МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

з курсу: «Організація та проектування баз даних»

для студентів спеціальності

123 «Комп'ютерна інженерія»

Затверджено
редакційно-видавничою
радою університету,
протокол № 1 від 15.02.2024 р.

Харків
НТУ «ХПІ»

2024

Методичні вказівки до виконання лабораторних робіт з курсу «Організація та проектування баз даних» : для студентів спеціальності 123 «Комп’ютерна інженерія» / уклад.: А. М. Філоненко, Р. В. Чернушенко, В. О. Бречко, О. П. Черних ; Нац. техн. ун-т «Харків. політехн. ін-т». – Харків : НТУ «ХПІ», 2024. – 73 с.

Укладачі: А. М. Філоненко

Р. В. Чернушенко

В. О. Бречко

О. П. Черних

Рецензент В. В. Усик

Кафедра комп’ютерної інженерії та програмування

Вступ

Основні ідеї сучасних інформаційних технологій базуються на концепції, згідно до якої вони повинні бути організовані в бази даних з метою адекватного відображення мінливого реального світу та задоволення інформаційних потреб користувачів. Ці бази даних створюються і функціонують під керуванням спеціальних програмних комплексів, які називають системами управління базами даних (СУБД).

Збільшення об'єму та структурної складності збереження даних, розширення кола користувачів інформаційних систем привели до широкого розповсюдження найбільш зручних і порівняно зручних для розуміння реляційних (табличних) СУБД для забезпечення одночасного доступу к даним безлічі користувачів, нерідко розташованих достатньо далеко один від одного, та від місця зберігання баз даних. Створені мережеві версії баз даних, розраховані на велику кількість користувачів, а також вони основані на реляційних структурах. У них тим або іншим шляхом вирішуються специфічні проблеми паралельних процесів, цільності (правильності) і безпеки даних, а також санкціонування доступу до даних.

Метою даної роботи є викладення методики проектування бази даних для конкретної предметної області, проектування концептуальної схеми бази даних, розробка SQL запитів.

Лабораторна робота 1
ВстановленняMySql Workbench

Введення

http://mithrandir.ru/professional/soft-and-hardware/mysql-workbench-basics.html

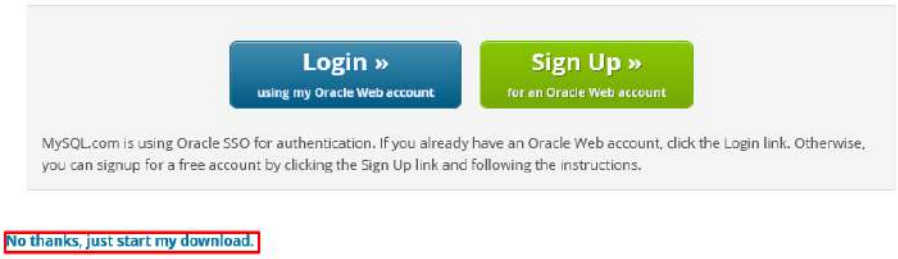
Де завантажити MySql Workbench?

Для того, щоб скачати інсталятор, необхідно перейти на офіційну сторінку: http://dev.mysql.com/downloads/windows/installer/

та натиснути на виділену кнопку"Download" рис. 1.1.

Рисунок 1.1 – Розташування кнопки "Download"

Далі переходимо на сторінку, де пропонують зареєструватися, ігноруємо та натискаємо на"No thanks, just start my download." Мал. 1.2.



Малюнок 1.2 – Пропускаємо реєстрацію

ВстановленняMySql Workbench

Запускаємо установник. Читаємо та погоджуємося з умовами ліцензійної угодиМал. 1.3.

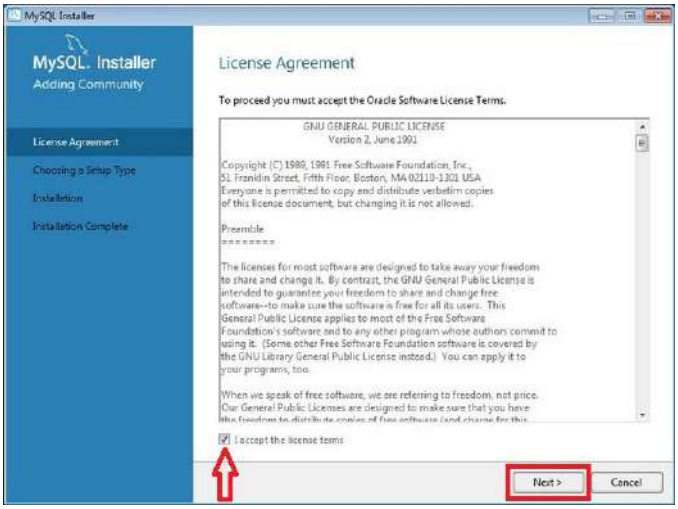


Рисунок 1.3 – Читаємо та погоджуємося з умовами ліцензійної угоди та натискаємо «Next»

Зазначаємо, що хочемо встановити, вибираємо вибір за умовчанням "Developer Default",натискаємо"Next"Мал. 1.4.

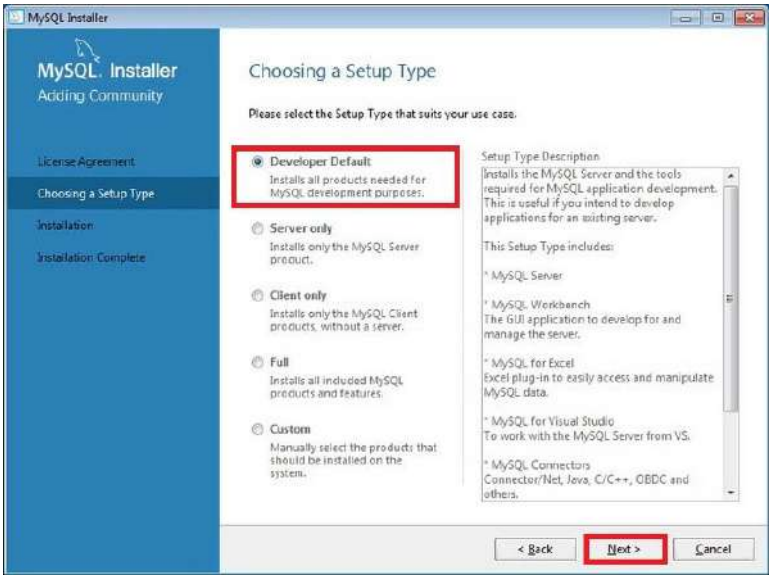


Рисунок 1.4 – Варіанти встановлення

Програма встановлення перевірятиме систему на наявність компонентів. Нічого не вибираємо (якщо є необхідність для роботи MySql Workbench з іншими средами, то ставимо галочку). Для виконання лабораторних робіт це не знадобиться, натискаємо"Next" рис. 1.5.

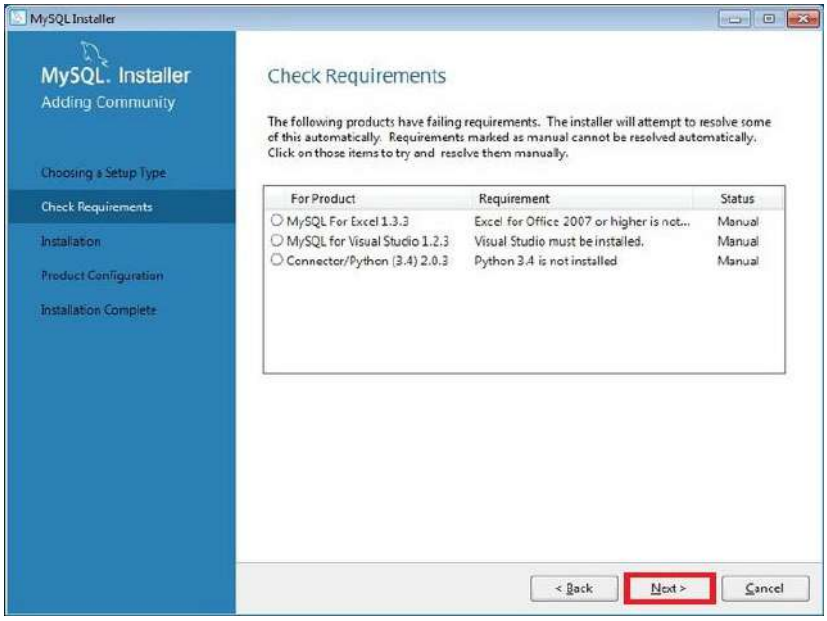


Рисунок 1.5 – Додаткові компоненти для середовища

Далі установник покаже, що саме він буде встановлювати, натискаємо"Execute" рис. 1.6.

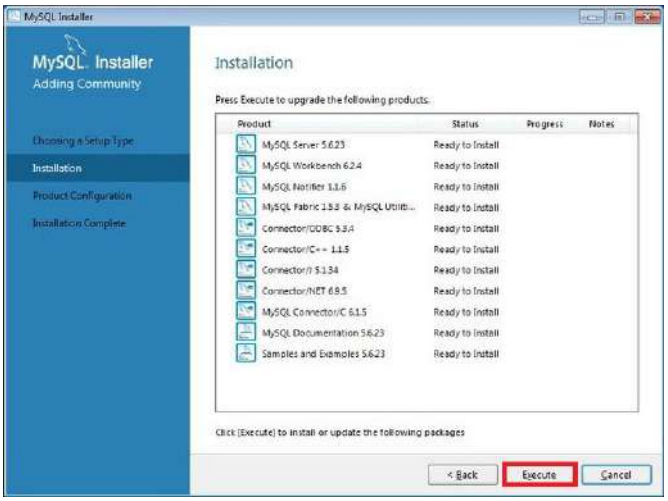


Рисунок 1.6 – Компоненти, які будуть встановлені

Після закінчення завантаження натискаємо кнопку"Next" рис. 1.7.

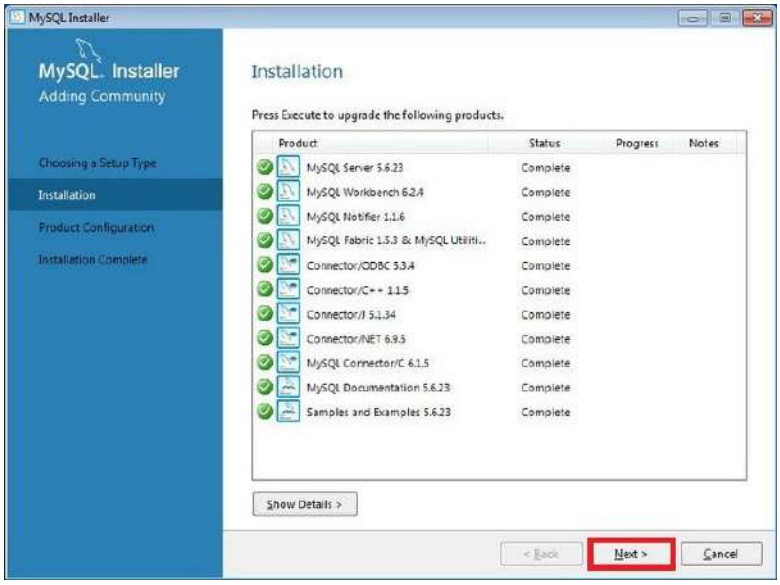


Рисунок 1.7 – Продовження встановлення

Далі установник налаштовує компоненти, натискаємо"Next" рис. 1.8.

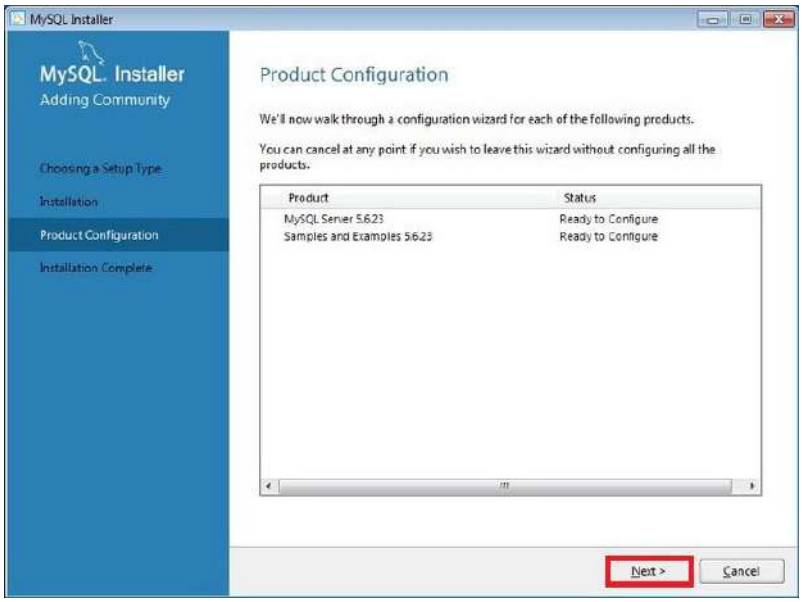


Рисунок 1.8 – Налаштування компонентів

На наступному вікні пропонуємо залишити все за замовчуванням, натискаємо"Next" рис. 1.9.

