (staff_name, K_dept_dept_num, staff_hiredate, staff_post)

VALUES('Іванів', 1, '1999-01-01', 'Менеджер');

INSERT INTO k_staff (staff_name, K_dept_dept_num, staff_hiredate, staff_post)

VALUES('Петрів', 2, '2010-10-13','Менеджер');

INSERT INTO k_staff (staff_name, K_dept_dept_num, staff_hiredate, staff_post)

VALUES('Сидорів', 3, '2005-12-01','Менеджер');

INSERT INTO k_staff (staff_name, staff_hiredate, staff_post)

VALUES('Семенов', '1990-01-01','Директор');

INSERT INTO k_staff (staff_name, K_dept_dept_num, staff_hiredate, staff_post)

VALUES('Григор'єв', 3, '2008-12-19','Програміст');

SELECT * FROM k_staff;
```



Що ж буде, якщо вказати неіснуючий номер відділу?

INSERT INTO k_staff

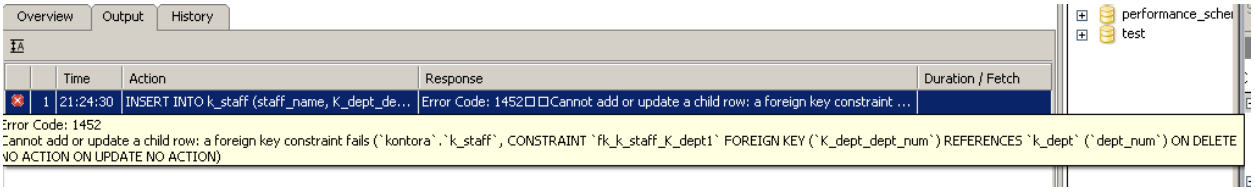
(staff_name, K_dept_dept_num, staff_hiredate, staff_post)

VALUES('Сміт', 10, '2013-01-01','Консультант');

Буде отримано таку помилку:

Error code: 1452

Cannot add або update a child row: a foreign key constraint fails (`kontora`, `k_staff`, CONSTRAINT `fk_staff_k_dept` FOREIGN KEY (`k_dept_dept_num`) REFERENCES `k_dept` (`dept



Заповнимо таблицю Договір

INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('A', 1, 1,'2011-11-01');

INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('B', 1, 2,'2011-10-01');

INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('C', 1, 1,'2011-09-01');

INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('A', 2, 2,'2011-11-15');

INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('B', 2, 2,'2011-08-01');

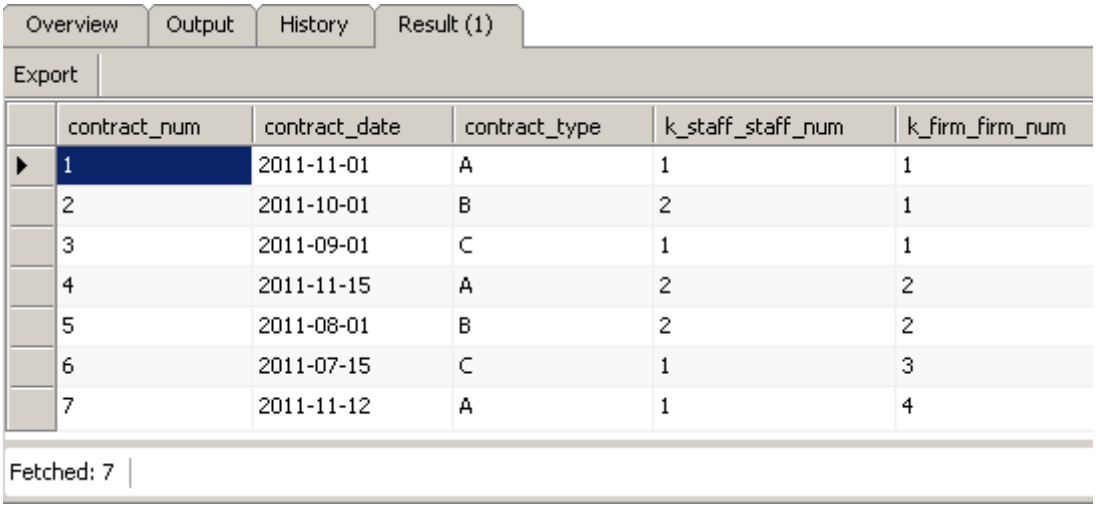
INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('C', 3, 1,'2011-07-15');

INSERT INTO k_contract

(contract_type, k_firm_firm_num, k_staff_staff_num, contract_date) VALUES('A', 4, 1,'2011-11-12');

SELECT * FROM k_contract;


Overview | Output | History | Result (1)

Export

contract_num	contract_date	contract_type	k_staff_staff_num	k_firm_firm_num
1	2011-11-01	A	1	1
2	2011-10-01	B	2	1
3	2011-09-01	C	1	1
4	2011-11-15	A	2	2
5	2011-08-01	B	2	2
6	2011-07-15	C	1	3
7	2011-11-12	A	1	4

Fetched: 7

Заповнимо таблицю Рахунок

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(1, '2011-11-12', '2011-12-12', 1000);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(1, '2011-12-12', '2012-01-12', 2000);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(1, '2012-01-12', '2012-02-12',2000);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(2, '2011-12-12', '2012-01-12', 6000);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(2, '2012-01-12', '2012-02-12', 2000);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(3, '2012-01-12', '2012-02-12', 2500);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(4, '2011-12-12', '2012-01-12', 1500);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(5, '2011-12-12', '2012-01-12', 1200);

INSERT INTO k_bill

(k_contract_contract_num, bill_date, bill_term, bill_sum)

VALUES(5, '2012-01-12', '2012-02-12', 10000);

SELECT * FROM k_bill;

Overview | Output | History | Result (1)

Export

bill_num	bill_date	bill_sum	bill_term	bill_peni	k_contract_contract_num
1	2011-11-12	1000.00	2011-12-12	NULL	1
2	2011-12-12	2000.00	2012-01-12	NULL	1
3	2012-01-12	2000.00	2012-02-12	NULL	1
4	2011-12-12	6000.00	2012-01-12	NULL	2
5	2012-01-12	2000.00	2012-02-12	NULL	2
6	2012-01-12	2500.00	2012-02-12	NULL	3
7	2011-12-12	1500.00	2012-01-12	NULL	4
8	2011-12-12	1200.00	2012-01-12	NULL	5
9	2012-01-12	10000.00	2012-02-12	NULL	5

Fetched: 9

та інші таблиці:

SELECT * FROM k_payment;

Overview | Output | History | Result (1)

Export

payment_num	payment_date	payment_sum	k_bill_bill_num
1	2011-12-15	1000.00	2
1	2012-01-13	1500.00	3
1	2012-01-12	1000.00	4
1	2012-01-05	100.00	7
1	2011-12-25	1000.00	8
2	2012-01-15	500.00	3
2	2012-01-12	900.00	7

SELECT * FROM k_price;

Overview | Output | History | Result (1)

Export

price_num	price_name	price_sum	price_type
1	Материализация духов	1000.00	У
2	Раздача слонов	100.00	У
3	Слоновий бивень	3000.00	Т
4	Моржовый клык	1500.00	Т
5	Копыто Пегаса	5000.00	Т

'У' означає послугу, 'Т' – товар.

SELECT * FROM k_protokol;

Overview | Output | History | Result (1)

Export

kolvo	price_sum	k_price_price_num	k_bill_bill_num
1	1000.00	1	1
2	1000.00	1	2
1	1000.00	1	5
2	1000.00	1	6
1	1000.00	1	8
20	100.00	2	3
10	100.00	2	5
5	100.00	2	6
2	100.00	2	8
2	3000.00	3	4
1	1500.00	4	7
2	5000.00	5	9

Крім команди додавання даних INSERT, є корисні команди зміни даних UPDATE та видалення даних DELETE.

Формат команди UPDATE:

UPDATE [INTO] Ім'яТаблиці SET Ім'яСтовпця=НовеЗначення

[WHERE Умова];

Квадратні дужки означають необов'язкову частину команди. Якщо умови немає, то змінюються ВСІ рядки заданої таблиці.

Застосуємо цю команду практично. Якщо ви звернули увагу, у таблиці Відділ залишився незаповненим стовпець k_staff_staff_num, який означає номер співробітника – керівника відділу.

UPDATE k_dept SET k_staff_staff_num=2

WHERE dept_short_name='Mart';

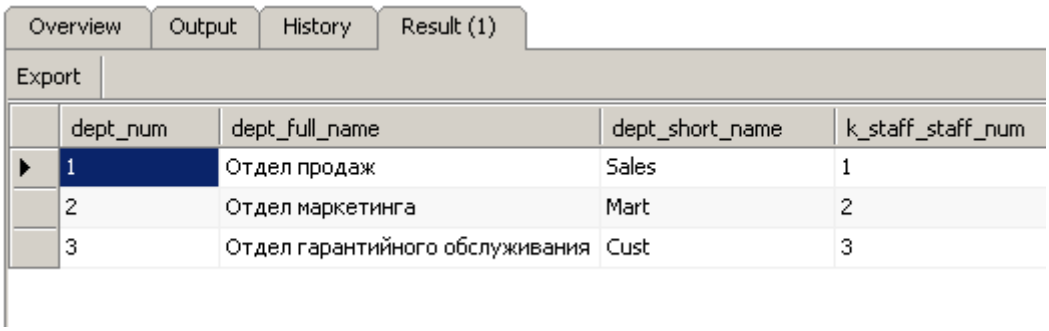
UPDATE k_dept SET k_staff_staff_num=3

WHERE dept_short_name='Cust';

UPDATE k_dept SET k_staff_staff_num=1

WHERE dept_short_name='Sales';

Результат:



dept_num	dept_full_name	dept_short_name	k_staff_staff_num
1	Отдел продаж	Sales	1
2	Отдел маркетинга	Mart	2
3	Отдел гарантийного обслуживания	Cust	3

Формат команди DELETE:

DELETE [FROM] имя_таблицы [WHERE умова];

Квадратні дужки означають необов'язкову частину команди. Якщо умови немає, то видаляються ВСІ рядки заданої таблиці.

Приклад: видаляємо фірму з номером 5:

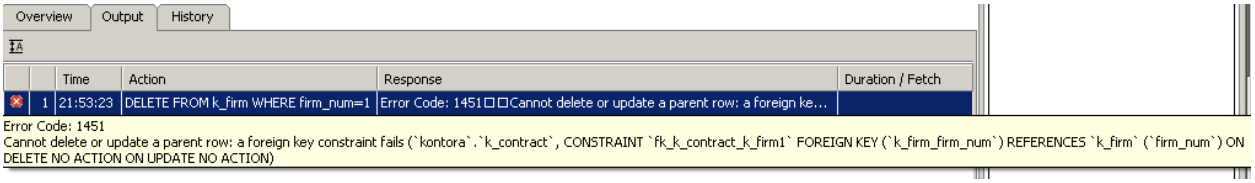
DELETE FROM k_firm WHERE firm_num=5;

Результат успішний. А що буде, якщо спробувати вилучити фірму з номером 1? Ця фірма має підпорядковані рядки в таблиці Договір.

Помилка:

Error Code 1451

Усунути або update parent row: a foreign key constraint fails (`kontora`, `k_contract`, CONSTRAINT `fk_contract_k_firm` FOREIGN KEY (`k_firm_firm_num`) REFERENCES `k_firm` (`firm_num`) …



Другий метод заповнення бази даних - використовуємо візуальні засоби

Щоб заповнювати базу даних за допомогою візуальних засобів, у вікні сервера потрібно “двічі натиснути” на потрібну таблицю (або виконати команду EDIT Ім'я Таблиці). Відкриється вікно редагування, в якому можна змінювати та додавати дані. Не забувайте зберігати зміни натисканням кнопки «галочка»!

Цей спосіб додавання даних дуже легкий – проблема виникає лише за необхідності перенесення даних на інший комп'ютер. Простих шляхів копіювання даних немає. Ви можете використовувати вивантаження у текстові файли.

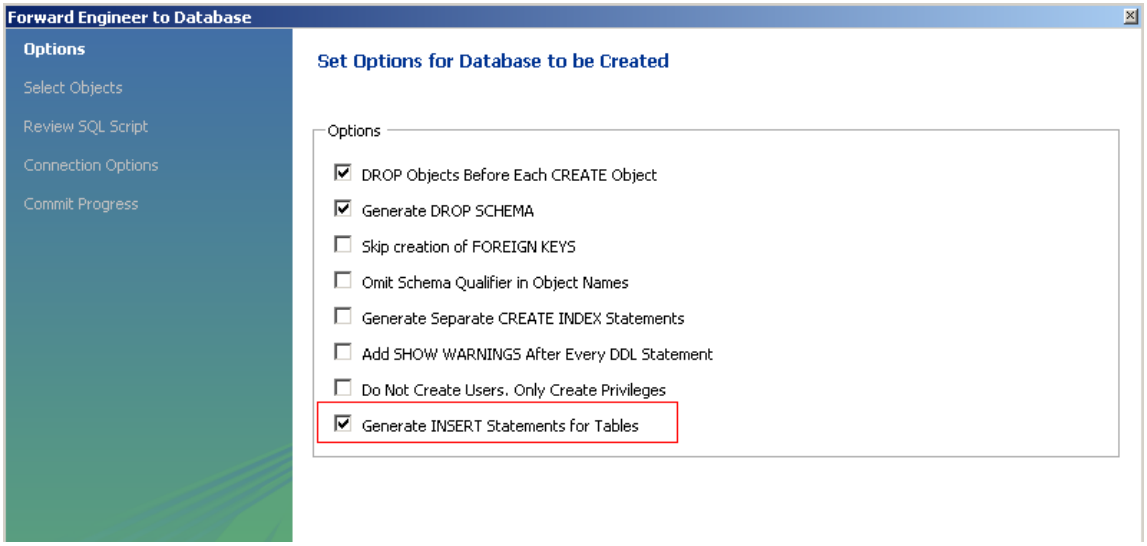
Третій метод заповнення бази даних – дані зберігаються в EER-моделі

Без цього способу заповнення можна обійтися, але для повноти картини розповімо про нього теж.

Для застосування цього способу доведеться повернутися на крок назад та відкрити EER-діаграму. Як ви пам'ятаєте, якщо в діаграмі двічі натиснути на ім'я таблиці, то відкривається вікно її властивостей. Під час створення структури таблиці ми використовували вкладку Columns. Тепер перейдемо на вкладку Inserts і заповнимо дані в таблиці.

Номер Фірми можемо не заповнювати, він має властивість Auto Increment. Телефон ми також не заповнили, він необов'язковий. Після заповнення даних не забудьте натиснути кнопку «галочка», щоб збереглися зміни в моделі. При використанні цього є дуже істотна проблема. При заповненні таблиць не перевіряються жодні обмеження, ні типи полів, ні зовнішні ключі. Тому дуже легко зробити помилку.

Цей режим роботи схожий на попередній спосіб заповнення бази даних, але має дуже важливу відмінність! Заповнені таким чином дані зберігаються лише в EER-моделі, на стороні сервера їх немає. Для того, щоб дані з'явилися на стороні сервера, потрібно заново виконати генерацію бази даних з EER-моделі, як у другому завданні. При цьому обов'язково слід зазначити прапорець Generate INSERT Statements for Tables:



У цей момент будуть виявлені всі раніше зроблені помилки під час введення даних.

Зрозуміло, при цьому стара база даних видаляється разом з усіма раніше введеними даними.

Завдання. Заповніть базу даних. У кожній таблиці створіть кілька рядків.

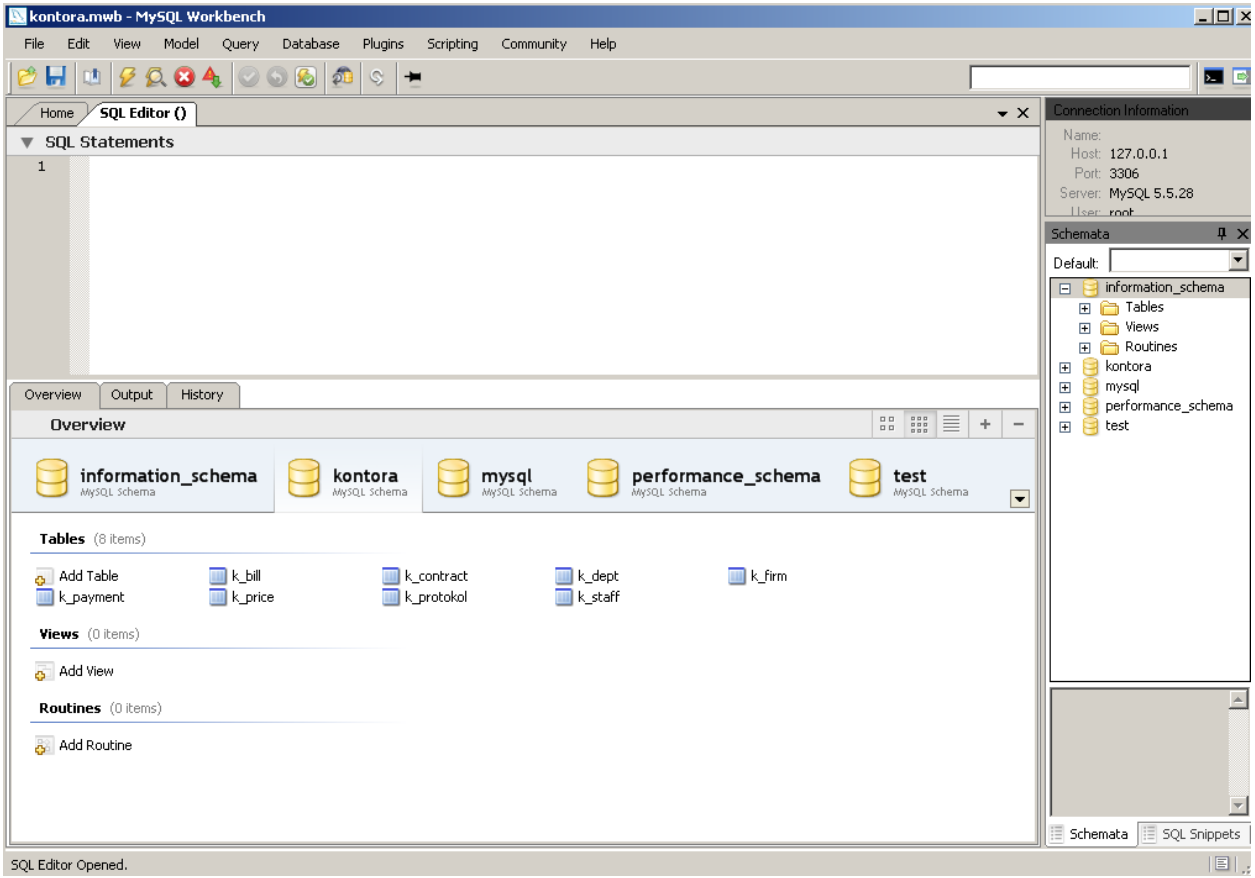
4.4. Запити до бази даних

На попередньому етапі ми заповнили таблицю бази даних. Розглянемо команду SELECT, що дозволяє виконувати запити до бази даних. Щодо цієї команди написані цілі книги, тут ми коротко на прикладах розглянемо її основні можливості.

Як і в попередньому завданні, підключимося до сервера. Для цього у секції

клацніть на посилання Open Connection to Start Querying. У вікні потрібно задати username і password, і натиснути на кнопку «OK».

Команди SELECT потрібно набирати у вікні SQL Statements:



Найпершою командою в сеансі даних має бути команда вибору поточної бази даних:

USE kontora;

Вибірка з однієї таблиці

Обов'язкові ключові слова команди – SELECT та FROM.

Тривіальна вибірка всіх полів та всіх рядків однієї таблиці.

Отримати повну інформацію про всі підприємства:

SELECT * FROM k_firm

Вибір окремих полів таблиці.

Отримати назви та адреси всіх підприємств:

SELECT firm_name, firm_addr FROM k_firm

Поля вибірки можна перейменовувати. Якщо нова назва складається з кількох слів, поміщайте її в лапки.

SELECT firm_name AS "Назва підприємства", firm_addr AS "Адреса підприємства" FROM k_firm

У списку полів вибірки можна використовувати вирази. У цьому випадку часто доводиться перетворювати дані з одного типу на інший за допомогою функції CONVERT. Рядкові константи слід поміщати в одинарні лапки. Функція CONCAT служить для зчеплення рядків.

Роздрукувати інформацію про рахунки:

```
SELECT CONCAT("Рахунок №", CONVERT(bill_num, CHAR),
" від ", CONVERT(bill_date,  CHAR),
"на суму", CONVERT(bill_sum, CHAR)) AS Рахунки
FROM k_bill
```

Щоб виключити дублікати рядків, необхідно використовувати ключове слово DISTINCT.

Надрукувати список міст, в яких знаходяться підприємства-клієнти:

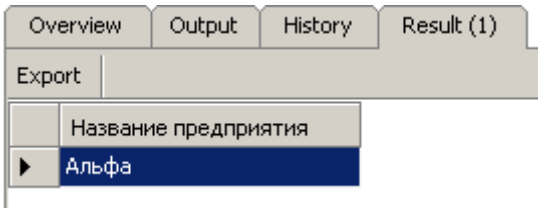
```
SELECT DISTINCT firm_addr FROM k_firm
```

Використання умов відбору

Для вибору окремих рядків за деяким критерієм використовується ключове слово WHERE

Отримати список підприємств, розташованих у Осло:

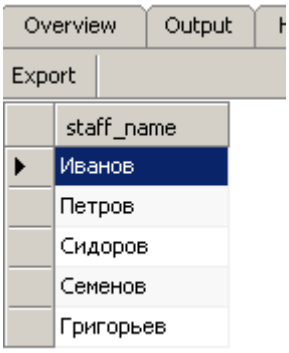
```
SELECT firm_name as "Назва підприємства"
FROM k_firm
WHERE firm_addr='Осло'
```



Для порівняння поля зі значенням NULL не можна використовувати операції = та !=, замість них потрібно використовувати вирази IS NULL та IS NOT NULL.

Отримати список співробітників, що постійно працюють, тобто таких, у яких staff_termdate дорівнює NULL:

```
SELECT staff_name FROM k_staff
WHERE staff_termdate IS NULL
```



Умови можуть бути складні, що є комбінацією декількох операцій порівняння. Вони можна використовувати логічні зв'язки AND і OR, і навіть заперечення NOT.

Отримати список підприємств, розташованих у Осло або Римі:

SELECT firm_name as "Назва підприємства"

FROM k_firm

WHERE firm_addr='Осло' OR firm_addr='Рим'

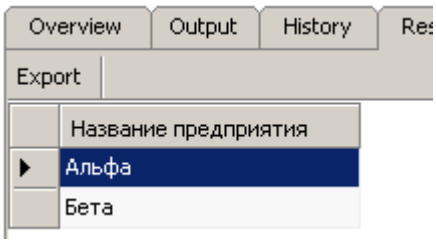
Якщо умовою є порівняння поля зі списком значень, зручно використовувати ключове слово IN.

Отримати список підприємств, розташованих у Осло або Римі:

SELECT firm_name as "Назва підприємства"

FROM k_firm

WHERE firm_addr IN ('Осло','Рим')



Якщо умова полягає у порівнянні з діапазоном значень, зручно використовувати ключове слово BETWEEN.

Отримати список договорів, укладених у листопаді 2011 р.:

SELECT * FROM k_contract

WHERE contract_date BETWEEN '2011-11-01' AND '2011-11-30'

Overview | Output | History | Result (1)

Export

	contract_num	contract_date	contract_type	k_staff_staff_num	k_firm_firm_num
▶	1	2011-11-01	A	1	1
	4	2011-11-15	A	2	2
	7	2011-11-12	A	1	4

Для полів рядкового типу можна застосовувати порівняння з підрядком.

Отримати список співробітників, прізвище яких починається на I:

```
SELECT staff_name FROM k_staff
WHERE staff_name LIKE 'I%'
```

Overview | Output | Histor

Export

	staff_name
▶	Иванов

Використання агрегатних функцій

Для підрахунку підсумкових значень служать функції SUM, COUNT, MAX, MIN, AVG. Якщо не використовується групування рядків, запит із застосуванням підсумкової функції поверне рівно один рядок.

Підрахувати, на яку суму виставлено рахунки у грудні 2011 року.

```
SELECT SUM(bill_sum) FROM k_bill
WHERE bill_date
BETWEEN '2011-12-01' AND '2011-12-31'
```

Overview | Output | History | Result

Export

	SUM(bill_sum)
▶	10700.00

Функція COUNT дозволяє підрахувати скільки рядків у таблиці є взагалі.

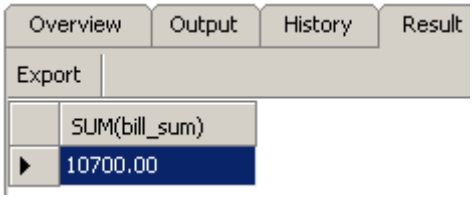
Підрахувати кількість працівників.

```
SELECT COUNT(*) FROM k_staff
```

А також ця функція дозволяє підрахувати скільки рядків з не-NULL-значеннями в певному полі.

Підрахувати кількість працівників, що тимчасово працюють (у них заповнений термін закінчення трудового договору – поле staff_termdate). Передбачається, що всі різні дати (точніше кажучи, тут підраховується кількість різних не-null значень).

SELECT COUNT(staff_termdate) FROM k_staff

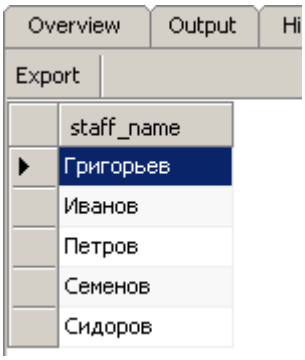


Сортування

Для сортування використовується ключове слово ORDER BY та ім'я поля або номер у списку полів вибірки.

Надрукувати список співробітників, відсортований за алфавітом:

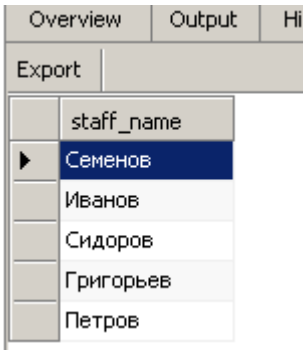
SELECT staff_name FROM k_staff ORDER BY 1



Можна сортувати рядки навіть таким полем, яке не входить до списку полів вибірки.

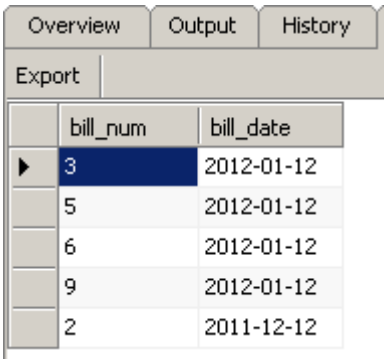
Надрукувати список працівників, відсортований за датою надходження на роботу:

SELECT staff_name FROM k_staff ORDER BY staff_hiredate



Сортувати дані можна і за спаданням. Крім того, можна обмежити кількість рядків у результаті. Надрукувати інформацію про 5 останніх виписаних рахунків у порядку зменшення дати рахунку:

```
SELECT bill_num, bill_date
FROM k_bill ORDER BY bill_date DESC LIMIT 5
```



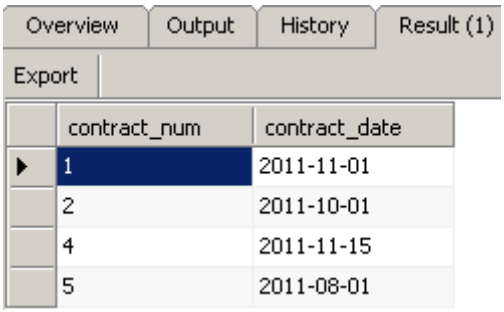
bill_num	bill_date
3	2012-01-12
5	2012-01-12
6	2012-01-12
9	2012-01-12
2	2011-12-12

Підзапити

Для складніших формулювань іноді зручно використовувати підзапити. Підзапит завжди вказується в дужках. Підзапит може бути непов'язаним, тобто. у формулюванні підзапиту немає посилання на головний запит. У цьому випадку підзапит виконується один раз під час виконання головного запиту. У цьому прикладі використовується ключове слово IN, оскільки підзапит може повертати кілька значень.

Отримати список договорів, за якими у грудні 2011 року виписано рахунки:

```
SELECT contract_num, contract_date FROM k_contract
WHERE contract_num IN
(SELECT k_contract_contract_num FROM k_bill
WHERE bill_date
BETWEEN '2011-12-01' AND '2011-12-31')
```



contract_num	contract_date
1	2011-11-01
2	2011-10-01
4	2011-11-15
5	2011-08-01

Той самий запит з використанням ключового слова ANY:

```
SELECT contract_num, contract_date FROM k_contract
WHERE contract_num = ANY
```

(SELECT k_contract_contract_num FROM k_bill

WHERE bill_date

BETWEEN '2011-12-01' AND '2011-12-31')

Той самий запит можна виконати і за допомогою пов'язаного підзапиту, тобто підзапиту, в якому є посилання на головний запит. Для посилання таблицю головного запиту необхідно вказати псевдонім. Таке запитування буде виконуватися заново для кожного рядка головного запиту.

Крім того, у цьому прикладі ілюструється використання ключового слова EXISTS:

SELECT contract_num, contract_date FROM k_contract c

WHERE EXISTS

(SELECT * FROM k_bill b

WHERE bill_date

BETWEEN '2011-12-01' AND '2011-12-31'

AND c.contract_num=b.k_contract_contract_num)

Приклад використання ключового слова ALL.

Надрукувати інформацію про товар (товари) з найменшою ціною.

SELECT price_name, price_sum FROM k_price

WHERE price_sum <= ALL

(SELECT price_sum FROM k_price)

Цей запит можна сформулювати по-іншому. У цьому вся прикладі ми можемо використовувати операцію порівняння =, т.к. підзапит повертає рівно один рядок і один стовпець.

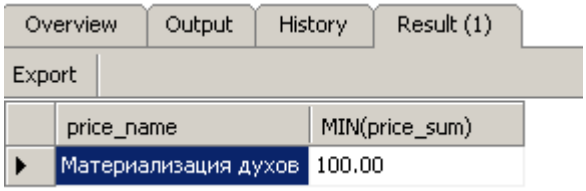
SELECT price_name, price_sum FROM k_price

WHERE price_sum =

(SELECT MIN(price_sum) FROM k_price)

А так, як у прикладі, запит формулювати не можна. При запуску помилок не буде, просто вийде неправильний результат:

SELECT price_name, MIN(price_sum) FROM k_price



Як бачите, значення стовпця price_name було просто взято з першого рядка таблиці.

Групування

Для підбиття підсумку по групі даних використовується комбінація ключового слова GROUP BY та функцій, що агрегують. Причому у списку полів для вибірки зазвичай присутні лише поля угруповання та функції, що агрегують. У разі потреби можна додати додаткові поля, які функціонально залежать від «ключа угруповання».

Отримати список договорів та загальну суму рахунків за кожним договором:

```
SELECT contract_num, SUM(bill_sum) AS contract_sum
FROM k_bill
GROUP BY contract_num
```

Overview | Output | History | Result (1)

Export

price_name	MIN(price_sum)
Материализация духов	100.00

У тому випадку, коли потрібно вибрати не всі групи, а лише деякі з них, використовується ключове слово HAVING:

Отримати список договорів, що мають 2 або більше рахунків, та загальну суму рахунків за кожним договором:

```
SELECT k_contract_contract_num, SUM(bill_sum) AS contract_sum
FROM k_bill
GROUP BY k_contract_contract_num
HAVING COUNT(bill_num)>=2;
```

Overview | Output | History | Result (1)

Export

k_contract_contract_num	contract_sum
1	5000.00
2	8000.00
5	11200.00

Вибірка з кількох таблиць

Для зв'язку таблиць можна використовувати те саме ключове слово WHERE, як і умов відбору. Під час вибірки з кількох таблиць рекомендується завжди використовувати псевдоніми таблиць. Справа в тому, що якщо в різних таблицях є однакові поля, то завжди потрібно уточнювати, до якої таблиці вони належать, тобто використовувати синтаксис

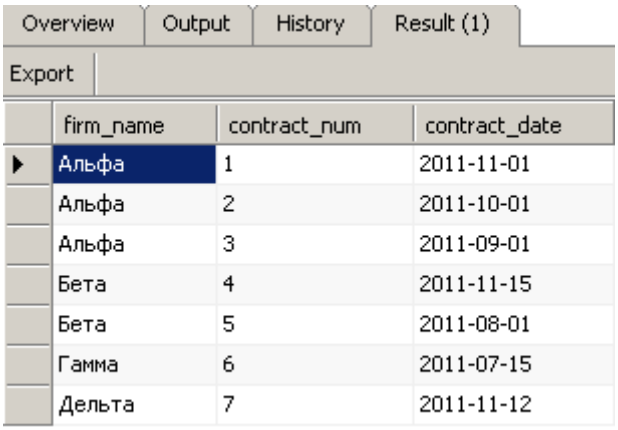
имя_таблицы.имя_поля. Оскільки імена таблиць зазвичай довгі, зручно замінювати їх псевдонімами.

Надрукувати список договорів із зазначенням назви підприємства.

SELECT firm_name, contract_num, contract_date

FROM k_firm f, k_contract c

WHERE f.firm_num=c.k_firm_firm_num


Overview | Output | History | Result (1)

Export

	firm_name	contract_num	contract_date
▶	Альфа	1	2011-11-01
	Альфа	2	2011-10-01
	Альфа	3	2011-09-01
	Бета	4	2011-11-15
	Бета	5	2011-08-01
	Гамма	6	2011-07-15
	Дельта	7	2011-11-12

Те саме можна отримати, якщо використовувати синтаксис JOIN...ON. Це так зване внутрішнє (INNER) з'єднання. Рядки з'єднуються, якщо збігаються значення полів за умови ON.

SELECT firm_name, contract_num, contract_date

FROM k_firm f JOIN k_contract c ON f.firm_num=c.k_firm_firm_num

Для з'єднання трьох та більше таблиць синтаксис у цьому форматі наступний:

Надрукувати список співробітників, номери та дати договорів, які вони уклали, із зазначенням назви підприємства.

SELECT staff_name, contract_num, contract_date, firm_name

FROM k_firm f JOIN k_contract c ON f.firm_num=c.k_firm_firm_num

JOIN k_staff s ON s.staff_num=c.k_staff_staff_num

Overview | Output | History | Result (1)

Export

staff_name	contract_num	contract_date	firm_name
Иванов	1	2011-11-01	Альфа
Иванов	3	2011-09-01	Альфа
Иванов	6	2011-07-15	Гамма
Иванов	7	2011-11-12	Дельта
Петров	2	2011-10-01	Альфа
Петров	4	2011-11-15	Бета
Петров	5	2011-08-01	Бета

Крім внутрішнього, бувають ще ліве (LEFT), праве (RIGHT) та повне (FULL) з'єднання.

Розглянемо, наприклад, ліве з'єднання. У результат потраплять рядки, у яких збігаються значення полів за умови ON, і рядки з лівої таблиці, котрим не знайшлося відповідних рядків у правій таблиці. Поля правої таблиці будуть заповнені значеннями NULL.

Надрукувати список договорів із зазначенням назви підприємства плюс список підприємств, які не мають договорів:

```
SELECT firm_name, contract_num, contract_date

FROM k_firm f LEFT JOIN k_contract c ON

f.firm_num=c.k_firm_firm_num
```

Overview | Output | History | Result (1)

Export

firm_name	contract_num	contract_date
Альфа	1	2011-11-01
Альфа	2	2011-10-01
Альфа	3	2011-09-01
Бета	4	2011-11-15
Бета	5	2011-08-01
Гамма	6	2011-07-15
Дельта	7	2011-11-12
Эпсилон	NULL	NULL

А що буде у тому випадку, якщо умову зв'язку взагалі не вказувати? Вийде так званий **декартовий добуток** таблиць, у якому кожен рядок першої таблиці буде зчеплена з кожним рядком другої таблиці. Результат виходить зазвичай дуже великим і таким, що не має сенсу.

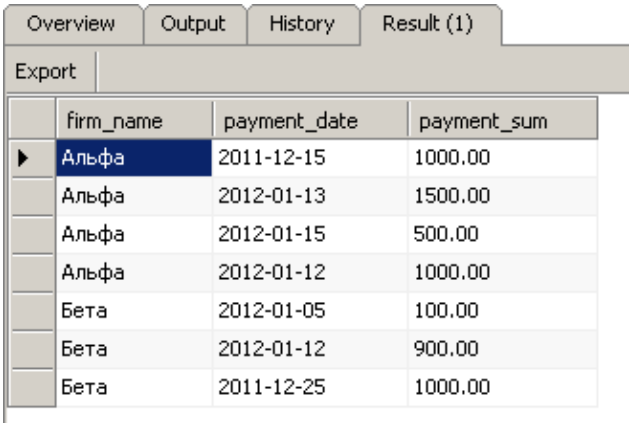
```
SELECT firm_name, contract_num, contract_date

FROM k_firm f, k_contract c

# Попередній запит повернув 35 рядків,
```

тобто. 5 підприємств помножити на 7 договорів.

Зрозуміло, в тому самому запиті можна пов'язувати не тільки дві, а три і більше таблиці, використовувати в цих запитах підзапити, угруповання і т.п. Наприклад, запит до 4 таблиць:

Надрукувати інформацію про платежі із зазначенням назви підприємства:

```
SELECT firm_name, payment_date, payment_sum
FROM k_firm f, k_contract c, k_bill b, k_payment p
WHERE f.firm_num=c.k_firm_firm_num AND
c.contract_num=b.k_contract_contract_num AND
b.bill_num=p.k_bill_bill_num
```



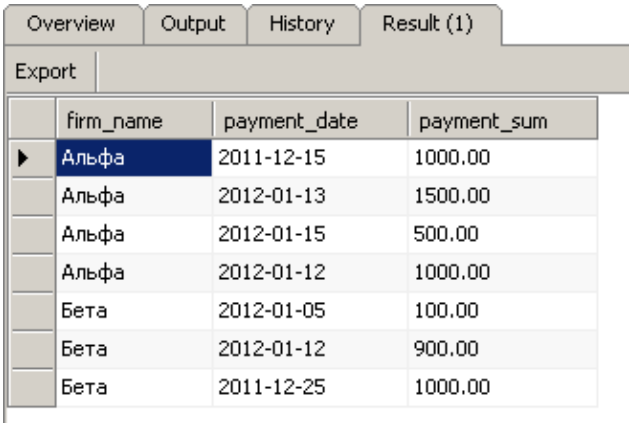
firm_name	payment_date	payment_sum
Альфа	2011-12-15	1000.00
Альфа	2012-01-13	1500.00
Альфа	2012-01-15	500.00
Альфа	2012-01-12	1000.00
Бета	2012-01-05	100.00
Бета	2012-01-12	900.00
Бета	2011-12-25	1000.00

Об'єднання запитів

Для об'єднання двох або більше запитів потрібно використовувати ключове слово UNION. Об'єднані запити повинні мати однакову кількість та тип полів. Параметр ORDER BY , якщо він потрібний, слід вказувати лише в останньому запиті.

Отримати список договорів та загальну суму рахунків за кожним договором, а також рядок із підсумковою сумою:

```
SELECT CONCAT('Договір №',
CONVERT(k_contract_contract_num, CHAR),
' на суму ') AS "Номер",
SUM(bill_sum) AS "Сума" FROM k_bill
GROUP BY k_contract_contract_num
UNION
SELECT 'РАЗОМ: ', SUM(bill_sum) FROM k_bill ORDER BY 1
```



Overview | Output | History | Result (1)

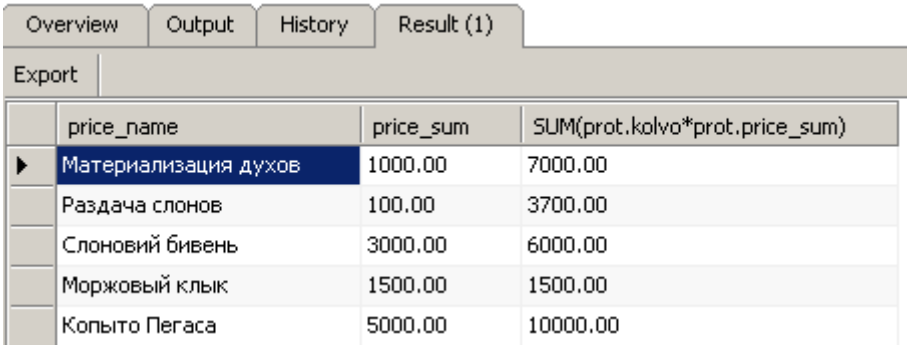
Export

	firm_name	payment_date	payment_sum
▶	Альфа	2011-12-15	1000.00
	Альфа	2012-01-13	1500.00
	Альфа	2012-01-15	500.00
	Альфа	2012-01-12	1000.00
	Бета	2012-01-05	100.00
	Бета	2012-01-12	900.00
	Бета	2011-12-25	1000.00

І ще кілька прикладів

Отримати прайс-лист із сумою замовлень щодо кожного товару. Зверніть увагу, що назва та ціна товару можуть використовуватись у списку полів для вибору, оскільки вони функціонально (однозначно) залежать від номера товару, за яким проводиться угруповання.

```
SELECT pr.price_name, pr.price_sum,
SUM(prot.kolvo*prot.price_sum)
FROM k_price pr, k_protokol prot
WHERE pr.price_num=prot.k_price_price_num
GROUP BY pr.price_num ORDER BY 1
```



Overview | Output | History | Result (1)

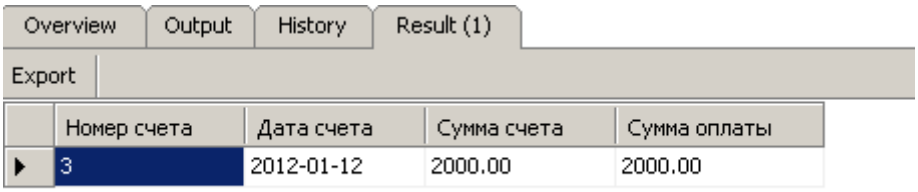
Export

	price_name	price_sum	SUM(prot.kolvo*prot.price_sum)
▶	Материализация духов	1000.00	7000.00
	Раздача слонов	100.00	3700.00
	Слоновий бивень	3000.00	6000.00
	Моржовый клык	1500.00	1500.00
	Копыто Пегаса	5000.00	10000.00

Повністю сплачені рахунки, тобто, рахунки, сума платежів за якими більша або дорівнює сумі рахунку. Зверніть увагу на застосування підзапиту.

```
SELECT b.bill_num AS "Номер рахунку",
b.bill_date AS "Дата рахунку",
b.bill_sum AS "Сума рахунку",
SUM(p.payment_sum) AS "Сума оплати"
FROM k_bill b, k_payment p
WHERE b.bill_num=p.k_bill_bill_num AND
```

```
b.bill_sum<=
(SELECT SUM(payment_sum) FROM k_payment p2
WHERE b.bill_num=p2.k_bill_bill_num)
GROUP BY b.bill_num
```



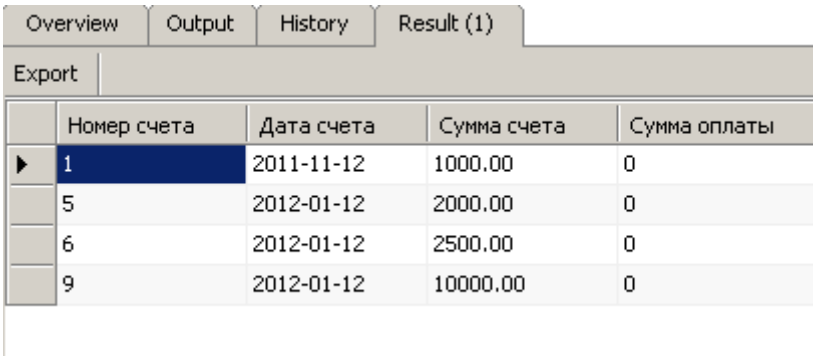
Overview | Output | History | Result (1)

Export

	Номер счета	Дата счета	Сумма счета	Сумма оплаты
▶	3	2012-01-12	2000.00	2000.00

Повністю неоплачені рахунки, якими взагалі немає платежів.

```
SELECT b.bill_num AS "Номер рахунку",
b.bill_date AS "Дата рахунку",
b.bill_sum AS "Сума рахунку",
0 AS "Сума оплати"
FROM k_bill b
WHERE b.bill_num NOT IN (SELECT k_bill_bill_num FROM k_payment)
```



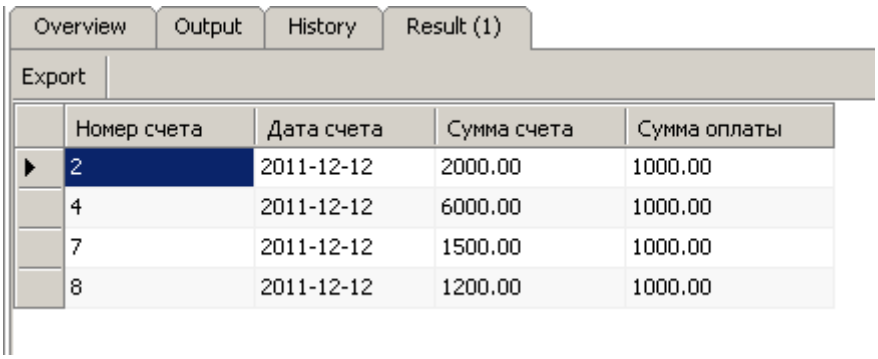
Overview | Output | History | Result (1)

Export

	Номер счета	Дата счета	Сумма счета	Сумма оплаты
▶	1	2011-11-12	1000.00	0
	5	2012-01-12	2000.00	0
	6	2012-01-12	2500.00	0
	9	2012-01-12	10000.00	0

Частково сплачені рахунки. Зверніть увагу, що в цьому прикладі у параметрі FROM замість другої таблиці використовується вкладений SELECT