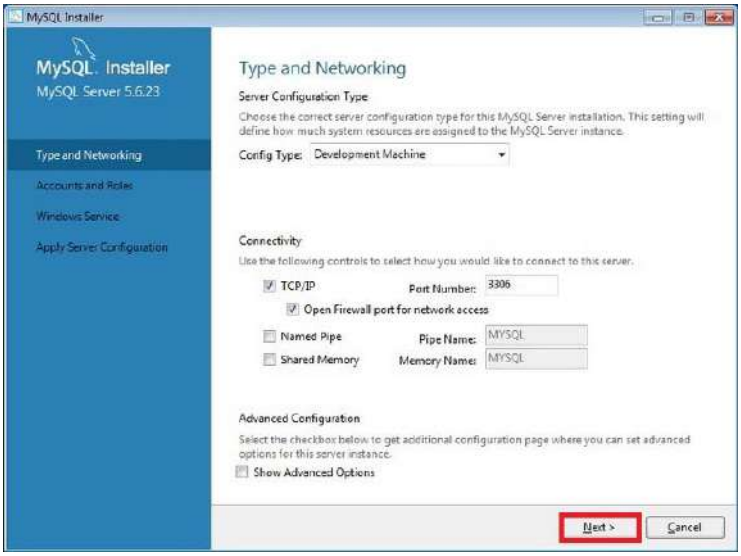


Рисунок 1.9 – Вікно налаштувань мережі

Далі нам потрібно придумати пароль для "root" користувача (нас), для створення користувача тиснемо "Add user" і натискаємо"Next" рис 1.10.

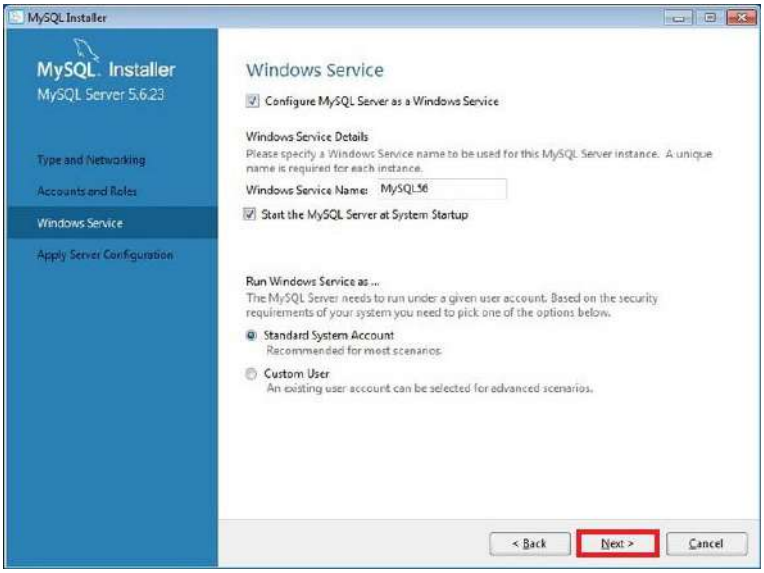


Рисунок 1.10 – Додаткові компоненти для середовища

Застосовуємо всі налаштування, натискаємо "Execute" рис 1.11.

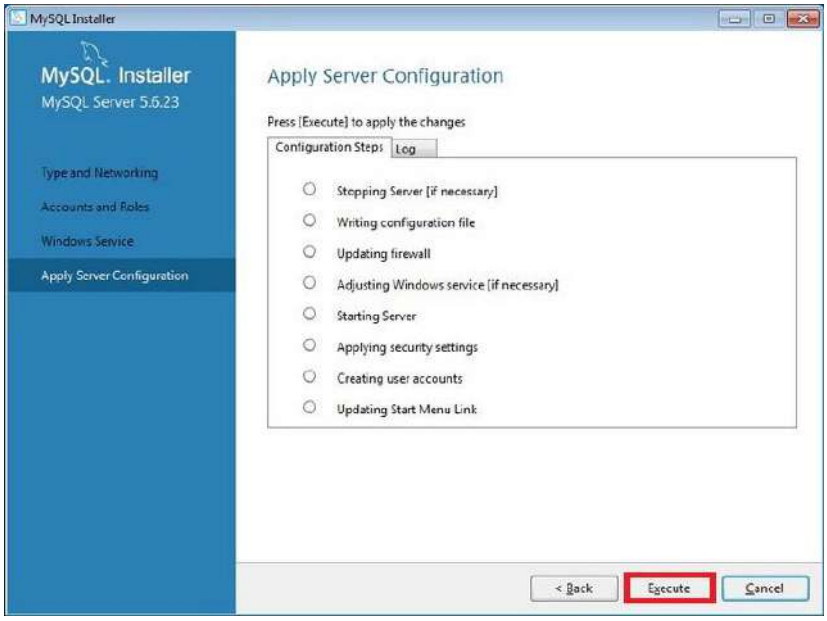


Рисунок 1.11 – Застосування конфігурації сервера

Сервер налаштований, натискаємо «Finish» рис 1.12.

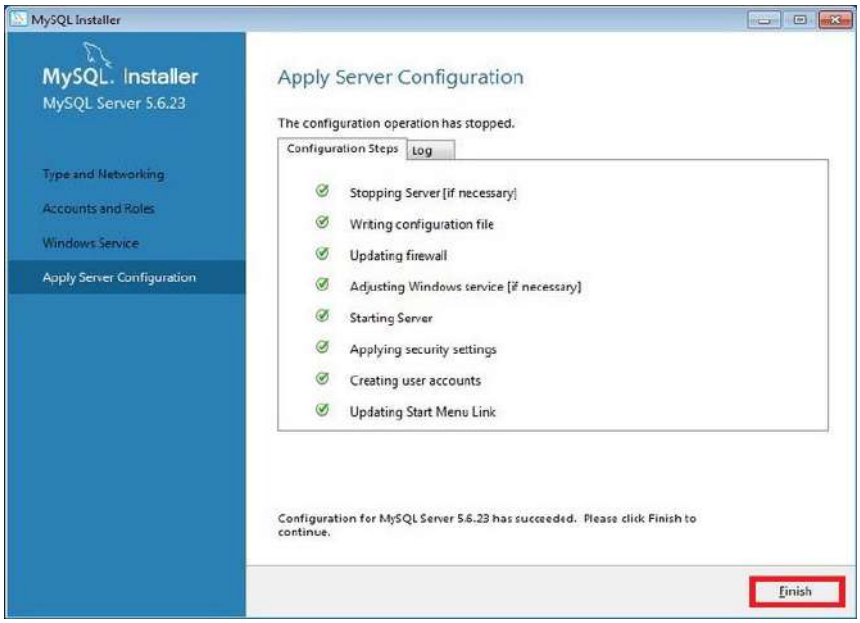


Рисунок 1.12 – Застосування конфігурації сервера

Для налаштування тестових даних сервера MySql натискаємо «Next» рис. 1.13.

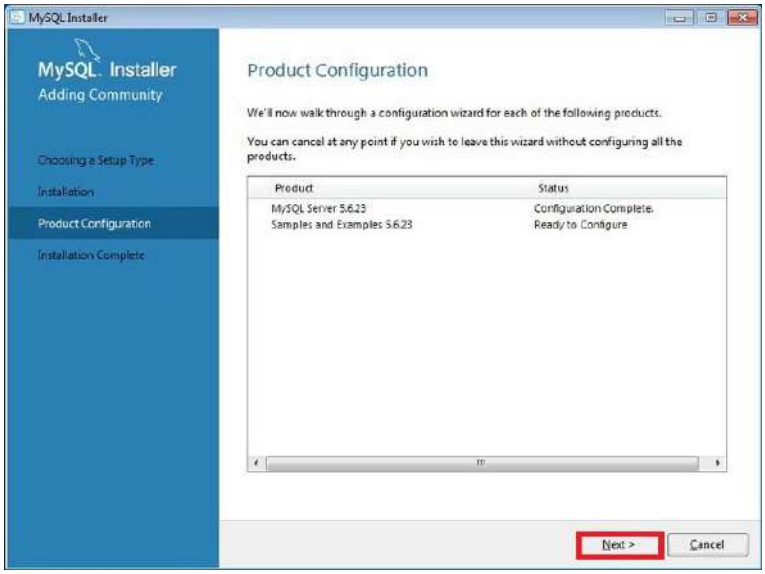


Рисунок 1.13 – Налаштування тестових даних

Для підключення до сервера натискаємо Check, далі Next (запам'ятайте введений пароль) рис 1.14.

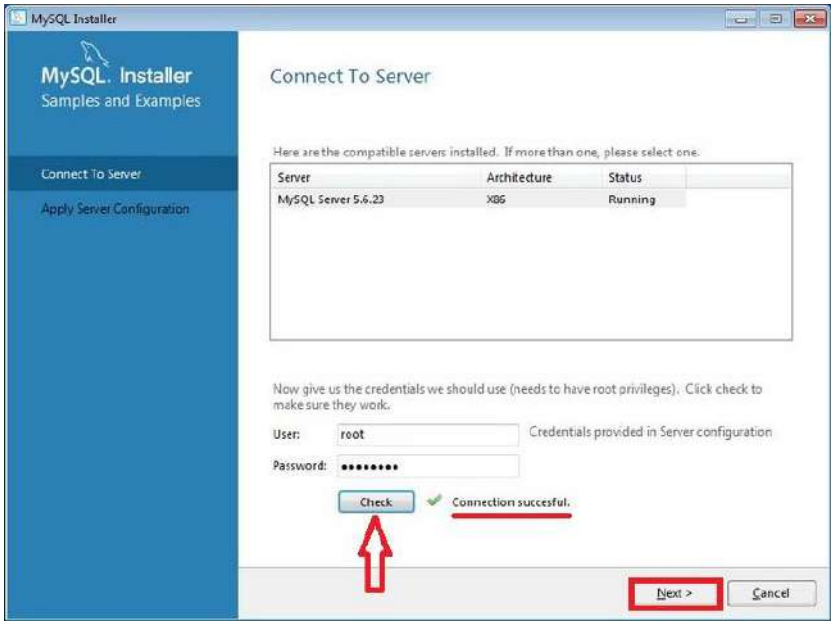


Рисунок 1.14 – Підключення до сервера

Натискаємо "Execute" рис 1.15.

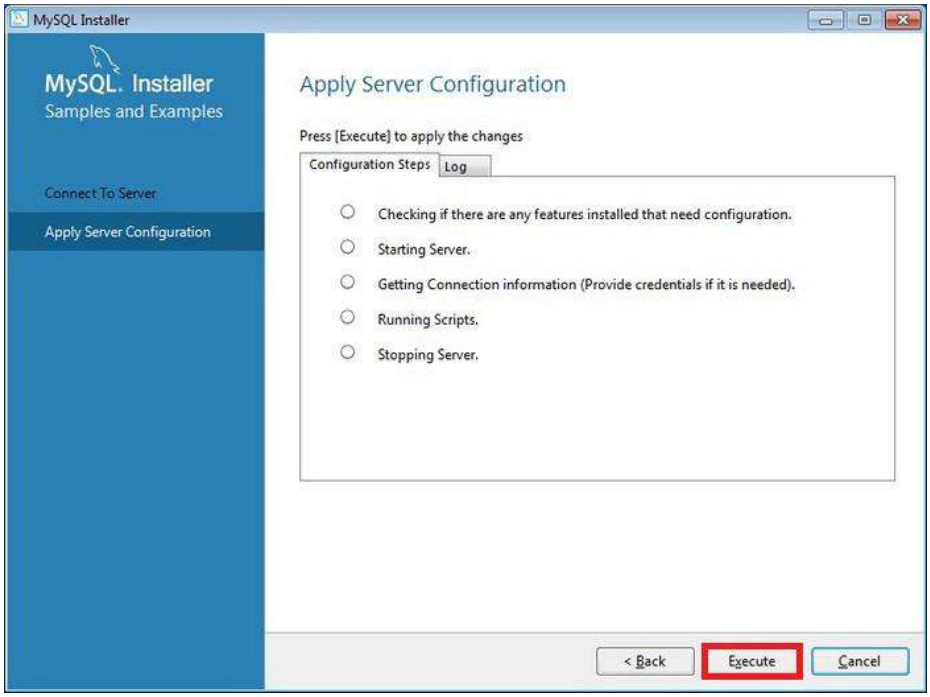


Рисунок 1.15 – Процес підключення до сервера

Натискаємо "Finish" рис. 1.16.

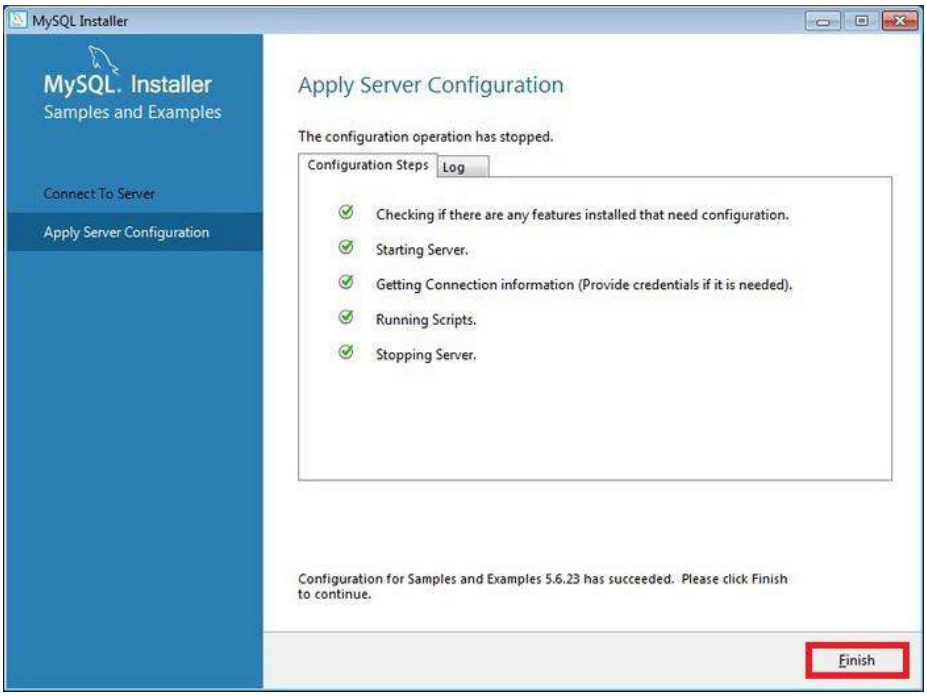
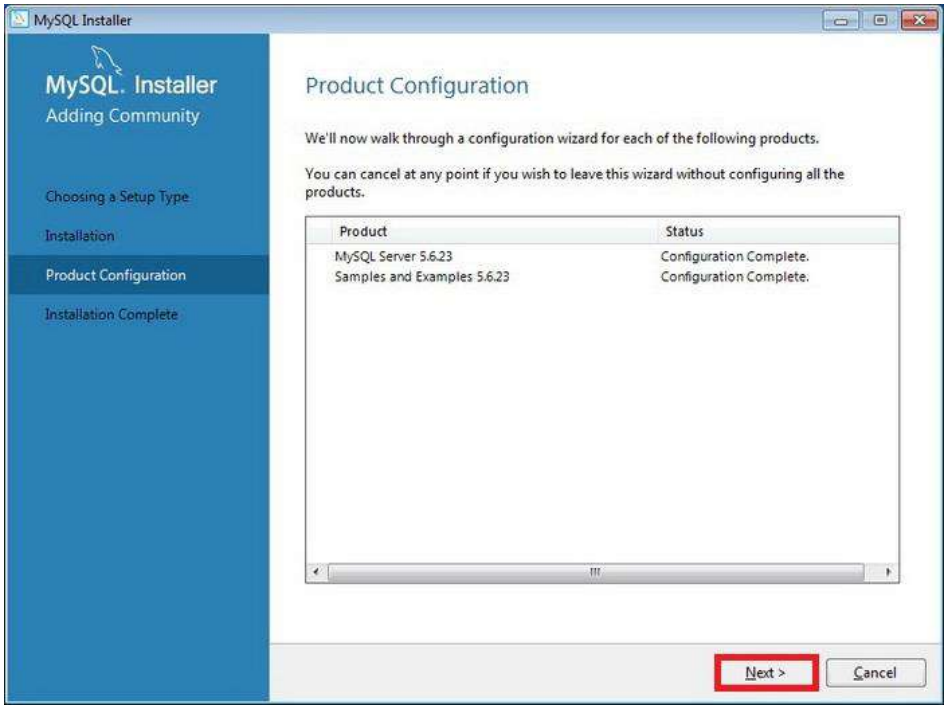


Рисунок 1.16 – Вдале підключення до сервера

Натискаємо "Next" рис 1.17.



Малюнок 1.17 – Конфігурація товару

Натискаємо галочку, для запуску програми відразу після встановлення, натискаємо Finish рис. 1.18.

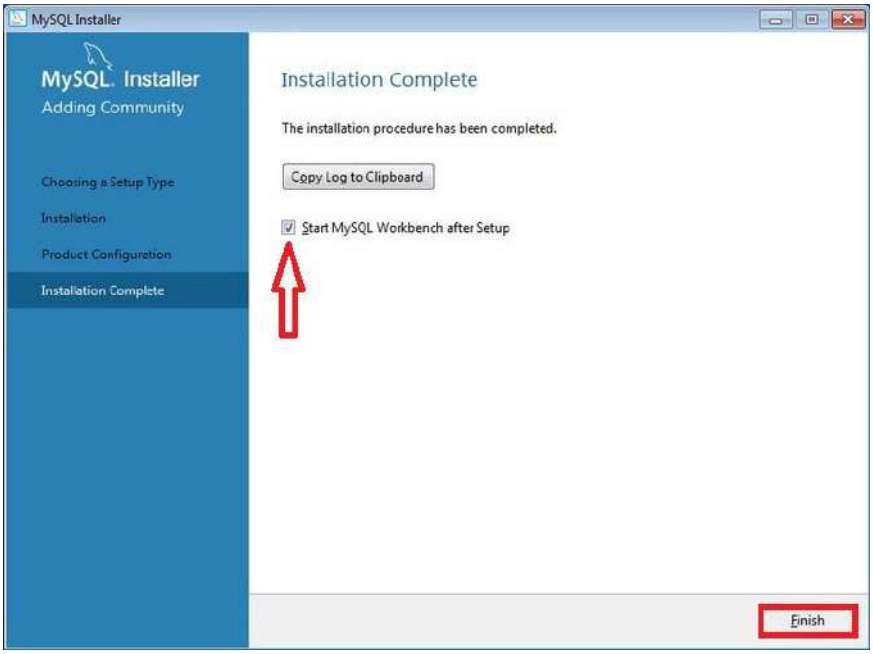


Рисунок 1.18 – Установка завершена

Після запуску MySql Workbench, вибираємо сервер, після встановлення він один, вибираємо його рис 1.19.

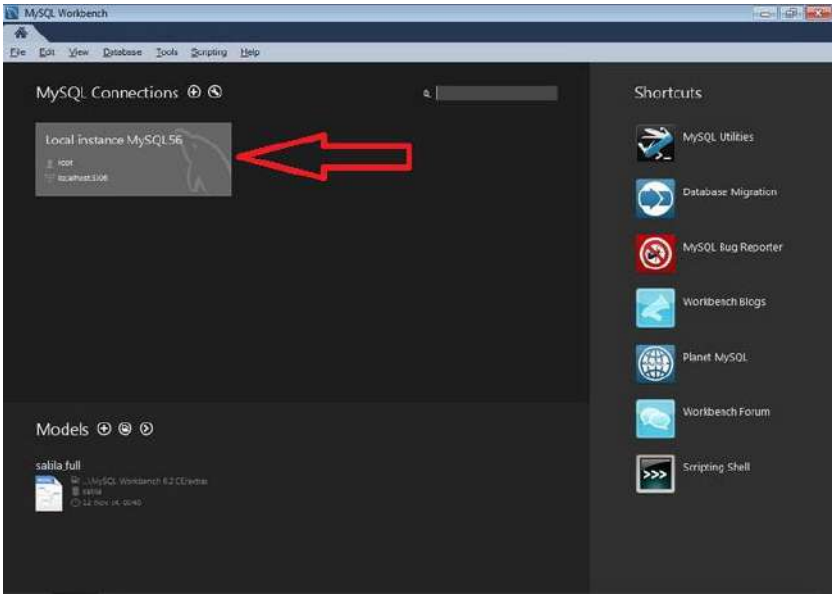


Рисунок 1.19 – Підключення до сервера

Вводимо пароль, який вигадали при налаштуванні сервера рис 1.20.

Малюнок 1.20 – Введення пароля

Розпочинаємо роботу на MySql Workbench.

Лабораторна робота 2
Розробка проекту бази даних

Ціль: отримати навички розробки бази даних

1 Постановка задачі

Проектування бази даних за індивідуальним завданням.

2 Теоретична частина

База даних – це лише набір взаємозалежних даних.

Майже всі сучасні системи базуються на реляційній (relational) моделі управління базами даних. Назва реляційна пов'язана з тим, що кожен запис у

такій базі даних містить інформацію, що стосується лише одного конкретного об'єкта.

У реляційної СУБД усі оброблювані дані представляються як плоских таблиць. Інформація про об'єкти певного виду подається в табличному вигляді: у стовпцях таблиці зосереджені різні атрибути об'єктів, а рядки призначені для описів всіх атрибутів до окремих примірників об'єктів.

Таблиця – це набір даних, з конкретної тематики. Використання окремої таблиці для кожної теми означає, що відповідні дані збережені лише один раз, що робить БД більш ефективним і зменшує кількість помилок при введенні даних.

При розробці таблиць слід враховувати:

- відомості не повинні дублюватися у таблиці або між таблицями;
- кожна таблиця має містити одну тему (ім'я таблиці);
- не рекомендується включати до таблиці дані, що є результатами виразів;
- дані слід розбити на найменші логічні одиниці (поле «Ім'я» розбиваємо на поля «Ім'я» та «Прізвище»).

Що таке SQL?

Крім визначення реляційної моделі Кодд запропонував мову до роботи з даними у реляційних таблицях, названий DSL/Alpha. Незабаром після публікації статті Кодда в IBM було організовано групу створення прототипу мови з урахуванням його ідей. Ця група розробила спрощену версію DSL/Alpha, названу SQUARE. В результаті вдосконалення SQUARE з'явилася мова SEQUEL, яка зрештою отримала ім'я SQL. Зараз SQL розміняв четвертий десяток, зазнавши безліч змін. У середині 1980-х Національний інститут стандартизації США (American National Standards Institute, ANSI) почав розробляти перший стандарт мови SQL, який був опублікований в 1986 р. Подальші доопрацювання були відображені в наступних версіях стандарту SQL (1989, 1992, 1999).). Поряд з удосконаленням базової мови в SQL з'явилися нові можливості для забезпечення об'єктно-орієнтованої функціональності. SQL йде пліч-о-пліч з реляційною моделлю, тому що результатом SQL-запиту є таблиця (в даному контексті також

звана результуючим набором). Таким чином, в реляційній базі даних можна створити постійну нову таблицю, просто зберігши результуючий набір запиту. Аналогічно як вхідні дані запит може використовувати як постійні таблиці, так і результуючі набори інших запитів. І останнє зауваження: SQL не акронім (хоча багато хто наполягає, що це скорочення від Structured Query Language (Структурована мова запитів)). Назва цієї мови вимовляється як термін «Сутність» (entity) – те, що цікавить користувачів бази даних, наприклад, клієнти, запчастини, географічне розташування тощо.

MySQL-вільнареляційна система управління базами даних. Розробку та підтримку MySQL здійснює корпораціяOracle.

Типи даних MySQL

Взагалі кажучи, всі популярні сервери БД мають здатність зберігати одні й самі типи даних, такі як рядки, дати і числа. Зазвичай їхня відмінність полягає у можливості зберігання спеціальних типів даних, наприклад, XML документів, або дуже великих текстів, або двійкових документів. Оскільки ця книга є введенням в SQL і 98% всіх стовпців, які ви коли-небудь зустрінете, будуть простими типами даних, ми розглянемо лише символьні, числові та тимчасові типи даних.

1) Символьні дані

Символьні дані можуть зберігатися як рядки фіксованої або змінної довжини. Різниця полягає в тому, що рядки фіксованої довжини праворуч доповнюються пробілами, тоді як рядки змінної довжини – ні. При визначенні стовпця символьного типу необхідно задати максимальний розмір рядка, що зберігається в ньому. Наприклад, якщо передбачається зберігати рядки довжиною до 20 символів, можна використовувати будь-який з цих описів:

```
CHAR(20)          /* рядок фіксованої довжини */
VARCHAR(20)       /* рядок змінної довжини */
```

В даний час максимальна довжина цього типу даних становить 255 символів (хоча в майбутніх версіях будуть допустимі довші рядки). Для збереження довших рядків (таких як повідомлення електронної пошти, XML документи тощо) використовуйте один з текстових типів – tinytext (крихітний текст), text (текст), mediumtext (середній текст), longtext (довгий текст)), - Розглянутих у даному розділі пізніше. Загалом тип char підходить для випадку, коли в стовпці передбачається зберігати лише рядки однакової довжини, наприклад, скорочені назви держав, а тип varchar – для рядків різної довжини. Типи char і varchar однаково застосовні у всіх основних серверах БД.