```
SELECT b.bill_num AS "Номер рахунку",
b.bill_date AS "Дата рахунку",
b.bill_sum AS "Сума рахунку",
p.pay_sum AS "Сума оплати"
FROM k_bill b,
(SELECT k_bill_bill_num, SUM(payment_sum) as pay_sum
FROM k_payment
GROUP BY k_bill_bill_num) p
WHERE b.bill_sum >p.pay_sum AND b.bill_num=p.k_bill_bill_num
```



Номер счета	Дата счета	Сумма счета	Сумма оплаты
2	2011-12-12	2000.00	1000.00
4	2011-12-12	6000.00	1000.00
7	2011-12-12	1500.00	1000.00
8	2011-12-12	1200.00	1000.00

Завдання. Напишіть кілька (не менше 10) цікавих запитів до бази даних. Використовуйте вкладені запити, групування, підсумкові значення, вибірки з кількох таблиць. Якщо ваш запит вимагає введення параметра, замініть його поки що на константу, запити з параметрами можна буде надалі реалізувати за допомогою збережених процедур.

Представлення

На попередньому етапі ми виконували запити до бази даних. Істотним недоліком запитів і те, що й формулювання не зберігаються у базі даних. Щоб подолати цей недолік, вмкористовують **представлення (view)**. Представлення – це об'єкти бази даних, які можна використовувати як віртуальні таблиці. Насправді зберігається лише формулювання команди SELECT, з допомогою якої проводиться вибірка даних із реальних таблиць.

Необхідність у використанні представлення виникає, наприклад, у тому випадку, коли потрібно заборонити доступ користувача до окремих стовпців або рядків таблиці – тоді можна просто написати представлення, в якому ці стовпці чи рядки не будуть присутніми, і надати доступ користувачеві саме до цього представлення, а не до реальної таблиці.

Іншою корисною можливістю є обчислення значень, які не зберігаються безпосередньо в таблиці, але можуть бути розраховані. Представлення, як і запит, може містити інформацію з різних таблиць.

Представлення можуть бути **поновлюваними** (тобто надавати можливість не тільки читання, а й зміни даних у вихідних таблицях) та **непоновлюваними.** Представлення буде поновлюваним лише в тому випадку, якщо його структура така, що SQL server може точно визначити, в які рядки яких таблиць потрібно помістити змінені дані. Непоновлюваними будуть, наприклад, представлення, що містять підсумкові дані та угруповання.

Для створення представлень використовується команда CREATE VIEW.

Короткий формат цієї команди:

CREATE VIEW ім'я_подання AS <Команда_SELECT>

Наприклад, створимо представлення, що містить список договорів та їх кураторів для відділу з номером 1. Чи буде це представлення поновлюваним?

CREATE VIEW k_contract1

AS

SELECT k_contract.contract_num, k_contract.contract_date,

k_contract.contract_type, k_contract.k_firm_firm_num,

k_staff.staff_name

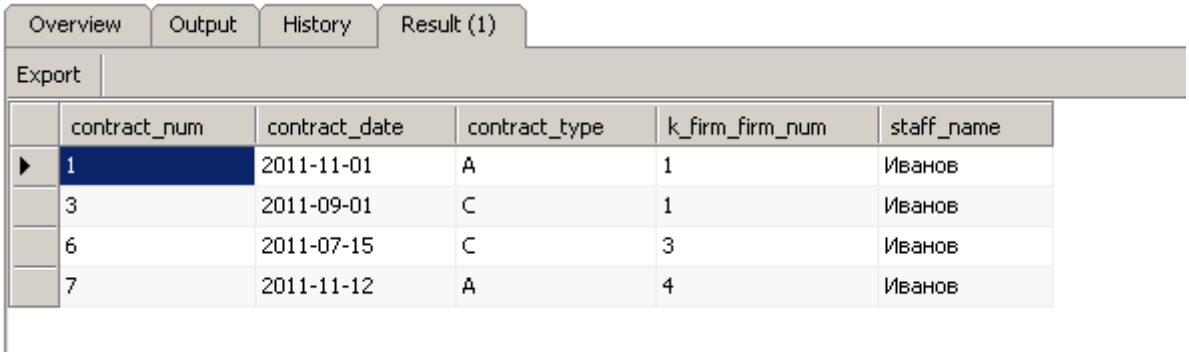
FROM k_contract INNER JOIN

k_staff ON k_contract.k_staff_staff_num = k_staff.staff_num

WHERE k_dept_dept_num = 1

Для перегляду представлення слід виконати команду

SELECT * FROM k_contract1



contract_num	contract_date	contract_type	k_firm_firm_num	staff_name
1	2011-11-01	A	1	Иванов
3	2011-09-01	C	1	Иванов
6	2011-07-15	C	3	Иванов
7	2011-11-12	A	4	Иванов

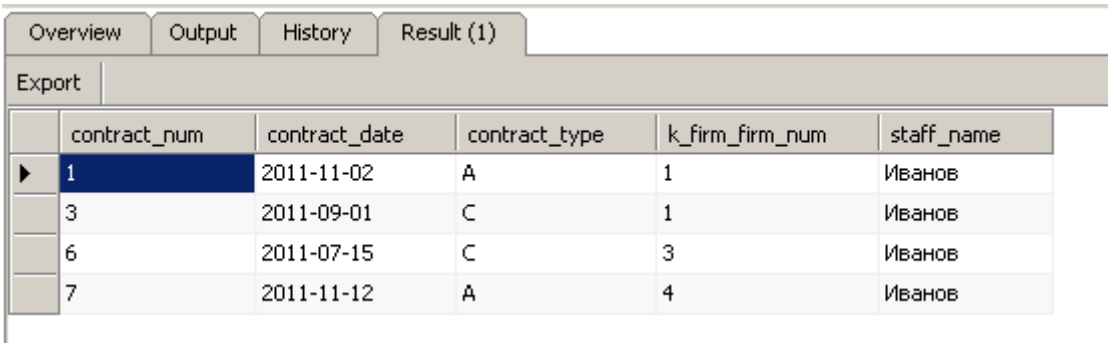
Перевіримо, чи є це представлення поновлюваним. Спробуємо змінити, наприклад, дату рахунку 1:

UPDATE k_contract1 SET contract_date='2011-11-02'

WHERE contract_num=1

Команда виконалася успішно і повторний запит на вибірку дає наступний результат:

SELECT * FROM k_contract1



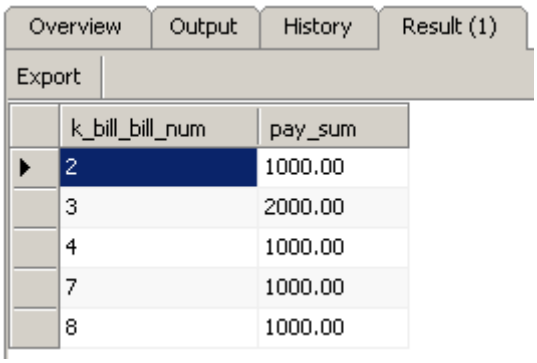
contract_num	contract_date	contract_type	k_firm_firm_num	staff_name
1	2011-11-02	A	1	Иванов
3	2011-09-01	C	1	Иванов
6	2011-07-15	C	3	Иванов
7	2011-11-12	A	4	Иванов

Створимо допоміжне представлення для запитів про повністю оплачені та частково оплачені рахунки (див. попереднє заняття). Це представлення для кожного рахунку містить його номер та суму оплати.

```
CREATE VIEW k_pay_sum
AS
SELECT k_bill_bill_num, SUM(payment_sum) AS pay_sum
FROM k_payment
GROUP BY k_bill_bill_num
```

Для перегляду представлення слід виконати команду

```
SELECT * FROM k_pay_sum.
```



k_bill_bill_num	pay_sum
2	1000.00
3	2000.00
4	1000.00
7	1000.00
8	1000.00

Це представлення не буде оновлюваним. Перевіримо:

```
UPDATE k_pay_sum SET pay_sum=1500 WHERE k_bill_bill_num=1
```

Отримаємо помилку: ERROR 1288: target table k_pay_sum of UPDATE is not updatable. Справді, неможливо змінити значення поля, де знаходиться сума чисел з різних рядків. Але нам і не потрібно оновлювати дані в цьому поданні. Тепер за допомогою цього представлення можна переформулювати запит на вибірку повністю оплачених рахунків, цей запит стане простішим.

Повністю сплачені рахунки

```
SELECT b.bill_num AS "Номер рахунку",
b.bill_date AS "Дата рахунку",
b.bill_sum AS "Сума рахунку",
p.pay_sum AS "Сума оплати"
FROM k_bill b, k_pay_sum p
WHERE b.bill_num=p.k_bill_bill_num AND
b.bill_sum<=p.pay_sum
```

Завдання. Створіть кілька (не менше 3) представлень для вашої бази даних. Чи будуть вони поновлюваними чи ні? Перевірте.

4.5. Збережені процедури

Збережені процедури – це об'єкти бази даних, які є програмами, що маніпулюють даними та виконуються на сервері. Ці програми, крім команд мови SQL, можуть використовувати нечисленні керуючі команди.

Структура процедури, що зберігається наступна:

DELIMITER //

CREATE PROCEDURE ім'я_процедури [(параметри)]

#Код процедури

//

Оголошення змінних має вигляд

DECLARE ім'я_змінної тип_змінної [(довжина)];

Блок операторів потрібно розмістити у операторні дужки BEGIN … END

Оператор присвоєння виглядає так:

SET змінна = значення;

Якщо потрібно присвоїти змінний результат команди SELECT, то використовується наступний формат (багато крапка означає стандартне продовження команди):

SELECT ім'я_стовпця INTO змінна FROM ...;

Умовний оператор має вигляд:

IF умова THEN

Оператор1 або Група операторів1

[ELSE

Оператор2 або Група операторів2]

END IF;

Є кілька операторів циклу, найпоширеніший із них:

WHILE умова DO

Оператор або Група операторів

END WHILE;

Вираз CASE застосовується для вибору на підставі кількох опцій:

CASE вираз

WHEN варіант1 THEN вираз1

WHEN варіант2 THEN вираз2

…

ELSE виразN

END CASE;

Для видалення процедур використовується команда:

DROP PROCEDURE IF EXISTS Ім'я_процедури;

Створимо процедуру, яка як параметр отримує прізвище співробітника та друкує список усіх договорів, якими він займається.

```
DELIMITER //
CREATE PROCEDURE show_contracts
(v_staff_name CHAR(50))
BEGIN
SELECT contract_num, contract_date, contract_type
FROM k_contract з JOIN k_staff s ON
c.k_staff_staff_num=s.staff_num
WHERE s.staff_name=v_staff_name;
END//
```

Для запуску цієї процедури потрібно виконати команду

```
CALL show_contracts('Іванів');
```

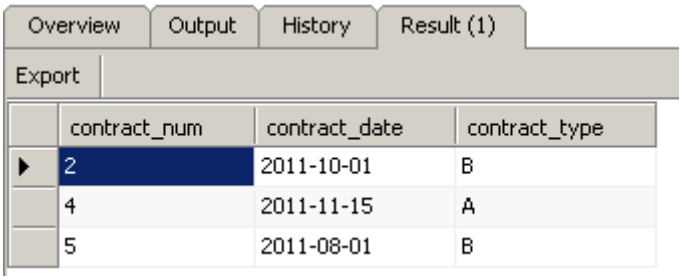
Overview | Output | History | Result (1)

Export

contract_num	contract_date	contract_type
1	2011-11-02	A
3	2011-09-01	C
6	2011-07-15	C
7	2011-11-12	A

або

```
CALL show_contracts ( "Петрів");
```



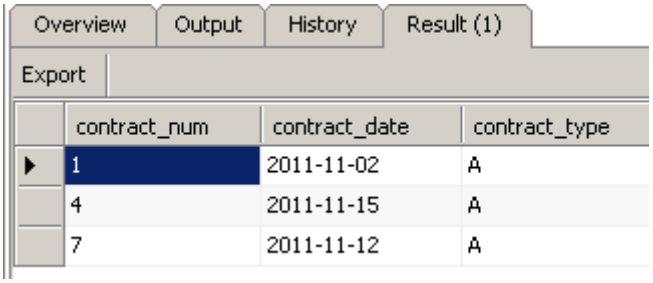
contract_num	contract_date	contract_type
2	2011-10-01	B
4	2011-11-15	A
5	2011-08-01	B

Створимо процедуру, яка отримує номер місяця та номер року та друкує договори за зазначений період.

```
DELIMITER //
CREATE PROCEDURE find_contracts_by_month_and_year
(v_month INT, v_year INT)
BEGIN
SELECT contract_num, contract_date, contract_type
FROM k_contract
WHERE MONTH(contract_date)=v_month AND
YEAR(contract_date)=v_year;
END//
```

Виконаємо процедуру:

```
CALL find_contracts_by_month_and_year(11,2011);
```



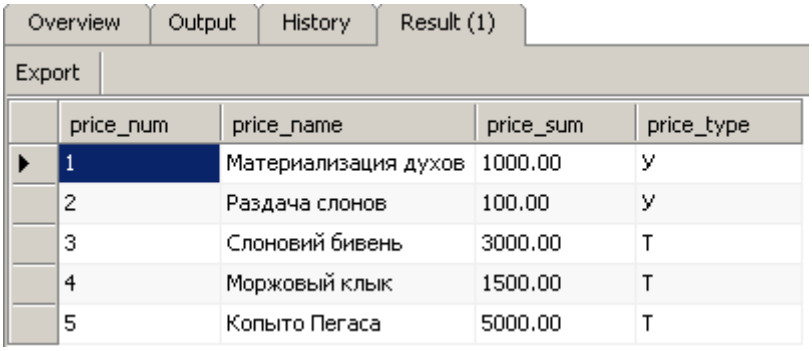
contract_num	contract_date	contract_type
1	2011-11-02	A
4	2011-11-15	A
7	2011-11-12	A

Створимо процедуру «Розпродаж», яка знаходить товар, який не продається (за кількістю) і знижує його ціну на заданий відсоток.

```
DELIMITER //
CREATE PROCEDURE clearance (% INT)
BEGIN
DECLARE p INT;
```

```
IF percent > 0 AND percent < 100 THEN
SELECT k_price_price_num INTO p FROM k_protokol
GROUP BY k_price_price_num
HAVING SUM(kolvo)<=ALL
(SELECT SUM(kolvo) FROM k_protokol
GROUP BY k_price_price_num);
UPDATE k_price
SET price_sum=price_sum*(100%)/100
WHERE price_num = p;
END IF;
END//
```

Таблиця "Прайс-лист" до виконання процедури:



Overview | Output | History | Result (1)

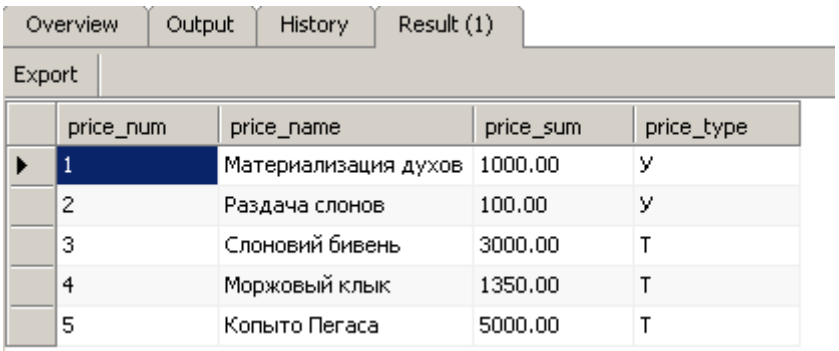
Export

price_num	price_name	price_sum	price_type
1	Материализация духов	1000.00	У
2	Раздача слонов	100.00	У
3	Слоновий бивень	3000.00	Т
4	Моржовый клык	1500.00	Т
5	Копыто Пегаса	5000.00	Т

Для запуску цієї процедури потрібно виконати, наприклад, команду

```
CALL clearance(10);
```

Вміст таблиці "Прайс-лист" після виконання процедури:



Overview | Output | History | Result (1)

Export

price_num	price_name	price_sum	price_type
1	Материализация духов	1000.00	У
2	Раздача слонов	100.00	У
3	Слоновий бивень	3000.00	Т
4	Моржовый клык	1350.00	Т
5	Копыто Пегаса	5000.00	Т

Як бачимо, товар із номером 4 у прайс-листі знижено в ціні на 10%.

А що станеться, якщо в нашій базі даних є кілька товарів, кількість продажу яких є мінімальною? На жаль, у нашому випадку при виконанні команди SELECT процедура видасть помилку: Error Code: 1172. Коли команді SELECT вибирається відразу кілька значень поля k_price з таблиці k_protokol, неможливо привласнити ці кілька значень однієї змінної p. Цю ситуацію можна опрацювати з допомогою курсорів.

Курсор

Курсор (current set of record) – це тимчасовий набір рядків, які можна перебирати послідовно, з першого до останнього.

Для роботи з курсорами є наступні команди.

Оголошення курсору:

DECLARE имя_курсора CURSOR FOR SELECT текст_запиту;

Таким чином, будь-який курсор створюється на основі оператора SELECT.

Відкриття курсору:

OPEN ім'я_курсора;

Тільки після відкриття курсору він стає активним, і з нього можна читати рядки.

Читання значень із поточного рядка курсору до набору змінних та переміщення вказівника на наступний рядок:

FETCH ім'я_курсора INTO список_змінних;

Змінні у списку повинні мати таку ж кількість і тип, як і стовпці курсора.

Закриття курсору:

CLOSE ім'я_курсора;

При переборі рядків курсору виникає необхідність перевірки, чи ми добралися до кінця курсору чи ще ні. У різних СУБД при цьому можуть бути передбачені різні кошти. У СУБД MySQL призначається обробник стану "NOT FOUND". Визначати його потрібно відразу після опису структури курсора:

DECLARE CONTINUE HANDLER FOR NOT FOUND оператор;

Наприклад, цей обробник може виглядати так:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

Тепер спробуємо модифікувати процедуру «Розпродаж» з урахуванням того, що ми можемо мати кілька товарів з мінімальною кількістю продажів.