2) Текстові дані

Якщо потрібно зберігати дані, для яких не вистачить 255 символів стовпця типу char або varchar, знадобиться один із текстових типів. У табл. 1.1 показані доступні текстові типи та їх максимальні розміри.

Таблиця 1.1 - Текстові типи даних MySQL

Тип	Максимальное число символов
Tinytext	255
Text	65 535
Mediumtext	16 777 215
Longtext	4 294 967 295

Вибираючи той чи інший текстовий тип, потрібно пам'ятати таке:

− якщо розмір даних, що завантажуються в текстовий стовпець, перевищує максимальний розмір для цього типу, дані, що не помістилися, відсікаються;

− на відміну від стовпця типу varchar, при завантаженні даних у такий стовпець прогалини в кінці рядка не видаляються;

− при використанні стовпців типу text для сортування або угруповання використовуються тільки перші 1024 байти, хоча при необхідності це значення, що обмежує, можна збільшити;

– різні текстові типи притаманні виключно MySQL. У SQL Server для великих символьних даних є лише один тип text, а DB2 і Oracle застосовується тип даних під назвою clob (Character Large Object, великий символьний об'єкт).

3) Чисельні дані

MySQL має у своєму розпорядженні кілька різних числових типів для роботи з цими (і багатьма іншими) видами інформації. Найчастіше числові типи використовують із зберігання цілих чисел. При заданні одного з таких типів можна також вказати, що дані беззнакові, тоді сервер буде знати, що всі дані, що зберігаються в стовпці, невід'ємні. У табл. 1.2 показано п'ять різних типів даних, призначених для зберігання цілих чисел.

Таблиця 1.2 - Типи даних MySQL для цілих чисел

Тип	Диапазон значений со знаком	Диапазон значений без знака
Tinyint	от -128 до 127	от 0 до 255
Smallint	от -32 768 до 32 767	от 0 до 65 535
Mediumint	от -8 388 608 до 8 388 607	от 0 до 16 777 215
Int	от -2 147 483 648 до 2 147 483 647	от 0 до 4 294 967 295
Bigint	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	от 0 до 18 446 744 073 709 551 615

При створенні стовпця одного з цілих типів MySQL виділить для зберігання даних відповідну кількість пам'яті - від 1 байта для типу tinyint до 8 байт для bigint. Тому спробуйте підібрати тип достатнього розміру для зберігання найбільшого з передбачуваних чисел без невиправданої витрати пам'яті. Для чисел з плаваючою точкою (таких як 3,1415927) можна вибрати один із типів, наведених у табл. 1.3.

Таблиця 1.3 - Типи даних MySQL для чисел з плаваючою точкою

Тип	Числовой диапазон
Float(*p*,*s*)	от -3,402823466E+38 до -1,175494351E-38 и от 1,175494351E-38 до 3,402823466E+38
Double(*p*,*s*)	от 1,7976931348623157E+308 до -2,2250738585072014E-308 и от 2,2250738585072014E-308 до 1,7976931348623157E+308

4) Тимчасові дані

Тимчасові дані Поряд із рядками та числами досить часто доводиться працювати з інформацією про дати та/або час. Цей тип даних називають тимчасовим (temporal). До прикладів тимчасових даних у базі даних належать:

- дата майбутньої події, наприклад доставки замовлення покупцю;
- фактична дата доставки замовлення покупцю;
- дата та час зміни користувачем певного рядка таблиці;
- дата народження працівника;
- рік, що відповідає рядку таблиці yearly_sales (продажу за рік) у сховищі даних;
- час, необхідний для монтажу електропроводки в автомобілі на складальному конвеєрі MySQL має типи даних для обробки всіх подібних ситуацій. У табл. 1.4 показано тимчасові типи даних, що підтримуються MySQL.

Таблиця 1.4 - Тимчасові типи даних MySQ

Тип	Формат по умолчанию	Допустимые значения
Date	YYYY-MM-DD	от 1000-01-01 до 9999-12-31
Datetime	YYYY-MM-DD HH:MI:SS	от 1000-01-01 00:00:00 до 9999-12-31 23:59:59
Timestamp	YYYY-MM-DD HH:MI:SS	от 1970-01-01 00:00:00 до 2037-12-31 23:59:59
Year	YYYY	от 1901 до 2155
Time	HHH:MI:SS	от -838:59:59 до 838:59:59

Таблиця 1.5 – Компоненти формату дати

Компонент	Описание	Диапазон
YYYY	Год, включая столетие	от 1000 до 9999
MM	Месяц	от 01 (январь) до 12 (декабрь)
DD	День	от 01 до 31
HH	Час	от 01 до 24
HHH	Часы (прошедшие)	от -838 до 838
MI	Минута	от 01 до 60
SS	Секунда	от 01 до 60

3 Практична частина

Перед створенням бази даних, необхідно вирішити яку сутність вона представлятиме, тобто. навіщо вона буде потрібна, яке завдання вирішуватиме?

У нашому прикладі створимо базу даних на тему: «Пошта», яка відповідатиме за стан товару (прийом, відправлення, доставку, отримання).

Визначимося з сутностями, які будуть присутні. У базі даних будуть присутні 2 статичні та 3 динамічні сутності.

Статичні сутності.

Клієнти

1. номер (номер клієнта у таблиці по рахунку);
2. Прізвище;
3. Ім'я;
4. По-батькові;
5. Номер паспорта;
6. Адреса.

Відділення

1. Номер відділення;
2. Адреса;
3. Номер телефону;
4. Тип (пункт прийому/видачі чи розподільчий центр);
5. Обмеження за вагою.

Динамічні сутності.

Журнал оформлення декларації

1. Номер вантажу (номер вантажу у таблиці за рахунком);
2. Номер відправляє клієнта (у таблиці клієнтів);
3. Номер клієнта (у таблиці клієнтів);
4. номер відділення відправки (у таблиці відділення);
5. Номер відділення одержання (у таблиці відділення);
6. Дата відправки;
7. Час відправки.

Журнал переміщень

1. номер вантажу (з журналу оформлення декларації);
2. Номер відділення (до якого прибув вантаж);
3. Час прибуття у відділення.

Журнал отримання

1. номер запису (у таблиці за рахунком);
2. Номер вантажу;
3. Дата отримання;
4. Час отримання.

4 Зміст звіту

1. Мета роботи.
2. Опис предметної сфери.
3. Опис виконання завдання.
4. Роздрук результатів виконання лабораторної роботи.

Лабораторна робота №2
Модифікація проекту бази даних

Мета: виробити навички з модифікації проекту бази даних MySQL Workbench.

1. Постановка задачі

1. Додавання нового поля до таблиці
2. Видалення опису поля
3. Модифікація властивостей даних
4. Зміна первинних ключів бази

2. Теоретична частина

Управління таблицями у базі даних MySQL

Модифікація таблиці

Оператор ALTER TABLE забезпечує можливість змінювати структуру наявної таблиці. Наприклад, можна додавати чи видаляти стовпці, створювати чи знищувати індекси чи перейменовувати стовпці чи саму таблицю. Можна також змінювати коментар для таблиці та її тип.

Якщо оператор ALTER TABLE використовується для зміни визначення типу стовпця, але DESCRIBE tbl_name показує, що стовпець не змінився, можливо, MySQL ігнорує дану модифікацію. Наприклад, при спробі змінити стовпець VARCHAR на CHAR MySQL продовжуватиме використовувати VARCHAR, якщо дана таблиця містить інші стовпці зі змінною довжиною.

Оператор ALTER TABLE створює тимчасову копію вихідної таблиці під час роботи. Необхідна зміна виконується на копії, потім вихідна таблиця видаляється, а нова перейменовується. Так робиться для того, щоб у нову таблицю автоматично потрапляли всі оновлення, крім невдалих. Під час виконання ALTER TABLE вихідна таблиця доступна читання іншими клієнтами. Операції оновлення та запису в цій таблиці призупиняються, доки не буде готова нова таблиця.

Слід зазначити, що при використанні будь-якої іншої опції для ALTER TABLE, крім RENAME, MySQL завжди буде створювати тимчасову таблицю, навіть якщо дані, строго кажучи, і не потребують копіювання (наприклад, зміни

імені стовпця). Для використання оператора ALTER TABLE необхідні привілеї ALTER, INSERT та CREATE для цієї таблиці.

- ОпціяIGNOREє розширенням MySQL стосовно ANSI SQL92. Вона керує роботоюALTER TABLEза наявності дублікатів унікальних ключів у новій таблиці. Якщо опціяIGNOREне задана, то цієї копії процес переривається і відбувається відкат назад. ЯкщоIGNOREвказується, тоді для рядків із дублікатами унікальних ключів тільки перший рядок використовується, а решта видаляється.
- Можна, можливо запустити кілька виразівADD,ALTER,DROPiCHANGEв одній командіALTER TABLE. Це є розширенням MySQL по відношенню до ANSI SQL92, де допускається лише один вираз зі згаданих в одній командіALTER TABLE.
- ОпціїCHANGE col_name,DROP col_nameiDROP INDEXтакож є розширеннями MySQL щодо ANSI SQL92.
- ОпціяMODIFYявляє собою розширення Oracle для командиALTER TABLE.
- Необов'язкове словоCOLUMNє "білий шум" і може бути опущено.
- При використанніALTER TABLE ім'я_таблиці RENAME TO нове_ім'ябез будь-яких інших опцій MySQL просто перейменовує файли, що відповідають заданій таблиці. І тут немає потреби створювати тимчасову таблицю. У виразіcreate_definitionдляADDiCHANGEвикористовується той же синтаксис, що і дляCREATE TABLE. Слід враховувати, що це синтаксис включає ім'я стовпця, а чи не просто його тип.
- Стовпець можна перейменовувати, використовуючи виразCHANGE имя_столбца create_definition. Щоб зробити це, необхідно вказати старе та нове імена стовпця та його тип в даний час. Наприклад, щоб перейменувати стовпецьINTEGERзаab, можна зробити таке:
- mysql> ALTER TABLE t1 CHANGE ab INTEGER;

При зміні типу стовпця, але не його імені синтаксис виразуCHANGEоднаково вимагає вказівки обох імен стовпця, навіть якщо вони однакові. Наприклад:

mysql> ALTER TABLE t1 CHANGE bb BIGINT NOT NULL;

Однак, починаючи з версії MySQL 3.22.16a, можна також використовувати виразMODIFYдля зміни типу стовпця без перейменування його:

mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;

- При використанніCHANGEабоMODIFYдля того, щоб зменшити довжину стовпця, в частині якого побудовано індекс (наприклад, індекс за першими 10 символами стовпцяVARCHAR), не можна зробити стовпець коротшим, ніж число проіндексованих символів.
- При зміні типу стовпця з використаннямCHANGEабоMODIFYMySQL намагається перетворити дані на новий тип якомога коректніше.
- У версії MySQL 3.22 і пізніших можна використовуватиFIRSTабоADD ... AFTER ім'я_стовпцядля додавання шпальти на задану позицію всередині табличного рядка. За замовчуванням стовпець додається до кінця. Починаючи з версії MySQL 4.0.1 можна також використовувати ключові словаFIRSTiAFTERв опціяхCHANGEабоMODIFY.
- ОпціяALTER COLUMNзадає для стовпця нове значення за промовчанням або видаляє старе. Якщо старе значення за замовчуванням видаляється і цей стовпець може набувати значенняNULL, то нове значення за замовчуванням будеNULL.
- ОпціяDROP INDEXвидаляє індекс. Це розширення MySQL по відношенню до ANSI SQL92. Якщо стовпці видаляються з таблиці, то ці стовпці видаляються також і з будь-якого індексу, який вони входять як частина. Якщо всі стовпці, що становлять індекс, видаляються, то цей індекс також видаляється.

- Якщо таблиця містить лише один стовпець, цей стовпець не може бути видалений. Натомість можна видалити цю таблицю, використовуючи командуDROP TABLE.
- ОпціяDROP PRIMARY KEYвидаляє первинний індекс. Якщо такого індексу в даній таблиці не існує, видаляється перший індексUNIQUEу цій таблиці. (MySQL відзначає перший унікальний ключUNIQUEяк первинний ключPRIMARY KEYякщо інший первинний ключPRIMARY KEYне було явно вказано). При додаванніUNIQUE INDEXабоPRIMARY KEYв таблицю вони зберігаються перед рештою неунікальних ключів, щоб можна було визначити дублюються ключі якомога раніше.
- ОпціяORDER BYдозволяє створювати нову таблицю з рядками, які розміщені в заданому порядку. Слід враховувати, що створена таблиця не зберігатиме цей порядок рядків після операцій вставки та видалення. У деяких випадках така можливість може полегшити операцію сортування MySQL, якщо таблиця має таке розташування стовпців, яке ви хотіли б мати надалі. Ця опція в основному корисна, якщо заздалегідь відомий певний порядок, в якому будуть запитуватися рядки. Використання цієї опції після значних перетворень таблиці дає можливість отримати більш високу продуктивність.
- При використанні командиALTER TABLEдля таблиць MyISAMвсі неунікальні індекси створюються в окремому пакеті (подібно доREPAIR). Завдяки цьому командаALTER TABLEза наявності кількох індексів працюватиме швидше.
- Починаючи з MySQL 4.0, вищезгадана можливість може бути активізована явно. КомандаALTER TABLE ... DISABLE KEYSблокує у MySQL оновлення неунікальних індексів для таблиць MyISAM. Після цього можна застосувати командуALTER TABLE ... ENABLE KEYSдля відтворення недостатніх індексів. Так як MySQL робить це за допомогою спеціального алгоритму, який набагато швидший у порівнянні зі вставкою ключів один

за одним, блокування ключів може дати істотне прискорення на великих масивах вставок.

- Застосовуючи функцію C APImysql_info(), можна визначити, скільки записів було скопійовано, а також (при використанніIGNORE) - скільки записів було видалено через дублювання значень унікальних ключів.
- ВиразиFOREIGN KEY,CHECKiREFERENCESпрактично нічого не роблять. Вони введені лише з міркувань сумісності, щоб полегшити перенесення коду з інших серверів SQL та запуск програм, які створюють таблиці з посиланнями. Нижче наведено приклади, що показують деякі випадки вживання командиALTER TABLE. Приклад починається з таблиціt1, яка створюється так:

mysql> CREATE TABLE t1 (a INTEGER, b CHAR (10));

Щоб перейменувати таблицю з t1 в t2:

mysql> ALTER TABLE t1 RENAME t2;

Щоб змінити тип стовпця з INTEGER на TINYINT NOT NULL (залишаючи ім'я колишнім) і змінити тип стовпця b cCHAR(10) на CHAR(20) з перейменуванням його з b на c:

mysql> ALTER TABLE t2 MODIFY TINYINT NOT NULL, CHANGE bc CHAR(20);

Щоб додати новий стовпець TIMESTAMP з ім'ям d:

mysql> ALTER TABLE t2 ADD d TIMESTAMP;

Для того щоб додати індекс до стовпця d і зробити стовпець a первинним ключем:

mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);

Для того, щоб видалити стовпець з:

mysql> ALTER TABLE t2 DROP COLUMN c;

Для того, щоб додати новий числовий стовпець AUTO_INCREMENT з ім'ям c:

mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,

ADD INDEX (c);

Зауважте, що стовпець с індексується, оскільки стовпці AUTO_INCREMENT мають бути індексовані, крім того, стовпецьс оголошується як NOT NULL, оскільки індексовані стовпці не можуть бути NULL.

При додаванні стовпця AUTO_INCREMENT значення цього стовпця автоматично заповнюються послідовними номерами (додавання записів). Перший номер послідовності можна встановити шляхом виконання команди SET INSERT_ID = # перед ALTER TABLE або використання табличної опції AUTO_INCREMENT = #. See section5.5.6 Синтаксис команди SET.

Якщо стовпець AUTO_INCREMENT для таблиць MyISAM не змінюється, то номер послідовності залишається незмінним. При видаленні стовпця AUTO_INCREMENT і додаванні іншого стовпця AUTO_INCREMENT номери будуть починатися знову з 1.

Індексування

Одне з основних завдань, що виникають під час роботи з базами даних, – це завдання пошуку. У цьому, оскільки у базі даних, зазвичай, міститься багато, перед програмістами постає завдання непросто пошуку, а ефективного пошуку, тобто. пошуку за порівняно невеликий час та з достатньою точністю. Для цього (для оптимізації продуктивності запитів) здійснюють індексування деяких полів таблиці. Використовувати індекси корисно для швидкого пошуку рядків із зазначеним значенням одного стовпця. Без індексу читання таблиці здійснюється по всій таблиці, починаючи з першого запису, доки знайдено відповідні рядки. Чим більша таблиця, тим більші накладні витрати. Якщо ж таблиця містить індекс по стовпцям, то база даних може швидко визначити позицію для пошуку в середині файлу даних без перегляду всіх даних. Це відбувається тому, що база даних містить проіндексовані поля ближче в пам'яті, так, щоб можна було швидше знайти їх значення. Для таблиці, що містить 1000 рядків, це буде щонайменше у 100 разів швидше порівняно з послідовним перебором усіх записів. Однак у випадку, коли необхідний доступ майже до всіх 1000 рядків, швидше буде послідовне читання, оскільки при цьому не потрібні операції пошуку по диску. Тож іноді індекси бувають лише на заваді.

Наприклад, якщо копіюється великий обсяг даних у таблицю, краще не мати ніяких індексів. Однак у деяких випадках потрібно задіяти відразу кілька індексів (наприклад, для обробки запитів до таблиць, що часто використовуються).

Якщо говорити про Mysql, то існує три види індексів: PRIMARY, UNIQUE, та INDEX, а слово ключ (KEY) використовується як синонім слова індекс (INDEX). Усі індекси зберігаються у пам'яті як B-дерев.

PRIMARY – унікальний індекс (ключ) з обмеженням, що це індексовані їм поля що неспроможні мати порожнього значення (тобто. вони NOT NULL). Таблиця може мати лише один первинний індекс, але може складатися з кількох полів.

UNIQUE – ключ (індекс), що задає поля, які можуть лише унікальні значення.

INDEX - звичайний індекс (як ми описали вище). У Mysql, крім того, можна індексувати рядкові поля за заданим числом символів від початку рядка.

Первинний ключ та зв'язки

Кожен рядок даних у таблиці ідентифікується унікальним "ключом", який називається первинним ключем. Часто первинний ключ це автоматично збільшується (автоінкрементне) число (1,2,3,4 і т.д). Дані різних таблицях можуть бути пов'язані разом під час використання ключів. Значення первинного ключа однієї таблиці можуть бути додані до рядка (запису) іншої таблиці, тим самим, пов'язуючи ці записи разом.

Первинний ключ використовується для ідентифікації записів у таблиці, для того, щоб кожен запис став унікальним.

Декілька прикладів:

- Номер замовлення, який ви отримуєте при покупці в інтернет-магазині, може бути первинним ключем якоїсь таблиці замовлень у базі даних цього магазину, т.к. він є унікальним значенням.

– Номер соціального страхування то, можливо первинним ключем у який-небудь таблиці у основі даних державної установи, т.к. як і в попередньому прикладі унікальний.

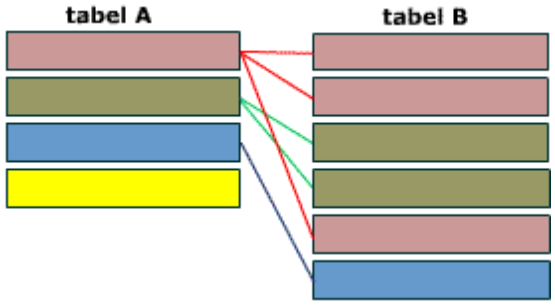
– Номер рахунку-фактури може бути використаний як первинний ключ у таблиці бази даних, в якій зберігаються видані клієнтам рахунки-фактури.

Первинний ключ має унікальне значення. Це означає, що будь-яке значення первинного ключа може зустрітися в стовпці, який обраний як первинний ключ, лише один раз. РСУБД влаштовані так, що не дозволять вам вставити дублікати у поле первинного ключа, отримайте помилку.

Другий крок у проектуванні баз даних - це вибір того, які зв'язки існують між сутностями у вашій системі.

Зв'язок одним-багатьом

Коли один запис у таблиці A може бути пов'язана з 0, 1 або безліччю записів у таблиці B, тип зв'язку один-багатьом. У реляційній моделі даних зв'язок одним-багатьом використовує дві таблиці.



Малюнок 2.1 - Схематичне уявлення зв'язку одним-багатьом

Деякі приклади зв'язку одним-багатьом:

– Машина та її частини. Кожна частина машини одночасно належить лише одній машині, але машина може мати безліч частин.

– Кінотеатри та екрани. В одному кінотеатрі може бути безліч екранів, але кожен екран належить лише одному кінотеатру.

– Будинки та вулиці. На вулиці може бути кілька будинків, але кожен будинок належить лише одній вулиці.

Створення схеми даних

Описати словами та показати на прикладі

Зв'язок багато-багатьом

Зв'язок багато-багатьом – це зв'язок, при якому множинні записи з однієї таблиці (A) можуть відповідати множинні записи з іншої (B). Прикладом такого зв'язку може бути школа, де вчителі навчають учнів. У більшості шкіл кожен учитель навчає багатьох учнів, а кожен учень може навчатися кількома учителями.

Створення зв'язку багато-багатьом.

Зв'язок багато-багатьом створюється за допомогою трьох таблиць.

Дві таблиці - "джерела" та одна сполучна таблиця. Первинний ключ сполучної таблиці A_B – складової. Вона складається із двох полів, двох зовнішніх ключів, які посилаються на первинні ключі таблиць A та B.

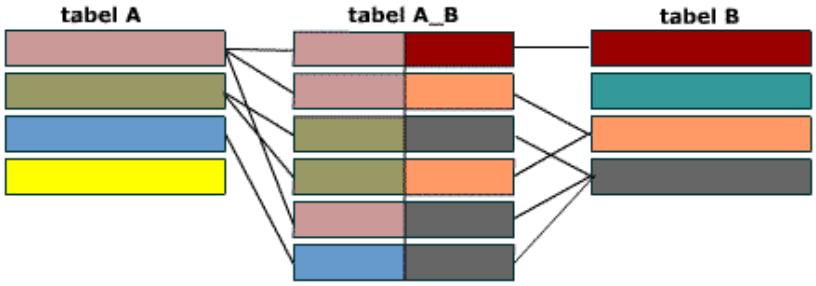


Рисунок 2.2 – Приклад зв'язку багато хто – до – багатьох

Усі первинні ключі мають бути унікальними. Це має на увазі і те, що комбінація полів A і B має бути унікальною в таблиці A_B. Зв'язок багато-багатьом складається з двох зв'язків одним-багатьом.

Приклад зв'язку багатьом: замовлення квитків в готелі (Малюнок 1.3).

У цьому прикладі видно, що між таблицями гостей і кімнат існує зв'язок багатьом. Одна кімната може бути замовлена багатьма гостями з часом і гість може замовляти багато кімнат в готелі. Сполучна таблиця у разі є класичної сполучної таблицею, що складається з двох зовнішніх ключів. Вона є окремою сутністю, яка має зв'язки із двома іншими сутностями.

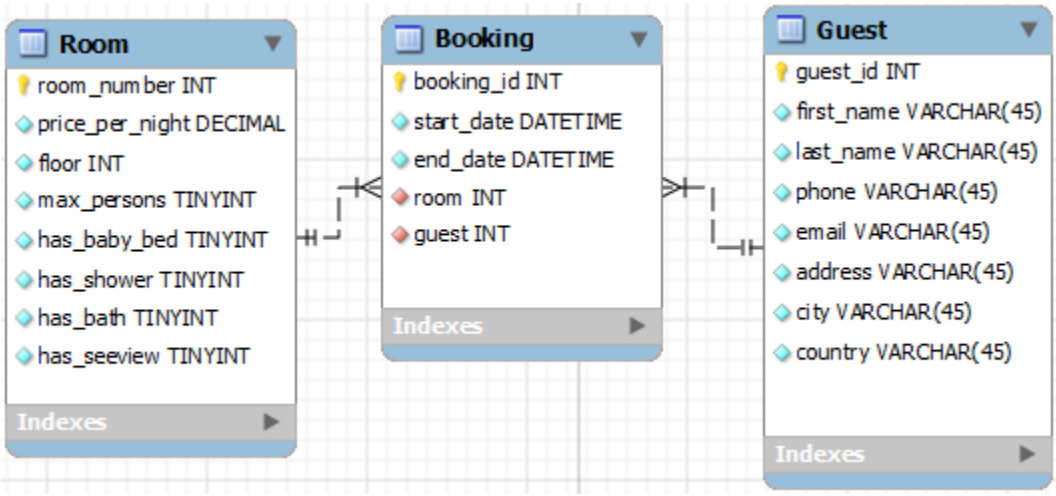


Рисунок 2.3 – Приклад зв'язку багатьом: замовлення квитків в готелі

Зв'язок один до одного

У зв'язку один до одного кожен блок сутності A може бути асоційований з 0, 1 блоком сутності B. Найманий працівник, наприклад, зазвичай пов'язаний з одним офісом.