```
DELIMITER //
CREATE PROCEDURE clearance2 (percent INT)
BEGIN
DECLARE p INT;
DECLARE finished NUMERIC(1);
- оголошуємо курсор на основі деякого оператора SELECT

DECLARE my_cursor CURSOR

FOR SELECT k_price_price_num FROM k_protokol

GROUP BY k_price_price_num

HAVING SUM(kolvo)<=ALL

(SELECT SUM(kolvo) FROM k_protokol

GROUP BY k_price_price_num);

-- оголошуємо обробник стану NOT FOUND

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

IF percent > 0 AND percent < 100 THEN

SET finished = 0;

OPEN my_cursor; - відкриваємо курсор

FETCH my_cursor INTO p; - читаємо перший рядок

WHILE( finished != 1) DO

UPDATE k_price

SET price_sum=price_sum*(100%)/100

WHERE price_num = p;

FETCH my_cursor INTO p; - читаємо черговий рядок

END WHILE;

CLOSE my_cursor; - Закриваємо курсор

END IF;

END//
```

Завдання. Створіть кілька (не менше 2) процедур, що зберігаються для вашої бази даних. Бажано використовувати параметри. Запустіть процедури виконання.

4.6. Тригери

Тригери – це процедури спеціального вигляду, що зберігаються, які автоматично виконуються при зміні даних за допомогою операторів INSERT, UPDATE і DELETE.

Тригер створюється для певної таблиці, але може використовувати дані інших таблиць та об'єкти інших баз даних.

Оператор CREATE TRIGGER дозволяє створити новий тригер і має наступний синтаксис:

CREATE TRIGGER ім'я_тригера час_тригера подія_тригера

ON имя_таблицы FOR EACH ROW

тіло_тригера

Конструкція час_тригера вказує момент виконання тригера і може приймати два значення:

BEFORE - дії триреру проводяться до виконання операції зміни таблиці;

AFTER - дії триреру виконуються після виконання операції зміни таблиці.

Конструкція подія_тригера може приймати значення

INSERT, UPDATE та DELETE.

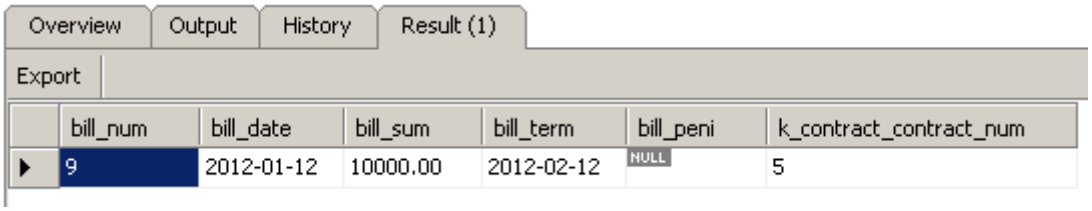
Ідентифікатори OLD та NEW означають старе та нове значення змінюваних даних.

Розглянемо приклад тригера вставки, який викликається під час виконання команди INSERT у таблиці протоколів рахунків. При додаванні нової позиції в рахунку нам потрібно перерахувати його загальну суму:

```
DELIMITER //
-- тригер запускається перед додаванням рядка до протоколу
--рахунків
CREATE TRIGGER ins_prot BEFORE INSERT ON k_protokol
FOR EACH ROW
BEGIN
DECLARE v_kolvo NUMERIC(6); --кількість
DECLARE v_bill_num NUMERIC(6); --номер рахунку
DECLARE v_price_num NUMERIC(6); --номер товару
DECLARE v_price_sum NUMERIC(9,2); - ціна товару
SET v_kolvo = New.kolvo;
SET v_bill_num=New.k_bill_bill_num;
SET v_price_num=New.k_price_price_num;
IF v_kolvo>0 THEN - тільки якщо кількість >0
--з прайс-листа отримуємо ціну товару
SELECT p.price_sum INTO v_price_sum FROM k_price p
```

```
WHERE p.price_num=v_price_num;
-- оновлюємо загальну суму рахунку
UPDATE k_bill
SET bill_sum=bill_sum+v_kolvo*v_price_sum
WHERE k_bill.bill_num=v_bill_num;
- ціну товару продублюємо в протоколі рахунку
SET New.price_sum=v_price_sum;
END IF;
END//
```

Протестуємо тригер. Попередньо подивимося інформацію про рахунок №9:

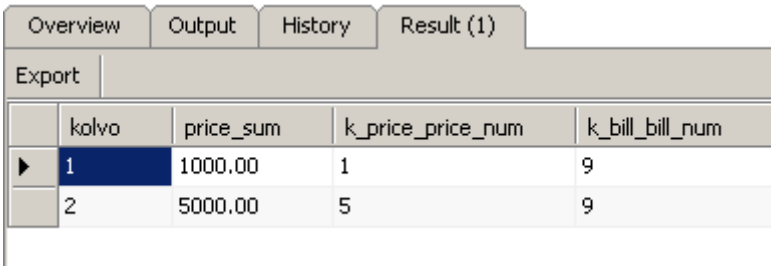

Overview | Output | History | Result (1)

Export

bill_num	bill_date	bill_sum	bill_term	bill_peni	k_contract_contract_num
9	2012-01-12	10000.00	2012-02-12	NULL	5

```
SELECT * FROM k_bill WHERE bill_num = 9;
```

Тепер додамо новий рядок до протоколу цього рахунку: додаємо 1 штуку товару з номером 1 і ціною 1000 грн. (Ціну беремо з таблиці k_price).


Overview | Output | History | Result (1)

Export

kolvo	price_sum	k_price_price_num	k_bill_bill_num
1	1000.00	1	9
2	5000.00	5	9

```
INSERT INTO k_protokol
(kolvo, price_sum, k_price_price_num, k_bill_bill_num)
VALUES(1, 0, 1, 9);
```

Подивимося, як змінився рахунок №9:

```
SELECT * FROM k_bill WHERE bill_num = 9;
```

Загальна сума рахунка збільшилася на 1000 грн.

Подивимося таблицю протоколу рахунків:

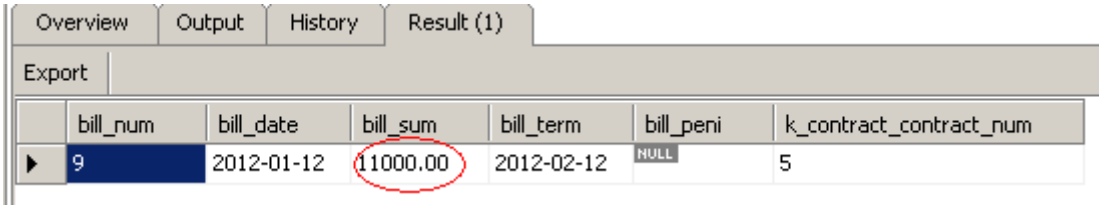
```
SELECT * FROM k_protokol WHERE k_bill_bill_num=9;
```

У доданому рядку з номером товару 1 ціна заповнилася автоматично із прайс-листа.

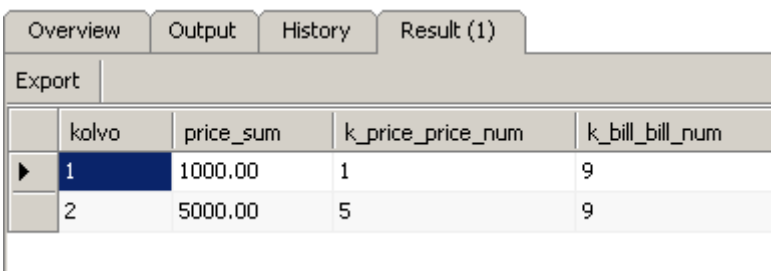
Тепер створимо тригер для операції видалення з тієї ж таблиці. При видаленні рядка з протоколу рахунку має зменшуватись загальна сума рахунку.

```
-- тригер запускається перед видаленням рядка з протоколу рахунків
DELIMITER //
CREATE TRIGGER del_prot BEFORE DELETE ON k_protokol
FOR EACH ROW
BEGIN
DECLARE v_kolvo NUMERIC(6); -- кількість
DECLARE v_bill_num NUMERIC(6); - Номер рахунку
DECLARE v_price_sum NUMERIC(9,2); - ціна товару
SET v_kolvo = Old.kolvo;
SET v_bill_num=Old.k_bill_bill_num;
SET v_price_sum=Old.price_sum;
IF v_kolvo>0 THEN - тільки якщо кількість >0
-- оновлюємо загальну суму рахунку
UPDATE k_bill
SET bill_sum=bill_sum-v_kolvo*v_price_sum
WHERE k_bill.bill_num=v_bill_num;
END IF;
END//
```

Протестуємо тригер. Спочатку подивимося зміст таблиць до виконання операції видалення:




```
SELECT * FROM k_bill WHERE bill_num = 9;
```



SELECT * FROM k_protokol WHERE k_bill_bill_num=9;

Тепер видалимо з протоколу рахунків інформацію про товар із номером 5:

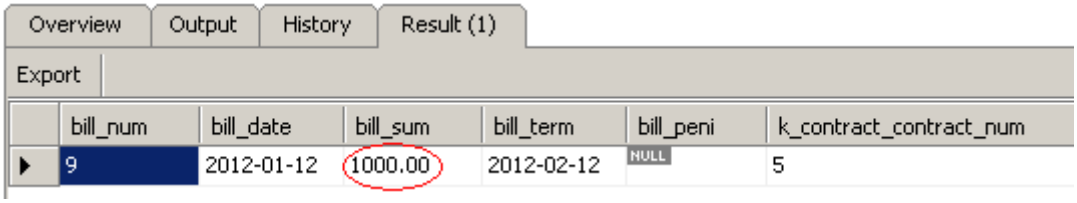
DELETE FROM k_protokol

WHERE k_bill_bill_num=9 AND k_price_price_num=5;

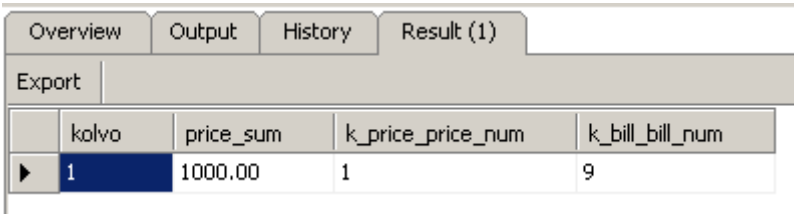
Знову подивимося вміст таблиці k_bill:

SELECT * FROM k_bill WHERE bill_num = 9;

Загальна сума рахунку зменшилася до 1000 грн.



SELECT * FROM k_protokol WHERE k_bill_bill_num=9;



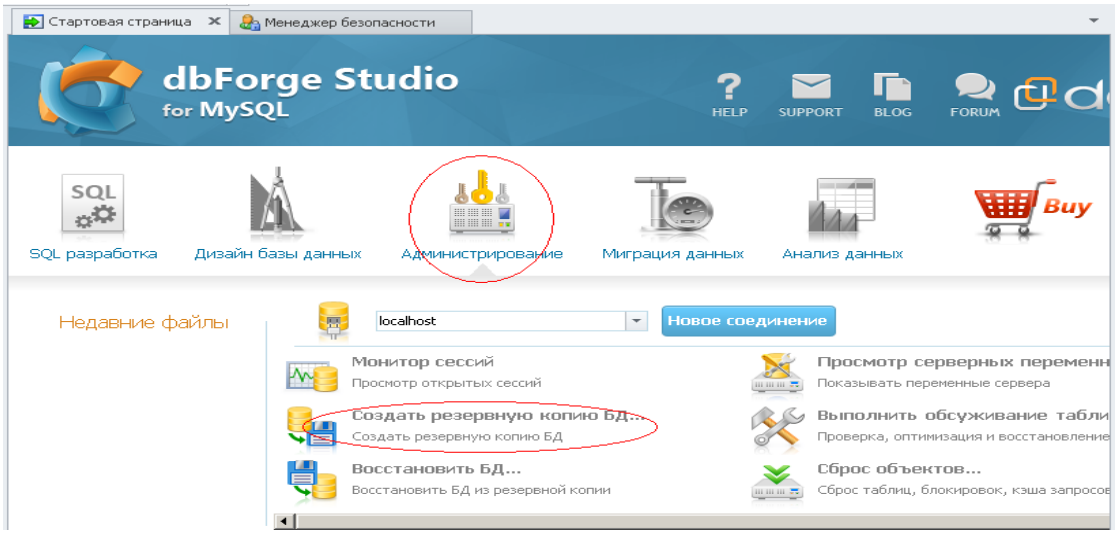
Завдання. Створіть та протестуйте принаймні 1 тригер для вашої бази даних.

5. Dbforge Studio від Devart

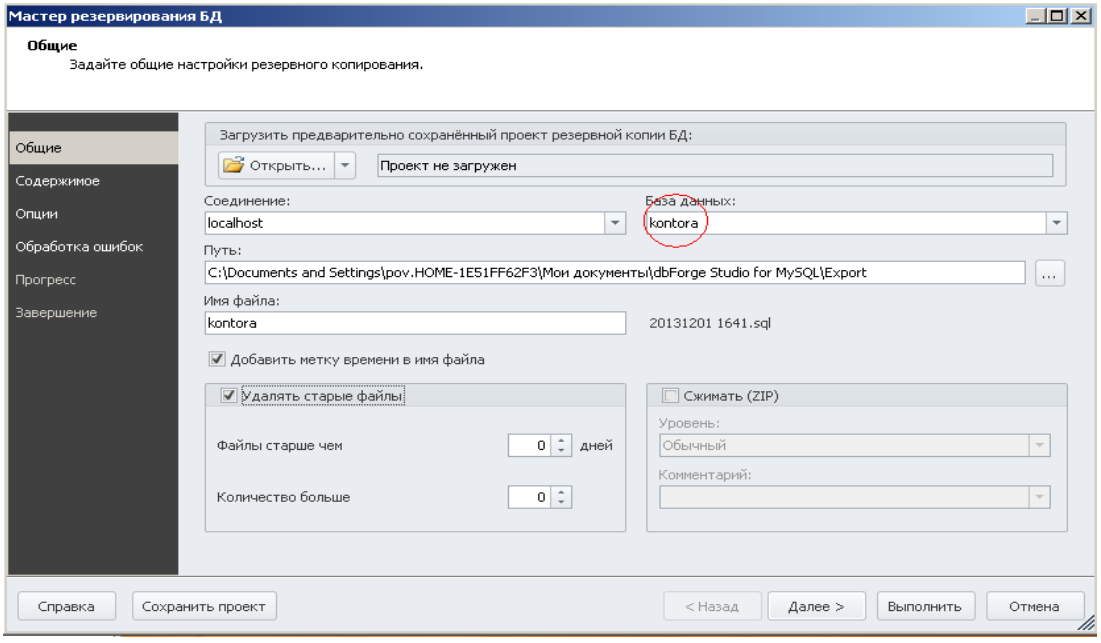
Для роботи з сервером MySQL є альтернативні інтерфейси, що дозволяють здійснювати розробку баз даних і адміністрування сервера. Один із найцікавіших варіантів – dbForge Studio від компанії Devart. Наведемо кілька ілюстрацій найбільш типових адміністративних функцій.

Резервне копіювання бази даних у dbforge Studio for mySQL

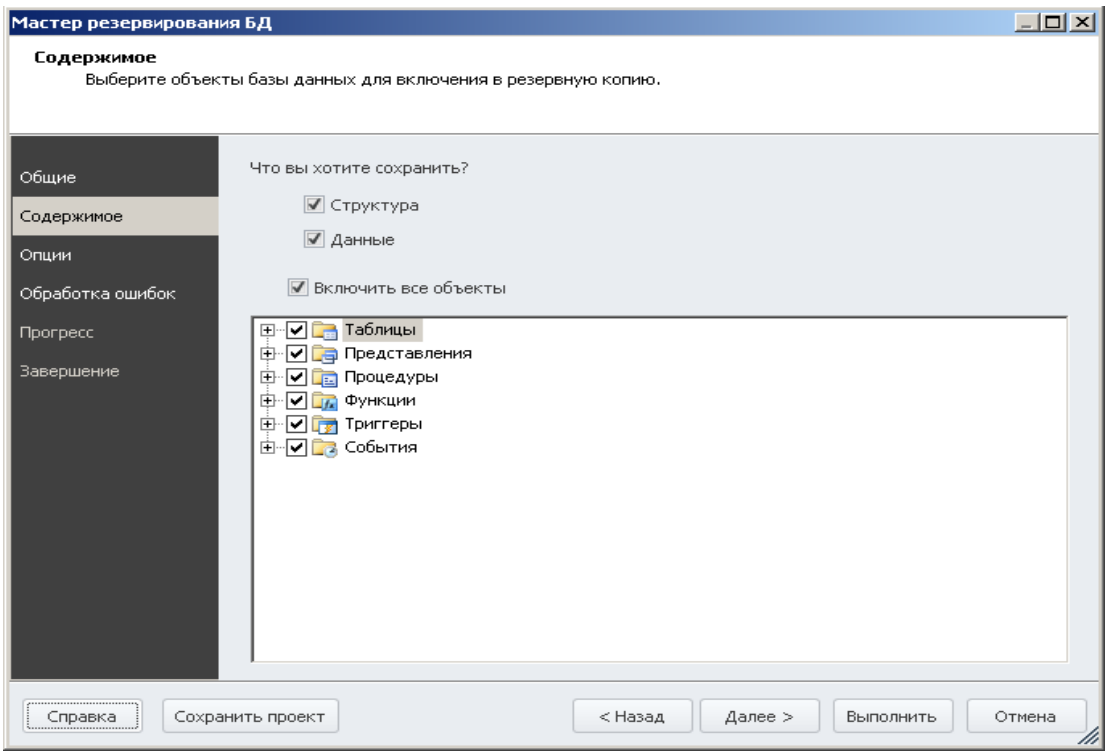
У секції Адміністрація вибираємо Створити резервну копію БД:



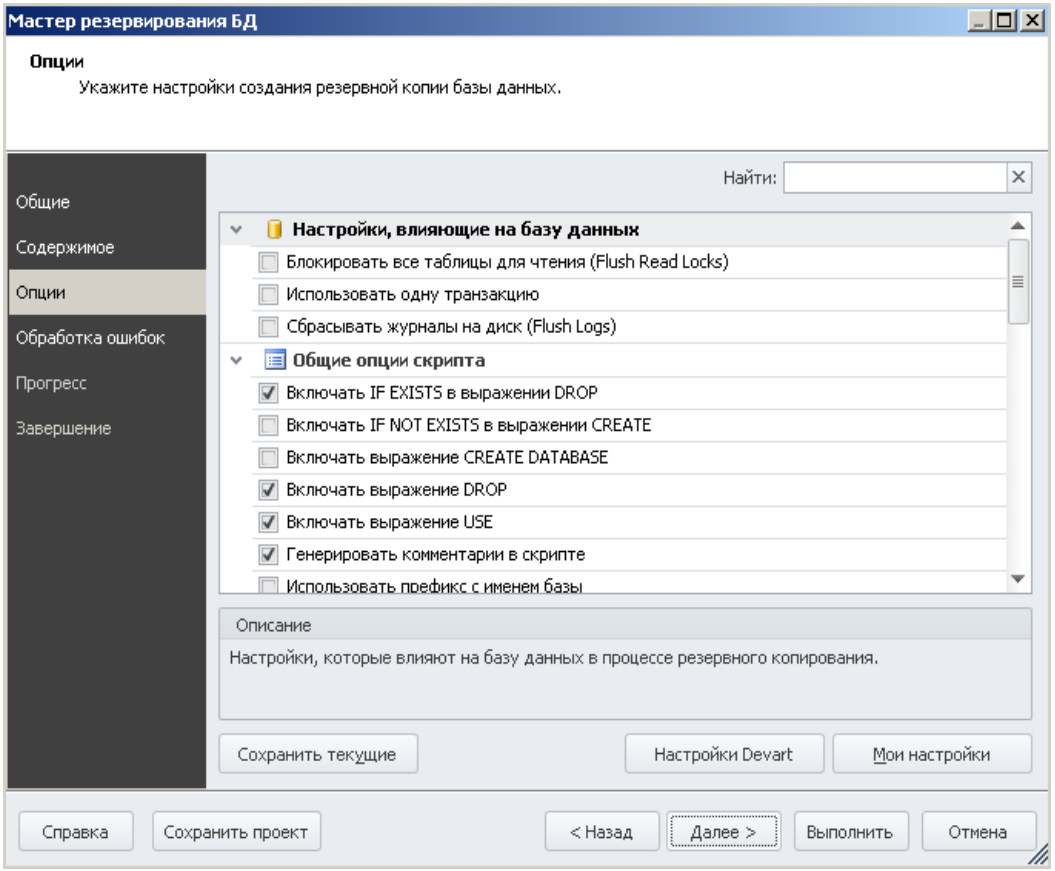
У наступному вікні слід вибрати потрібну базу даних:



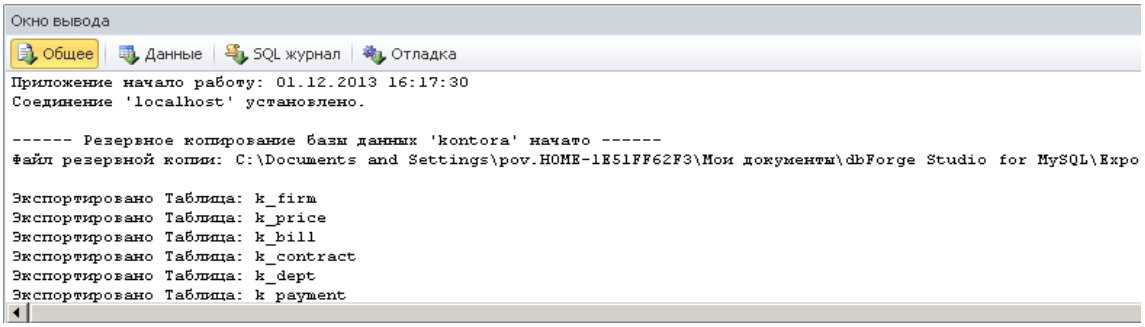
Далі можна зазначити, які саме типи об'єктів бази даних слід архівувати:

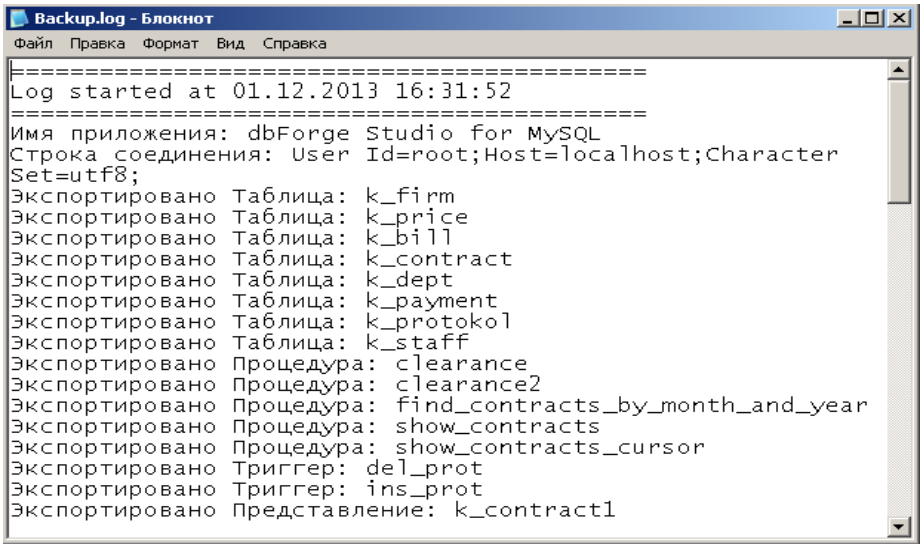


У наступному вікні можна уточнити параметри резервного копіювання:



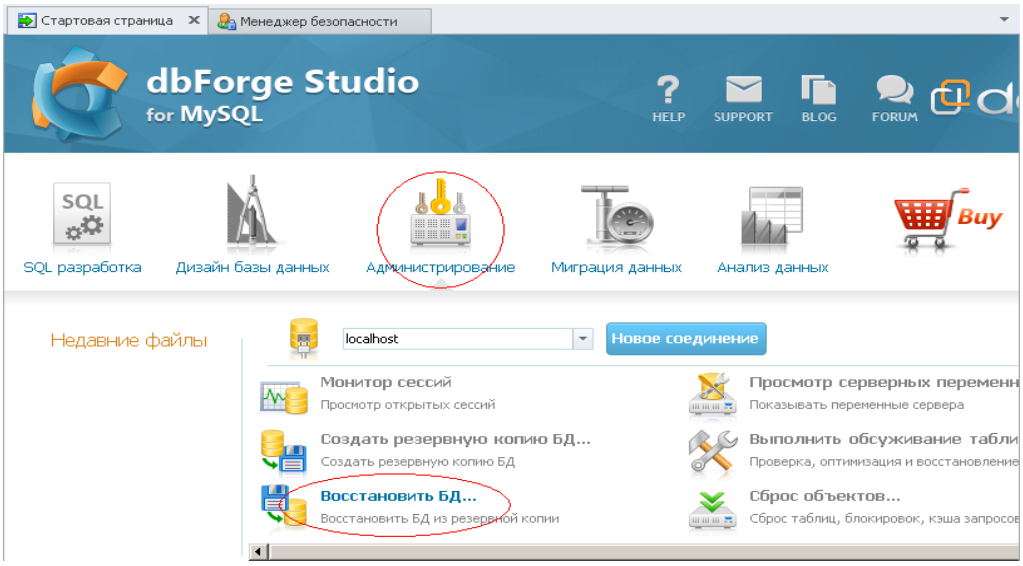
Після натискання кнопки Виконати відбувається архівування, і протокол роботи може бути виведений на екран:



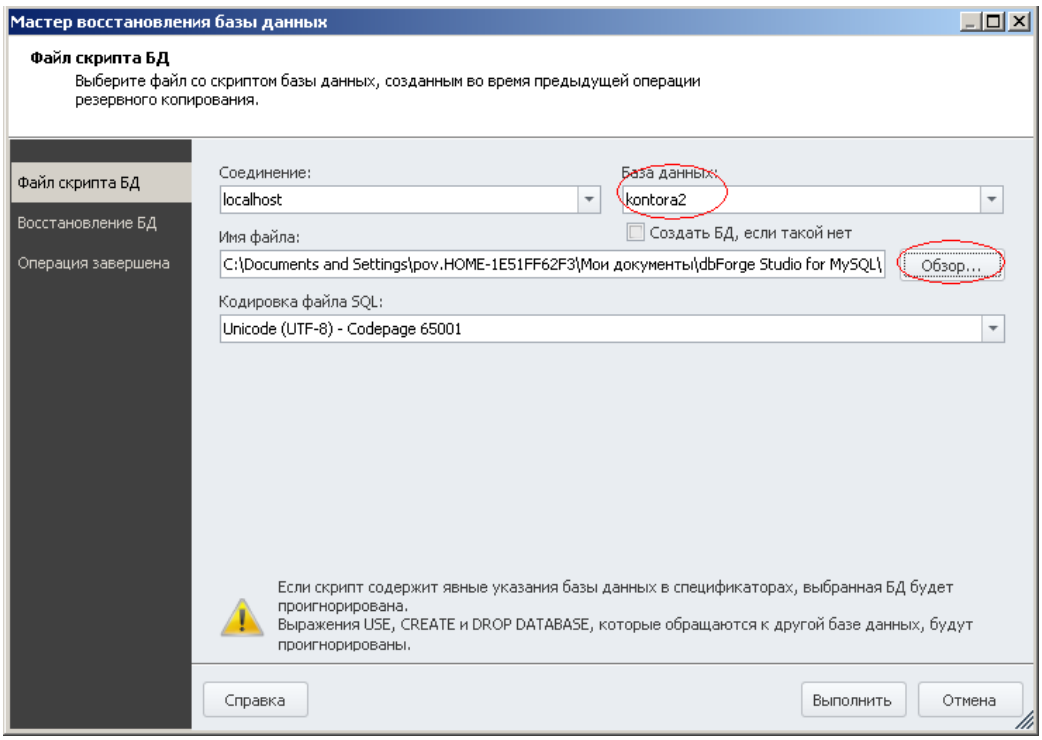


Відновлення бази даних у dbforge Studio for mySQL

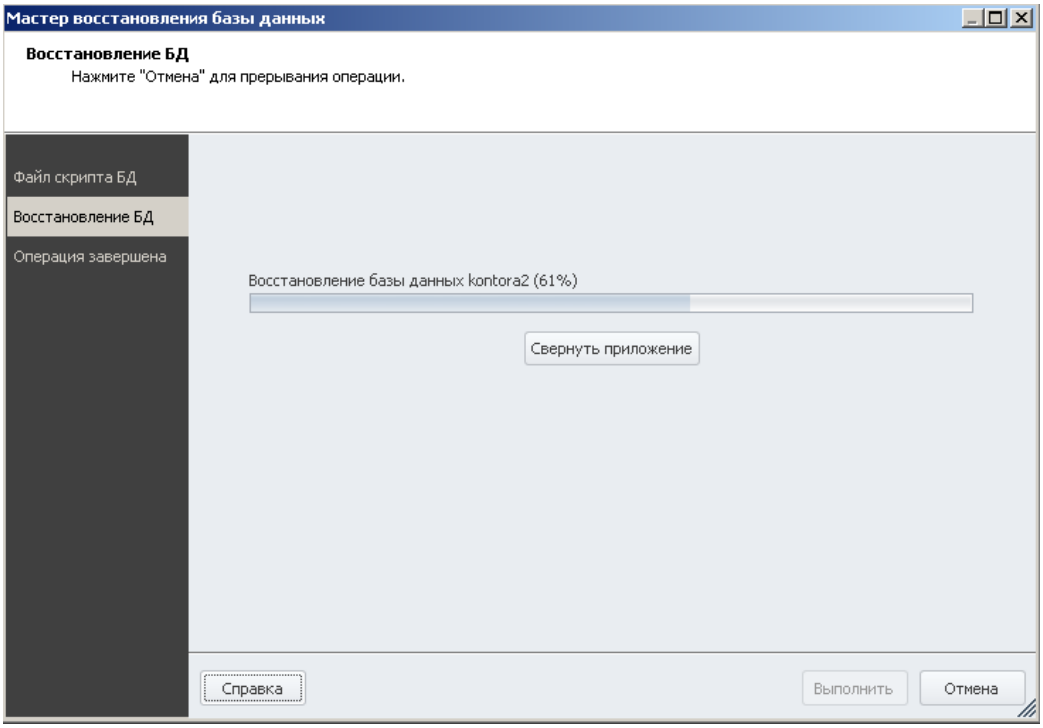
У секції Адміністрація є пункт відновлення бази даних з архівної копії:



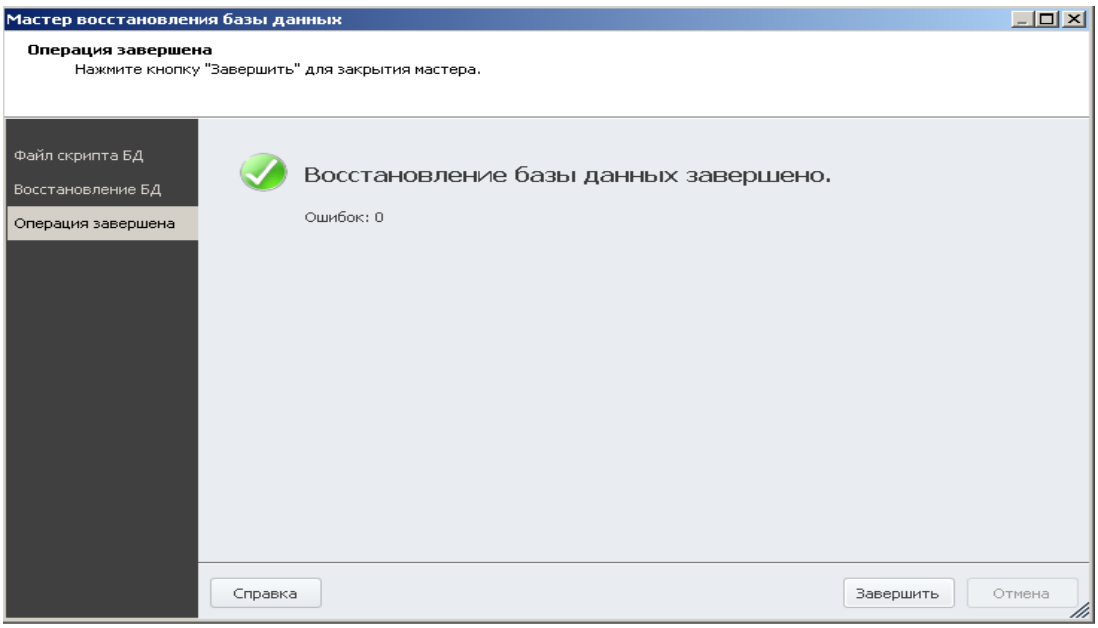
На першому кроці потрібно задати ім'я для бази даних, що відновлюється:



І натиснути на кнопку Виконати:

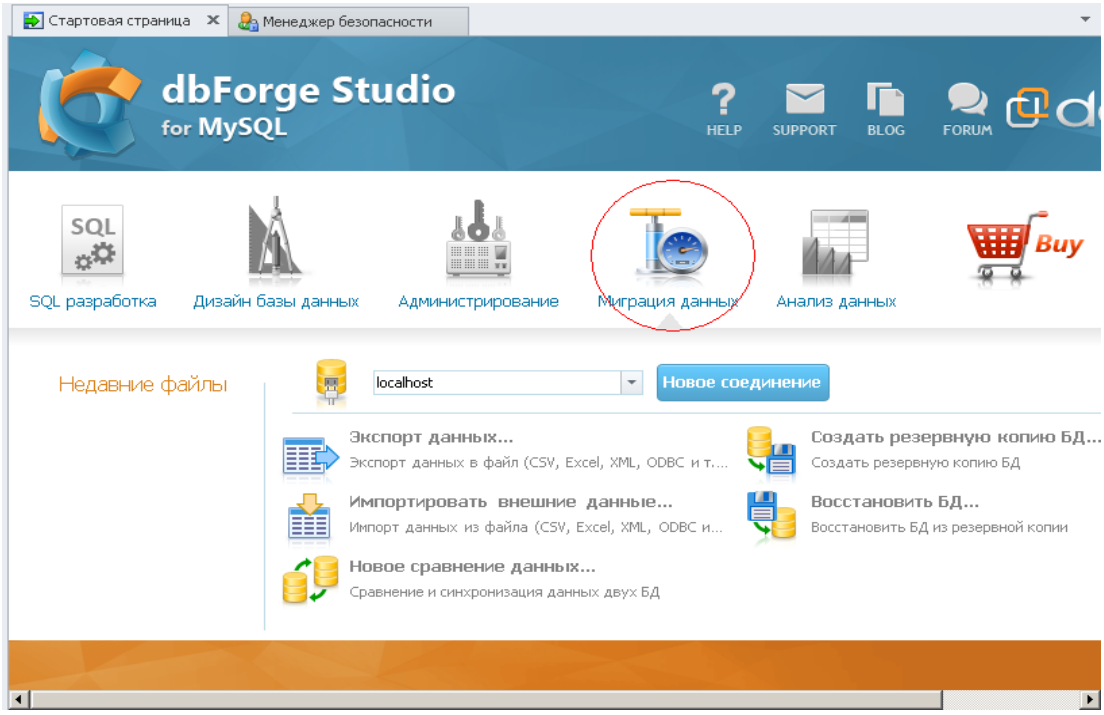


Після виконання операції відновлення буде виведено повідомлення про її результати:



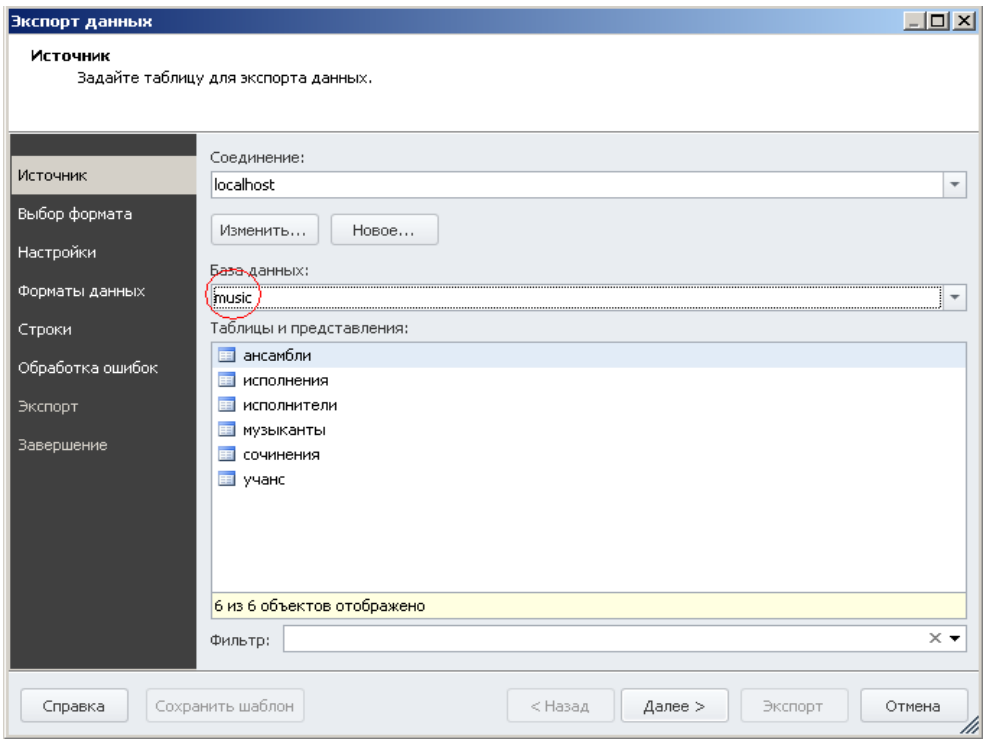
Міграція даних у dbforge Studio for mySQL

Під міграцією даних розуміють операції імпорту та експорту – обмін даними із додатками інших форматів.

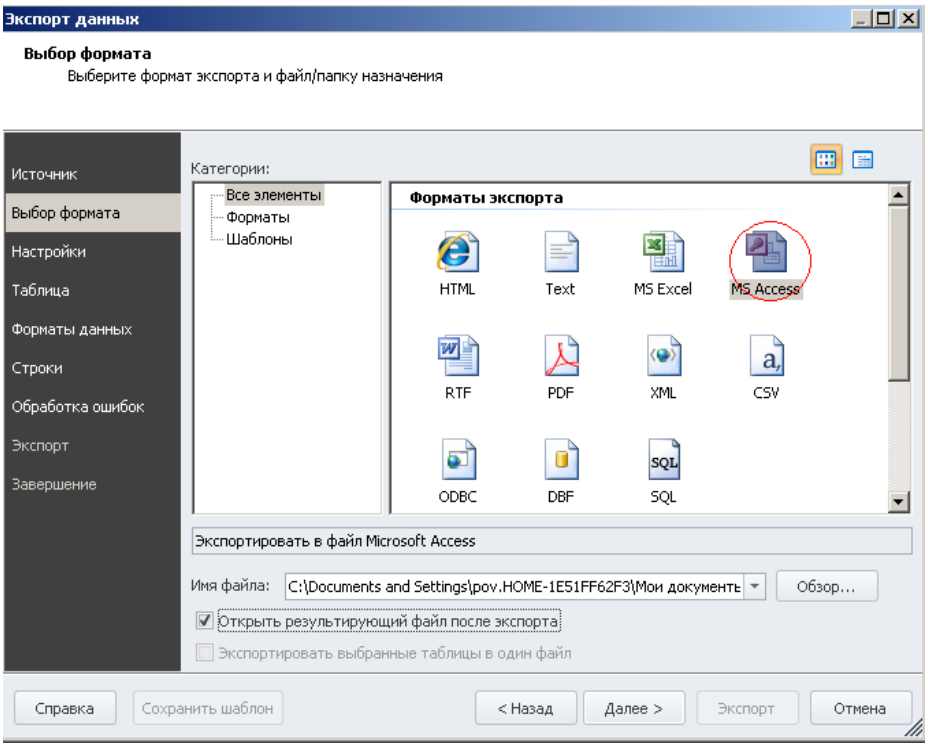


Експорт даних у dbforge Studio for mySQL

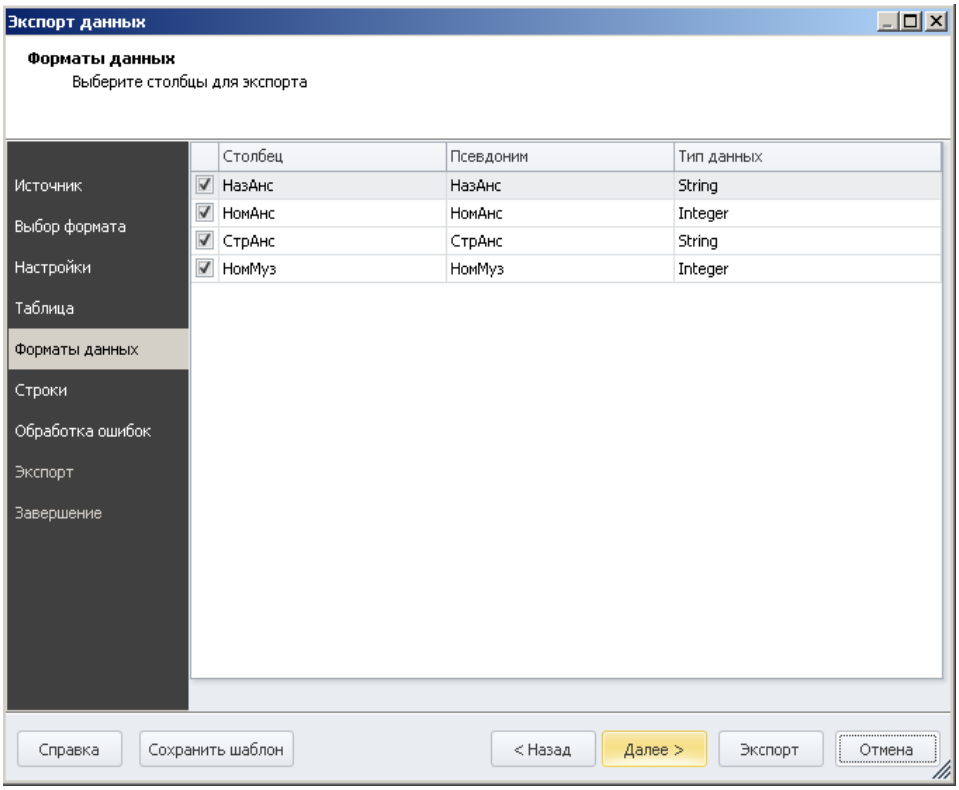
У першому вікні слід вибрати базу даних та її об'єкти для експорту. Виберемо базу даних music та таблицю Ансамблі:



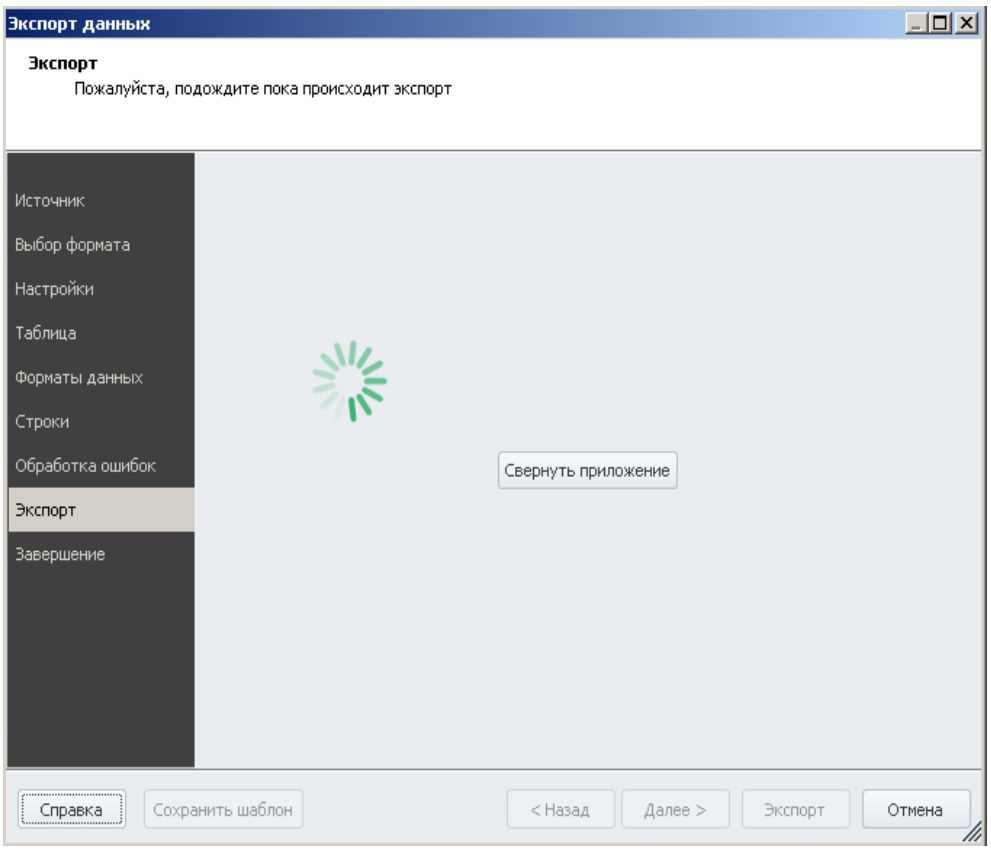
Далі вибираємо формат, у який імпортуватимуться дані. Наприклад візьмемо формат MS Access:

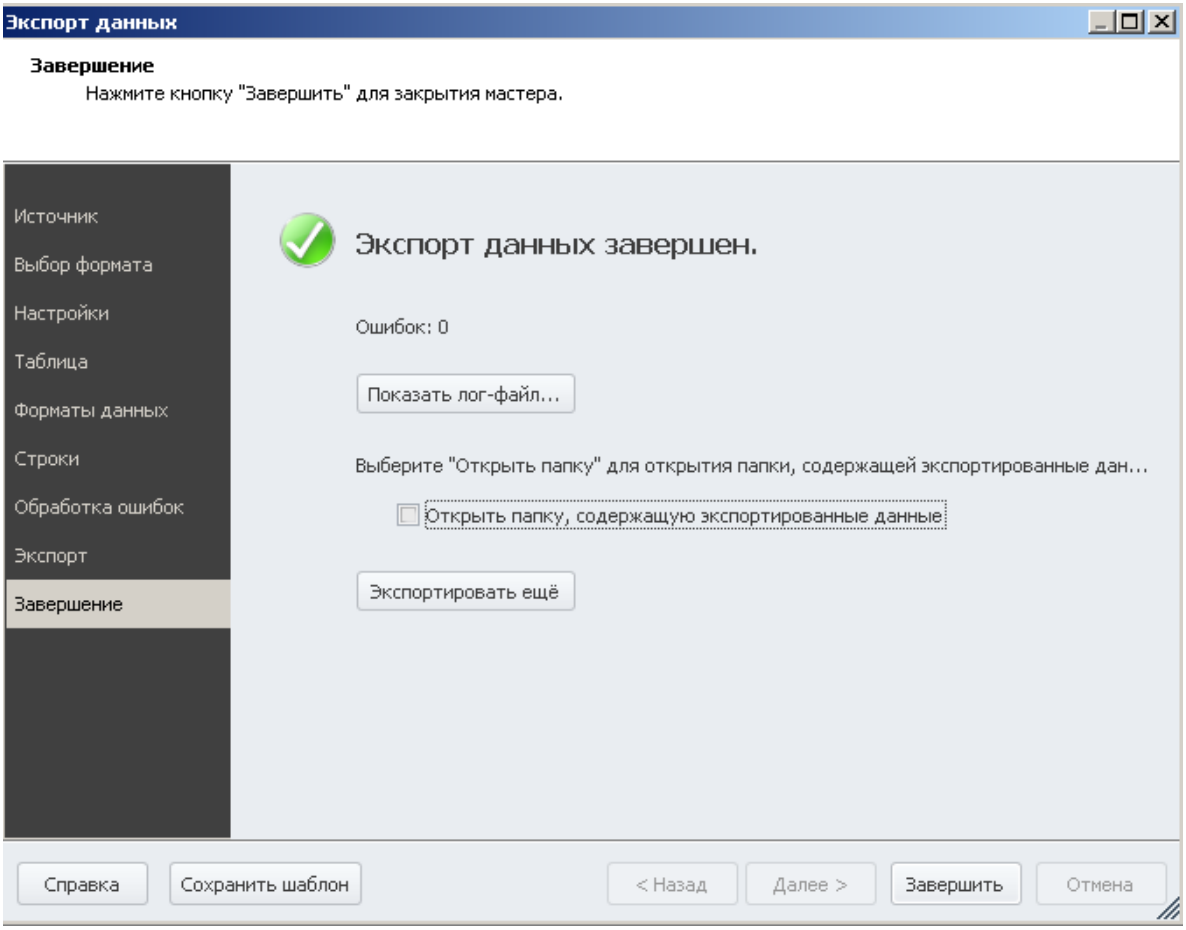


Далі можна уточнити структуру даних, що експортуються:



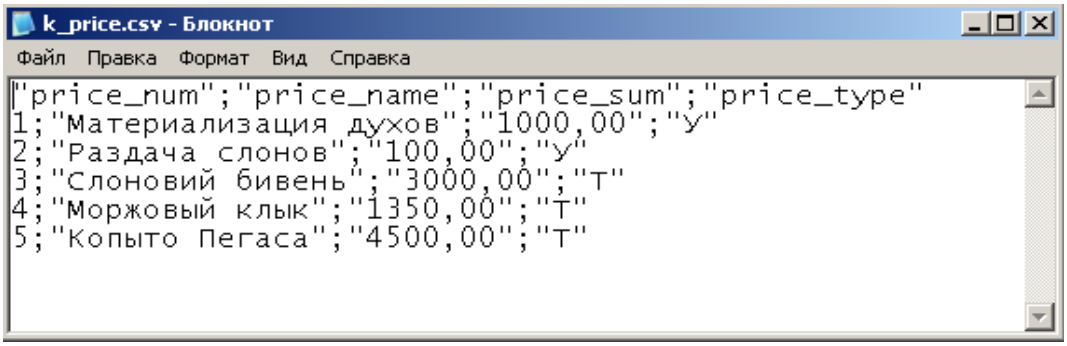
І можна запускати експорт. Операція займає деякий час:



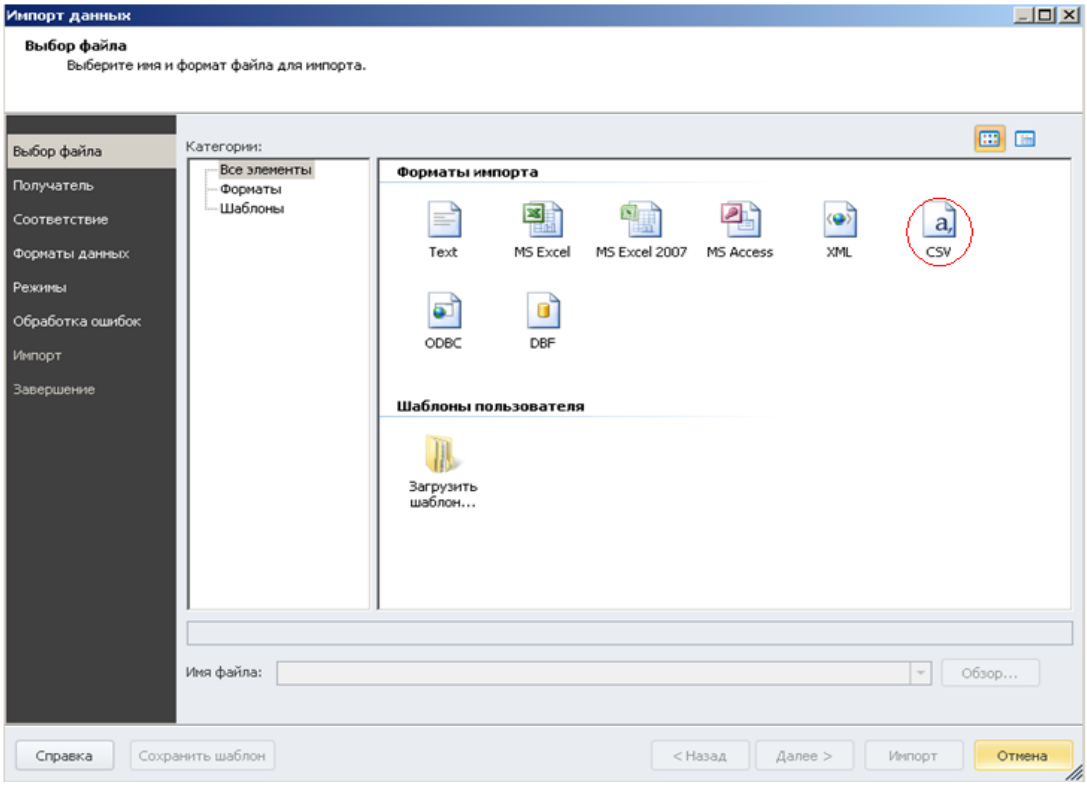


Імпорт даних у dbforge Studio for mySQL

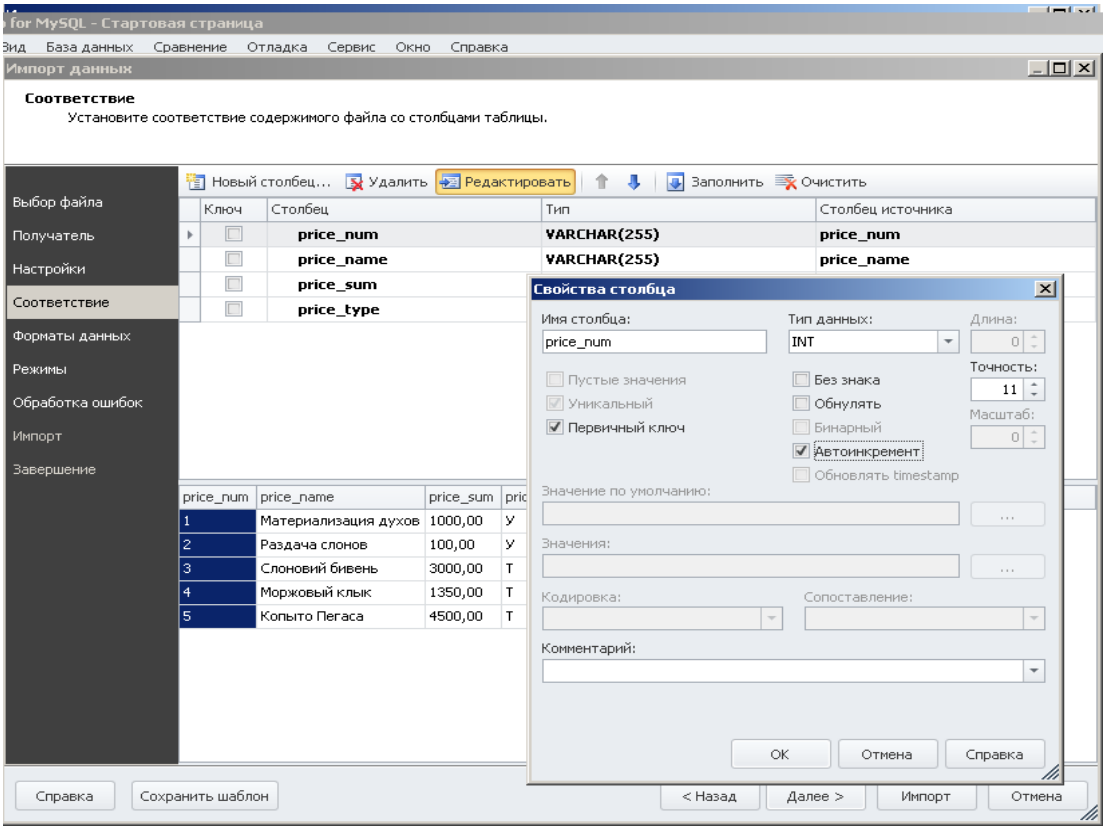
Так само можна виконувати і зворотний процес – імпорт даних із зовнішніх джерел. Нехай у нас є структурований текстовий файл із роздільниками, що містить інформацію про товари:



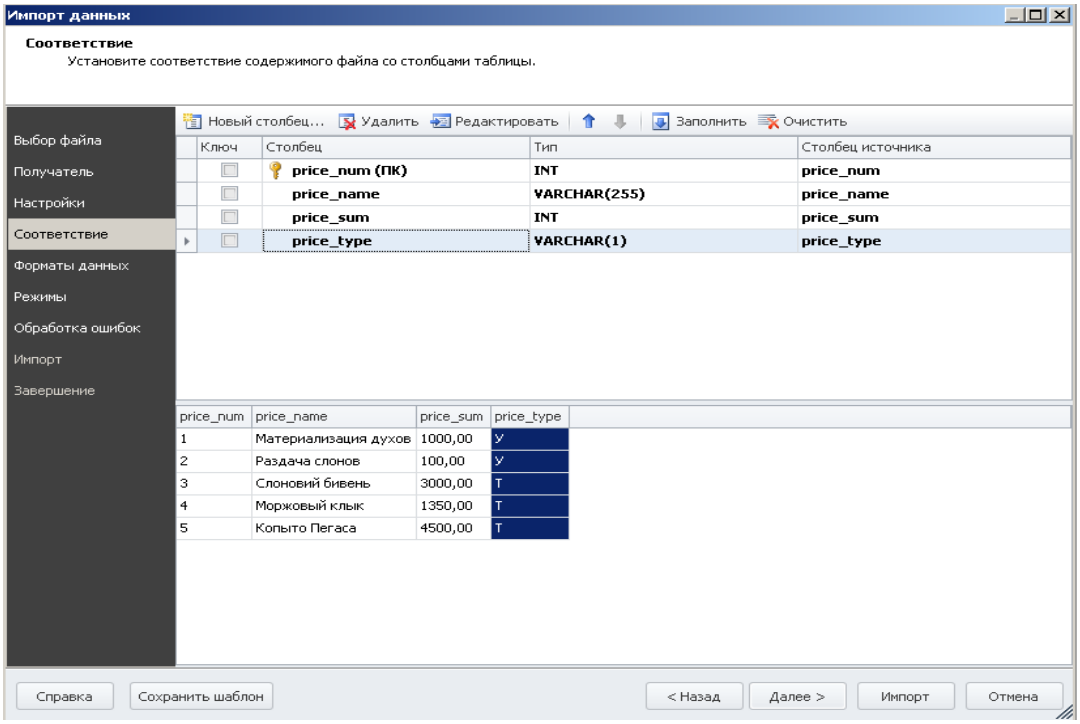
Такі файли зазвичай мають тип csv. Запускаємо майстер для імпорту, на першому екрані вибираємо тип файлу:



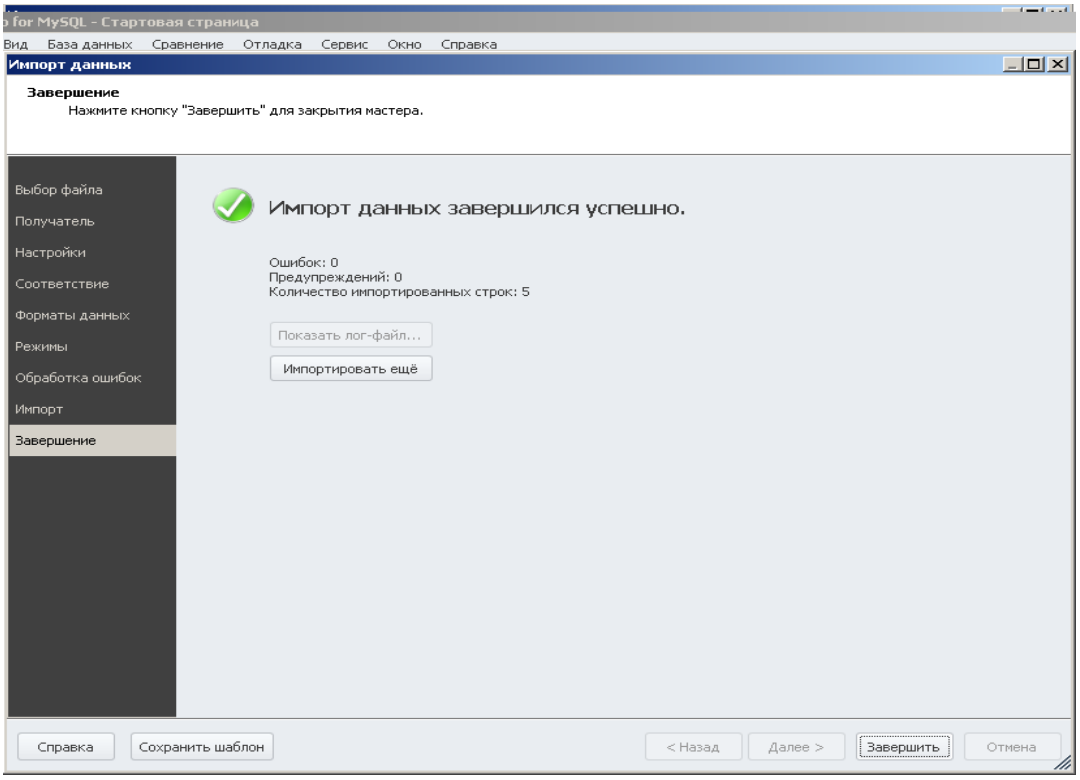
Далі майстер імпорту аналізує дані та пропонує для них відповідні типи та розміри. Запропоновані значення за необхідності можна змінити, і навіть призначити у таблиці первинний ключ та інші обмеження.



Ось так виглядає структура таблиці після налаштування:



Після завершення операції видається повідомлення про її результат:



БІБЛОБГРАФІЯ

1. Уроки SQL - aCode
2. https://arato.inf.unideb.hu/ispany.marton/Database/Lectures2023/7_ermodell.pdf
3. https://users.iit.uni-miskolc.hu/~debreceni/db/dbgyak_er_rel.pdf
4. https://www.inf.u-szeged.hu/~gnemeth/adatbgyak/kidolgozott_peldak.pdf
5. https://szit.hu/doku.php?id=oktatas:adatbazis-kezeles:adatbazis-diagramok
6. https://people.inf.elte.hu/sila/eduAB1/AB1_01A_BevRelMod.pdf
7. https://learn.microsoft.com/hu-hu/office/troubleshoot/access/database-normalization-description