Приклад зв'язку один до одного: люди та їх паспорти. Кожна людина в країні має тільки один паспорт, що діє, і кожен паспорт належить тільки одній людині.

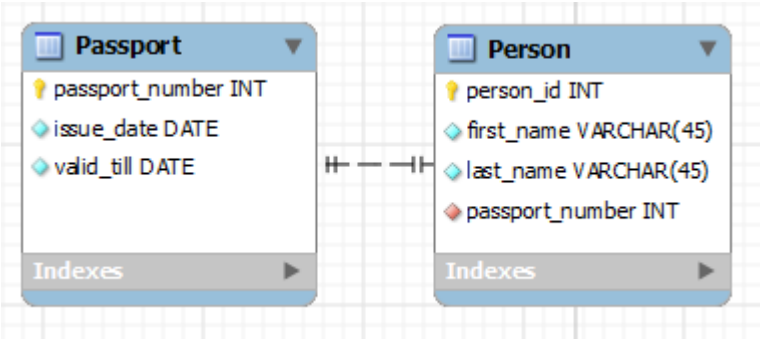


Рисунок 2.4 – Приклад зв'язку один до одного

З видів модифікації бази даних іноді може знадобитися розщеплення однієї таблиці на кілька, або об'єднання групи таблиць в одну.

Розщеплення таблиць

Розщеплення таблиць може знадобитися в тому випадку, якщо одна з таблиць проекту містить групу полів, що рідко використовується. Наприклад, у запитах використовуються основні ознаки об'єкта, а докладніша інформація буває затребувана рідко в підсумках, зведеннях.

Має сенс розщепити таку таблицю на дві (або більше) частини, зв'язавши їх за якоюсь ознакою об'єкта. Також підвищує продуктивність системи розщеплення таблиці, якщо частини даних потрібно обмежити доступ користувачів. До конфіденційних даних можна віднести інформацію про адреси замовників, зарплату співробітників, деталі контрактів тощо. У цьому випадку закрита інформація відокремлюється в окремі таблиці і доступ до них користувачів обмежується.

Об'єднання таблиць

Працюючи з базою даних може виникнути необхідність об'єднання кількох таблиць.

Об'єднання таблиць підвищує продуктивність системи, коли:

− більшість звернень до даних проводиться у запитах у дві чи більше таблиць, але за окремістю вони використовуються рідко;

− Дві (або більше) таблиці описують один об'єкт, причому одна з них містить великий обсяг даних, і є група запитів, коли ці таблиці використовуються разом.

Об'єднання таких таблиць може зменшити час доступу до даних та збільшити швидкість їх обробки. Великі бази даних підтримують засіб тестування продуктивності системи після внесення змін до проекту.

3. Практична частина

Створення таблиці та додавання нового поля

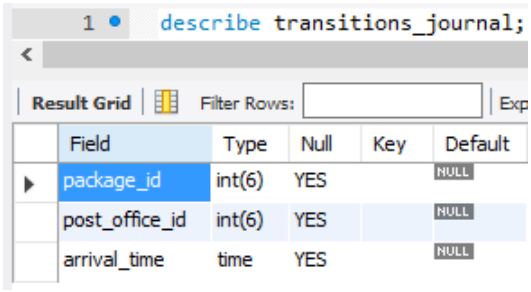
Створимо динамічну таблицю - журнал переміщень "clients", виконавши запит:

```
CREATE TABLE transitions_journal (package_id INT(6),
post_office_id INT(6));
```

У цій таблиці не вистачає 1 поля – «Час прибуття у відділення», тому ми зараз додамо його запитом:

ALTER TABLE transitions_journal ADD arrival_time TIME;

Структура таблиці після додавання поля див. мал. 2.5



Малюнок 2.5 – Структура таблиці transitions_journal

Цей запит можна модифікувати, додавши вказівку, після якого поля додавати нове поле, за замовчуванням воно додається в кінець, але можна було б додати його після першого поля «package_id» таким чином:

ALTER TABLE transitions_journal ADD arrival_time TIME AFTER package_id;

Тоді б у структурі таблиця другою місці стояло час прибуття у відділення («arrival_time»), але в третьому номер відділення («post_office_id»).

Додавання даних до таблиці

Тепер додамо до таблиці «clients» дані:

INSERT INTO clients VALUES ('1', 'Egor', 'Avilov', 'Olegovich', '0951234567', 'St. Petrova 3-2-123');

Адреса у форматі: вулиця-будинок-корпус-квартира.

Так само додамо ще 3-х клієнтів, і таблиця набуде такого вигляду (рис. 2.1):

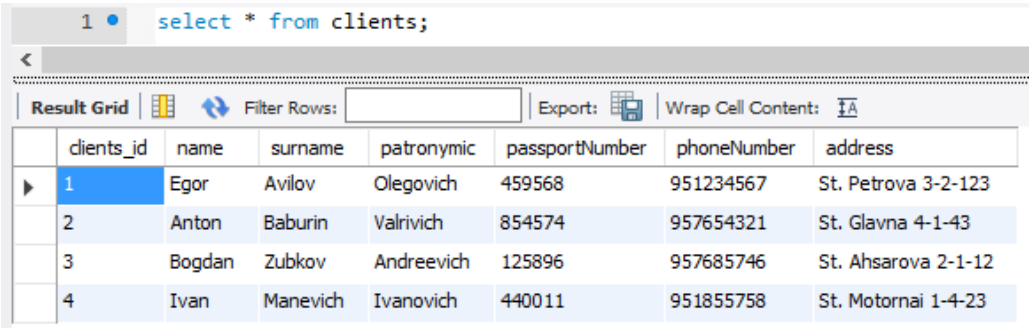


Рисунок 2.6 – Виведення таблиці

Видалення значення поля з таблиці

Щоб видалити значення поля з таблиці, потрібно скористатися запитом:

```
DELETE FROM clients WHERE clients_id = 4;
```

Ми видалили останній запис.

Якщо WorkBench видасть помилку 1175 «про неможливість видалення», використовуйте команду, щоб виправити її:

```
SET SQL_SAFE_UPDATES = 0;
```

Зміна типу поля

Щоб змінити тип поля в таблиці на інший, потрібно скористатися командою:

```
ALTER TABLE clients MODIFY phoneNumber INT;
```

Ми змінили тип поля "phoneNumber" з BIGINT на INT, тому що мобільний номер міститься і в нього.

Зміна первинних ключів бази даних

Відкриваємо нашу базу даних (рис. 2.7) і натискаємо на значок гайкового ключа (рис. 2.8), вибираючи редагування таблці.

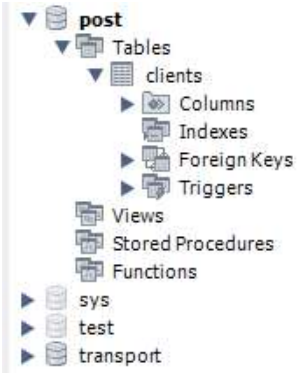


Рисунок 2.7 – Відкриття БД

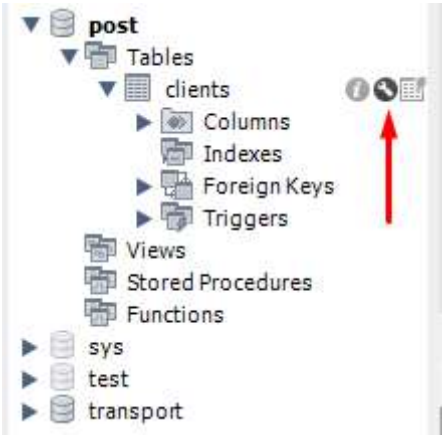


Рисунок 2.8 – Вибираємо редагування таблиці

Після чого у вас відкриється приблизно таке вікно (рис. 2.9)

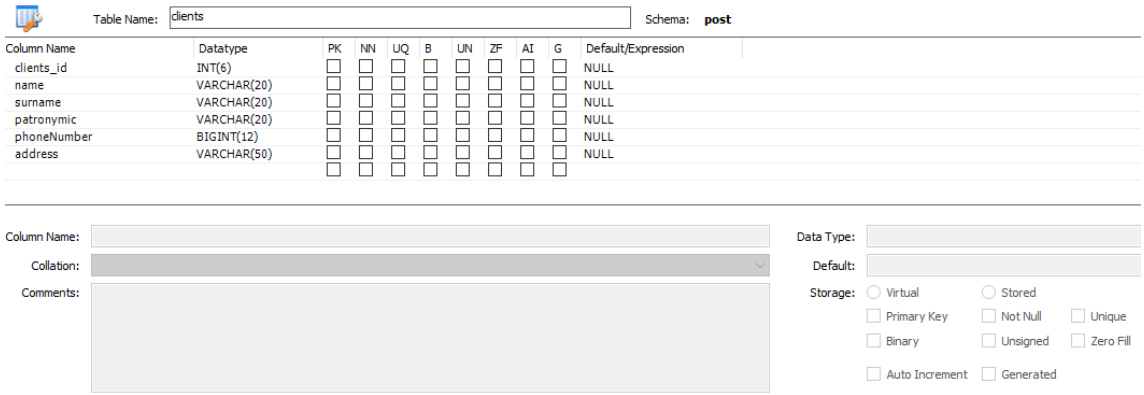
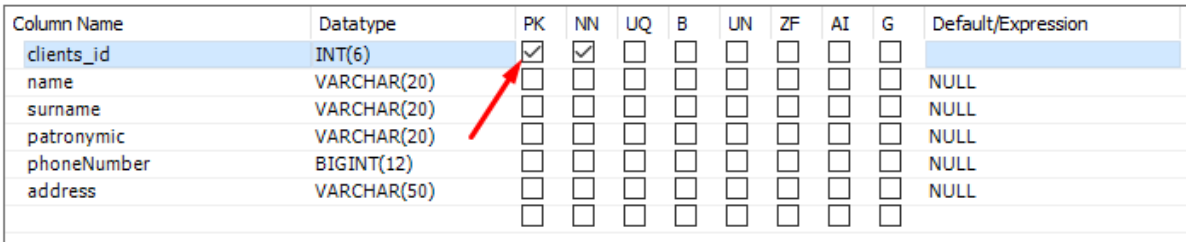


Рисунок 2.9 – Меню редагування таблиці

І ставимо галочку на PK (див. рис. 2.10) (галочка на NN встановиться автоматично).



Малюнок 2.10 – Створення первинного ключа

Створили ключ зовнішньої таблиці.

4 Зміст звіту:

1. Мета роботи.

2. Опис предметної сфери.

3. Короткий опис виконаного завдання та використаних засобів.

4. *SQL*запити, що використовувалися.

5. Результати виконання створених запитів.

6. Висновки про можливостіпрограмного пакета Workbench та мови запитів SQL для вирішення поставленого завдання.

Лабораторна робота №3
Експорт, імпорт та з'єднання даних у MySQL

Мета: отримати навички експортування та імпортування даних у MySQL Workbench.

1. Постановка задачі

1. Імпорт даних
2. Імпорт таблиць баз даних
3. Імпорт даних із електронної таблиці Excel
4. Імпорт текстових файлів
5. Експорт даних
6. Приєднання зовнішніх даних
7. Приєднання таблиці Результати з БД філії

2. Теоретична частина

Експорт та імпорт даних MySQL зазвичай потрібно при перенесенні інформації з однієї бази даних MySQL в іншу і для здійснення резервного копіювання. Резервне копіювання даних має суто технологічний характер. Це означає, що у разі будь-якого програмного чи апаратного збою обладнання, буде можливість відновити актуальні дані.

Копіювання файлів бази:

Базу даних MySQL можна скопіювати, якщо тимчасово вимкнути MySQL-сервер і просто скопіювати файли з папки/var/lib/mysql/db/. Якщо сервер не вимкнути, з очевидних причин можлива втрата та псування даних. Для великих навантажених баз ця можливість близька до 100%. Крім того, при першому запуску з «брудною» копією бази даних MySQL сервер почне процес перевірки всієї бази, який може затягнутися на годинник.

У більшості «живих» проектів регулярне вимкнення сервера БД тривалий час неприйнятно. Для вирішення цієї проблеми застосовується трюк, що базується на снепшотах файлової системи. Снепшот — це щось подібне до «фотографії» файлової системи на певний момент часу, зроблений без реального копіювання даних (і тому швидко). Аналогічно працює «ледаче копіювання» об'єктів у багатьох сучасних мовах програмування.

Загальна схема дій така: блокуються всі таблиці, скидається файловий кеш БД, робиться сніпшот файлової системи, розблоковуються таблиці. Після цього файли спокійно копіюються зі снепшота, після чого він знищується. "Блокуюча" частина такого процесу займає час порядку секунд, що вже терпимо. Як розрахунок на якийсь час, поки «живий» снепшот, знижується продуктивність файлових операцій, що в першу чергу б'є за швидкістю операцій запису в базу.

Копіювання файлів — найшвидший спосіб перекинути базу даних з одного сервера на інший.

Копіювання через текстові файли:

Щоб рахувати в бэкап дані з production-бази, необов'язково смикати файли. Можна вибрати дані запитом та зберегти їх у текстовий файл. Для цього використовується SQL-командаSELECT INTO OUTFILEі парна їйLOAD DATA INFILE. Вивантаження проводиться рядково (можна відібрати для збереження лише потрібні рядки, як у звичайному SELECT). Структура таблиць ніде не вказується — це має піклуватися програміст. Він також повинен подбати про включення команд SELECT INTO OUTFILE у транзакцію, якщо це необхідно для забезпечення цілісності даних. На практиці SELECT INTO OUTFILE використовується для часткового бекапу великих таблиць, які не можна скопіювати ніяким іншим чином.

3. Практична частина

Імпорт даних:

Перед імпортом необхідно створити базу даних і в ній таблицю, створити CSV-файл, з однаковою кількістю стовпців і однаковим форматом, і необхідно запустити MySQL WorkBench.

Створюємо текстовий файл, вписуємо дані, наприклад для таблиці «clients» (рис. 3.1).

clients_id	name	surname	patronymic	phoneNumber	address
2	Anton	Baburin	Valrivich	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	957685746	St. Ahsarova 2-1-12
4	Oleg	Isaev	Denisovich	958475098	St. Artema 12-3-145

Рисунок 3.1 – Таблиця "clients"

Вона має 6 стовпців і вони мають такі типи:

clients_id - INT,

name - VARCHAR,

surname - VARCHAR,

patronymic -VARCHAR,

phoneNumber - BIGINT,

address – VARCHAR.

Відповідно заповнюємо текстовий файл (рис. 3.2):

```
5,Antonina,Maturin,Malrivich,0997654121,St. Tedar 1-2-31
6,Manto,Nabutin,Dalrivich,0987654323,St. Kintr 3-1-43
7,Ento,Aburin,Nalrivich,0977654322,St. Sdaki 12-3-43
```

Рисунок 3.2 – Вміст текстового файлу

При збереженні текстового файлу його розширення вказуємо як .scv.

Для того, щоб ми могли змінювати розширення файлів, ми повинні зайти в Пуск - Параметри (Панель управління) - У пошуку вбиваємо "параметри провідника" - Вид - Забираємо галочку з "Приховувати розширення для зареєстрованих типів".

Далі знаходимо наш текстовий файл та змінюємо його розширення на .scv.

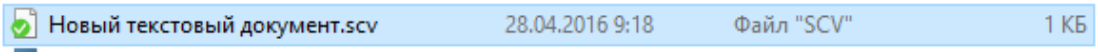


Рисунок 3.3 – Збережений документ на локальному диску

Далі, щоб додати вміст текстового файлу до таблиці, ми повинні пройти такі дії:

Крок 1:

Відкриваємо MySQL WorkBench, заходимо на сервер.

Крок 2:

Вибираємо нашу БД у лівому кутку.

Рисунок 3.4 – БД у середовищі Workbanch

Знаходимо розділ Tables.

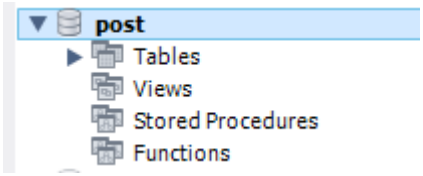


Рисунок 3.5 – БД у середовищі Workbanch

Знаходимо нашу таблицю.

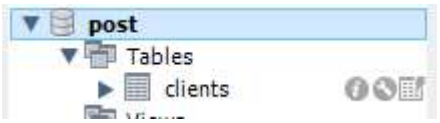


Рисунок 3.6 – БД у середовищі Workbanch

Натискаємо на показ таблиці.

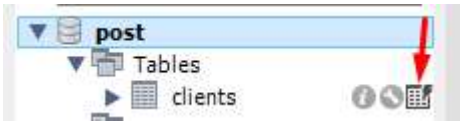


Рисунок 3.7 – БД у середовищі Workbanch

Крок 3:

У нас вивелася таблиця "clients" (див. рис. 3.8).

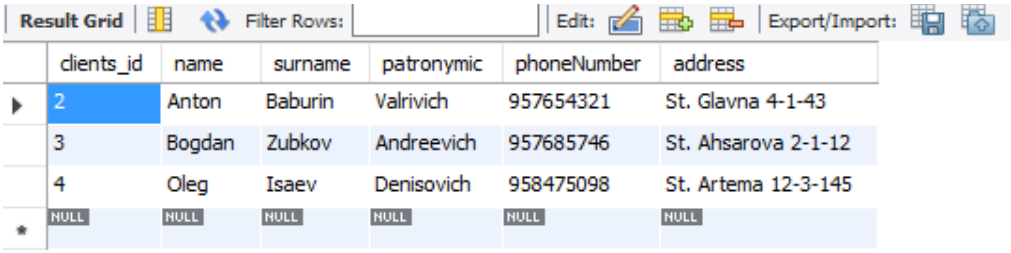


Рисунок 3.8 – Таблиця "clients"

Вибираємо імпорт:

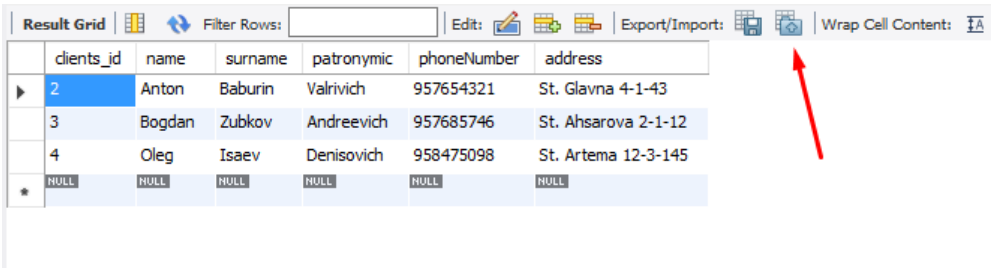


Рисунок 3.9 –Кнопка імпорту WB

Шукаємо наш scv файл і натискає відкрити:

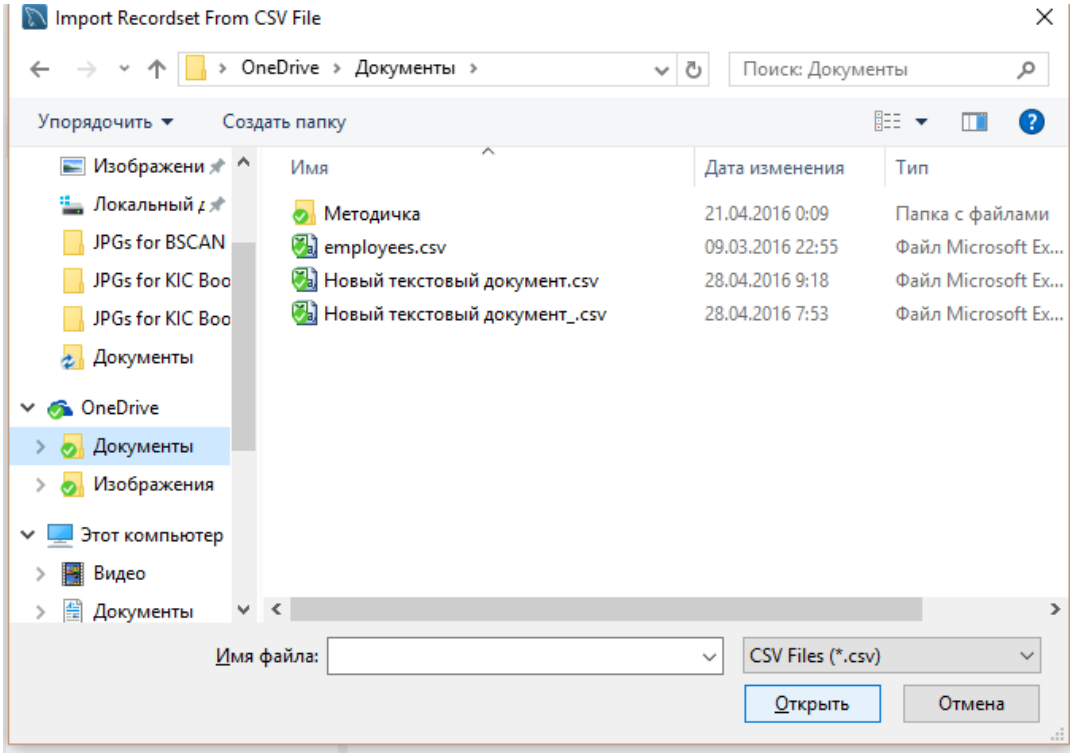


Рисунок 3.10 - Пошук файлу з розширенням .csv

Як бачимо, імпорт пройшов успішно (див. рис. 3.11):

clients_id	name	surname	patronymic	phoneNumber	address
2	Anton	Baburin	Valrivich	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	957685746	St. Ahsarova 2-1-12
4	Oleg	Isaev	Denisovich	958475098	St. Artema 12-3-145
5	Antonina	Maturin	Malrivich	0997654121	St. Tedar 1-2-31
6	Manto	Nabutin	Dalrivich	0987654323	St. Kintr 3-1-43
7	Ento	Aburin	Nalrivich	0977654322	St. Sdaki 12-3-43
NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 3.11 – Таблиця "clients" після імпорту

Імпорт текстових файлів

Щоб завантажити з файлу в таблицю потрібно виконати команду:

LOAD DATA LOCAL INFILE "F:\\post_office.txt" INTO TABLE post_office;

"F:\post_office.txt" – шлях до файлу, де F - жорсткий диск.

Після чого в нашій таблиці "post_office" з'явиться все, що ми завантажили з файлу (рис. 3.12):

```
1 •  select * from post_office;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

po_id	address	phoneNumber	weight_limit	type
1	St. Michyrina 1-4	673756742	15	department
2	St. Bagrationa 9-12	958756542	30	department
3	St. Nagornai 3-1	958438454	300	distribution center

Рисунок 3.12 – Таблиця "post_office"

Експорт таблиці:

Для експорту таблиці ми користуватимемося тією самою таблицею (рис. 3.13):

Result Grid | Filter Rows: | Edit: | Export/Import:

clients_id	name	surname	patronymic	phoneNumber	address
2	Anton	Baburin	Valrivich	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	957685746	St. Ahsarova 2-1-12
4	Oleg	Isaev	Denisovich	958475098	St. Artema 12-3-145
5	Antonina	Maturin	Malrivich	0997654121	St. Tedar 1-2-31
6	Manto	Nabutin	Dalrivich	0987654323	St. Kintr 3-1-43
7	Ento	Aburin	Nalrivich	0977654322	St. Sdaki 12-3-43
NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 3.13 – Таблиця "clients"

Але на цей раз ми вибираємо кнопку експорту:

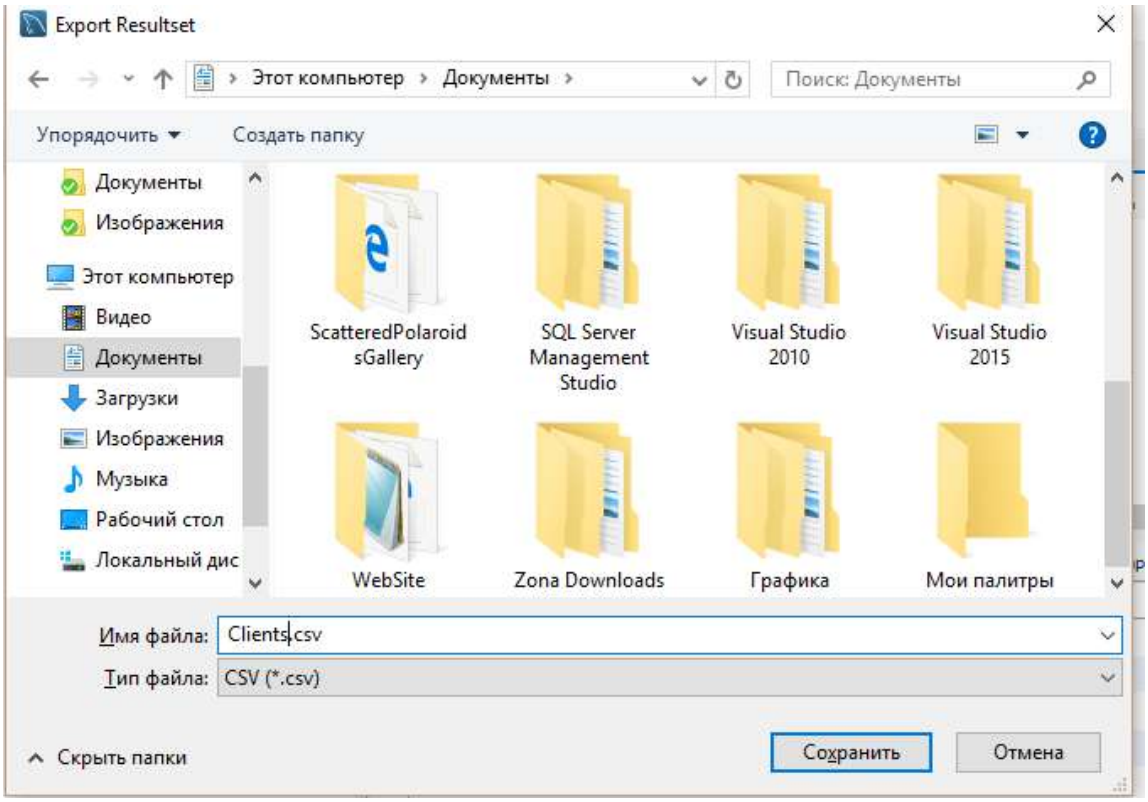


Рисунок 3.14 – Розміщення файлу на ПК

Зберігаємо файл, відкриваємо його за допомогою блокнота та дивимося результат експорту (див. рис. 3.15):