8. http://tehetseg.inf.elte.hu/tananyagok/adatbazis/lecke3_lap1.html
9. https://people.inf.elte.hu/kiss/DB/AB_1_folia.pdf
10. https://www.inf.u-szeged.hu/~gnemeth/adatbgyak/exe/Normalizalas_web/normlformk.html
11. https://www.inf.u-szeged.hu/~gnemeth/adatbgyak/exe/Normalizalas_web/index.html#

Виробничо-практичне видання

Методи проектування баз даних / Adatbázis-tervezési módszerek
Методичні вказівки до самостійної роботи з дісципліни
«Системи керування базами даних»
2025 р.

Затверджено до використання у навчальному процесі
на засіданні кафедри математики та інформатики ЗУІ ім. Ф.Ракоці ІІ
(протокол № 10 від «23» червня 2025 року)

Розглянуто та рекомендовано Навчально-методичною радою
Закарпатського угорського інституту імені Ференца Ракоці ІІ
(протокол № 7 від «26» серпня 2025 року)

Рекомендовано до видання в електронній формі (PDF)
рішенням Вченої ради Закарпатського угорського інституту імені Ференца Ракоці ІІ
(протокол № 8 від «28» серпня 2025 року)

Підготовлено до видання в електронній формі (PDF) кафедрою математики та інформатики спільно з Видавничим відділом Закарпатського угорського інституту імені Ференца Ракоці ІІ (101 с., українською мовою)

Розробник методичних вказівок:

Йожеф ГОЛОВАЧ – професор кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доктор технічних наук

Рецензенти:

Федір ГЕЧЕ – професор УжНУ, доктор техніічих наук, професор

Олександр ТИЛИЩАК – професор кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доктор фізико-математичних наук, доцент

Відповідальні за випуск:

Каталін КУЧІНКА – завідувач кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доцент, кандидат фізико-математичних наук

Олександр ДОБОШ – начальник Видавничого відділу ЗУІ ім. Ф.Ракоці ІІ

Видавництво: Закарпатський угорський інститут імені Ференца Ракоці ІІ (адреса: пл. Кошута 6, м. Берегове, 90202. Електронна пошта: foiskola@kmf.uz.ua) *Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції Серія ДК ____ від _____ 2025 року*

Шрифт «Courier New». Розмір сторінок методичних вказівок: А4 (210х297мм)