```
clients_id,name,surname,patronymic,phoneNumber,address
2,Anton,Baburin,Valrivich,957654321,"St. Glavna 4-1-43"
3,Bogdan,Zubkov,Andreevich,957685746,"St. Ahsarova 2-1-12"
4,Oleg,Isaev,Denisovich,958475098,"St. Artema 12-3-145"
5,Antonina,Maturin,Malrivich,0997654121,"St. Tedar 1-2-31"
6,Manto,Nabutin,Dalrivich,0987654323,"St. Kintr 3-1-43"
7,Ento,Aburin,Nalrivich,0977654322,"St. Sdaki 12-3-43"
```

Рисунок 3.15 – Таблиця "clients" після експорту в текстовому файлі

4 Зміст звіту:

7. Мета роботи.
8. Опис предметної сфери.
9. Короткий опис виконаного завдання та використаних засобів.
10. *SQL*запити та дії у робочому середовищі, які використовувалися.
11. Результати виконання створених запитів та виконаних дій.
12. Висновки про можливостіпрограмного пакета Workbench та мови запитів SQL для вирішення поставленого завдання.

Лабораторна робота №4

Тема: "Запити вибірки даних"

Мета: отримати навички створення запитів у MySQL Workbench.

1. Постановка задачі

3. Створення простого запиту. +
4. Створення підсумкового запиту. -
5. Створення запиту з параметром +
6. Створення перехресного запиту –
7. Створення запиту типу "зовнішнє об'єднання". +
8. Використання майстра запитів. +

2. Теоретична частина

MySQL запит –це звернення до бази даних MySQL, за допомогою якого ми можемо реалізувати: отримання, зміну, видалення, сортування, додавання та інші маніпуляції з даними бази.

SQL запити зазвичай виконують такі завдання:

- створення, модифікація та видалення таблиць бази даних;
- вставка інформації (записів) до таблиць бази даних;
- редагування інформації (записів) у таблицях бази даних;
- вибірка (витяг) інформації з таблиць бази даних;
- видалення інформації (записів) із бази даних.

Усе**MySQL** запити поділені на прості та складні запити. Нами також виділено категорію дуже простих запитів, структура яких дуже зрозуміла та не потребує додаткових роз'яснень.

Прості mysql запити- Запити в яких бере участь одна таблиця бази даних.
Складні mysql запити- Запити в яких можуть брати участь дві і більше таблиць БД.

Дуже прості MySQL запити

show databases;
Виведе список усіх баз.

show tables in base_name;
Покаже список усіх таблиць у базі даних base_name.

Прості MySQL запити

Знаючи структуру БД, таблиць у БД і полів, можна надсилати такі запити MySQL.

- **SELECT запити**

Слово SELECT, каже саме за себе, і стає зрозуміло, що, користуючись даними запитами, ми вибиратимемо (читати) інформацію з БД.

SELECT count(*) FROM table_name;
Виведе кількість усіх записів у таблиці

*SELECT * FROM table_name;*
Вибирає всі записи з таблиці БД

*SELECT * FROM table_name LIMIT 2,3;*
Вибирає 3 записи із таблиці, починаючи з 2 записи. Цей запит корисний під час створення блоку сторінок навігації.

*SELECT * FROM table_name ORDER BY field;*
Вибере всі записи з таблиці у порядку зростання значень щодо певного поля.

*SELECT * FROM table_name ORDER BY field DESC;*
Вибирає всі записи з таблиці, але вже в порядку зменшення (тобто у зворотному порядку).

*SELECT * FROM table_name ORDER BY field LIMIT 5;*
Вибирає 5 записів із таблиці, у порядку зростання.

*SELECT * FROM table_name WHERE field = 'значення';*
Вибирає всі записи таблиці, де поле відповідає значенню.

*SELECT * FROM table_name WHERE field LIKE '**%';*
Вибирає всі записи з таблиці, де значення поля починаються з певного символу.

*SELECT * FROM table_name WHERE field LIKE '%**' ORDER BY field;*
Вибирає всі записи з таблиці, де певне поле закінчується певний символ, і впорядковує записи у порядку зростання значення з іншого поля.

*SELECT * FROM table_name WHERE field IN (значення, значення, значення);*
Виведе всі записи таблиці у яких значення поля дорівнюватиме одному з трьох значень.

SELECT max(field) FROM person;
Вибере максимальне значення поля з таблиці (поле має чисельне значення).

- **INSERT запити**

Ці запити дозволяють вставити запис у таблицю БД. Тобто створити рядок у таблиці або додати інформацію до таблиці БД.

INSERT INTO table_name(field1, field2) VALUES (значення1, значення2);
Вставить у таблицю table_name, а точніше у поля таблиці відповідні значення.

- **UPDATE запити**

Направлено зміну вже наявних даних у таблиці БД.

UPDATE table_name SET field = значення 'WHERE field = 'значення';
Змінює значення поля на певне значення у таблиці table_name де ще одне поле має вказане значення.

- **DELETE запити**

Видаляють записи з таблиці БД.

DELETE FROM table_name WHERE field = 'значення';
Видаляє запис з table_name поле із зазначеним значенням.

Перехресний запит.

Перехресний запит- Це різновид запиту на вибірку. Під час перехресного запиту результати відображаються в таблиці, структура якої відрізняється від інших типів таблиць.

Перехресний запит можна задати SQL як інструкцію TRANSFORM. Для інструкцій TRANSFORM використовується наступний синтаксис:

TRANSFORM агрегатна_функція *інструкція_select*
PIVOT поле_зведеної_таблиці [IN (значення1[, значення2[, ...]])]

Інструкція TRANSFORM складається з наступних елементів (див. табл. 4.1):

Таблиця 4.1 – інструкція TRANSFORM

Елемент	Опис
агрегатна_функція	Агрегатна функція SQL, що обробляє вибрані дані.
інструкція_select	Інструкція SELECT.
поле_зведеної_таблиці	Поле або вираз, який потрібно використовувати для створення заголовків стовпців у набір результатів запиту.
значення1,*значення2*	Фіксовані значення, які використовуються для створення заголовків стовпців.

TRANSFORM
SELECT FROM GROUP BY PIVOT;

1. У першому рядку, після інструкції TRANSFORM, введіть вираз для розрахунку зведених значень, наприклад Sum([Кількість]).

   ПРИМІТКА:Якщо як джерело записів використовується більше однієї таблиці або одного запиту, вкажіть ім'я таблиці або запиту як частину імені кожного поля, наприклад Sum([Витрати].[Кількість]).

2. У другому рядку, після інструкції SELECT, введіть список полів або виразів полів, які потрібно використовувати як заголовки рядків. Елементи списку відокремлюються один від одного комами, наприклад: [Бюджет]. [ІД_відділу], [Витрати]. [Тип].
3. У третьому рядку, після інструкції FROM, перерахуйте таблиці або запити, які потрібно використовувати як джерела записів, наприклад Бюджет, Витрати.
4. У четвертому рядку, після інструкції GROUP BY, вкажіть ті самі поля, що й у реченні SELECT на кроці 6.
5. У п'ятому рядку після вказівки PIVOT введіть ім'я поля або вираз, який потрібно використовувати для заголовків стовпців, наприклад PIVOT [Бюджет].[Рік].

Додавання порядку сортування поля заголовків рядків.

Для налаштування порядку сортування у перехресному запиті у режимі SQL використовується пропозиція ORDER BY.

1. Вставте рядок між пропозиціями GROUP BY та PIVOT.
2. У новому рядку введіть ORDER BY, а потім пробіл.
3. Далі введіть ім'я поля або вираз, за яким потрібно виконати сортування, наприклад ORDER BY [Витрати].[Клас_витрат].

   За промовчанням пропозиція ORDER BY сортує значення зростання. Якщо потрібно виконати сортування за спаданням, введіть DESC після імені поля або виразу.

4. Якщо потрібно виконати сортування ще за одним полем або виразом, введіть кому, а потім вкажіть ім'я цього поля або вираз. Сортування буде виконано в порядку, в якому поля або вирази вказані у реченні ORDER BY.

   Обмеження значень, що використовуються як заголовки рядків або стовпців

   Дотримуючись вказівок, наведених нижче, можна задати список значень для заголовків стовпців і додати умови для полів із заголовками рядків. Щоб виконати ці дії, відкрийте перехресний запит у режимі SQL.

Вкажіть фіксовані значення, які потрібно використовувати як заголовки стовпців.

- Наприкінці пропозиції PIVOT введіть IN, а потім у дужках вкажіть через кому значення, які повинні стати заголовками стовпців. Наприклад, якщо ввести IN (2007, 2008, 2009, 2010), будуть задані чотири заголовки стовпців: 2007, 2008, 2009, 2010.

ПРИМІТКА:Якщо ввести фіксоване значення, яке не відповідає значенню з поля зведеної таблиці, воно стане заголовком порожнього стовпця.

Додавання умов запиту для обмеження заголовків рядків

1. Вставте новий рядок після пропозиції FROM.
2. Введіть WHERE, а потім – умова для поля.

Якщо потрібно застосувати додаткові умови, розширте пропозицію WHERE за допомогою операторів AND та OR. Використовуючи дужки, ви також можете поєднати умови в логічні групи.

Оператор GROUP BY

Пропозиція GROUP BY використовується визначення груп вихідних рядків, до яких можуть застосовуватися агрегатні функції (COUNT, MIN, MAX, AVG і SUM). Якщо ця пропозиція відсутня, і використовуються агрегатні функції, всі стовпці з іменами, згаданими в SELECT, повинні бути включені в агрегатні функції, і ці функції будуть застосовуватися до всього набору рядків, які задовольняють предикату запиту. В іншому випадку всі стовпці списку SELECT, що не увійшли до агрегатних функцій, повинні бути вказані в пропозиції GROUP BY. У результаті всі вихідні рядки запиту розбиваються на групи, що характеризуються однаковими комбінаціями значень цих стовпцях. Після чого до кожної групи буде застосовано агрегатні функції. Слід мати на увазі, що для GROUP BY всі значення NULL трактуються як рівні, тобто при групуванні по полю, що містить значення NULL, всі такі рядки потраплять в одну групу.

Якщо за наявності пропозиції GROUP BY, у пропозиції SELECT відсутні агрегатні функції, запит просто поверне по одному рядку з кожної групи. Цю можливість, поряд із ключовим словом DISTINCT, можна використовувати для виключення дублікатів рядків у результуючому наборі.

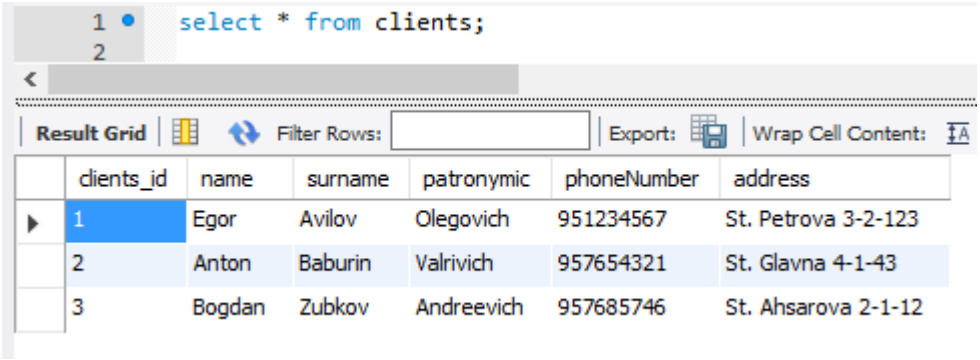
3. Практична частина

Створення простого запиту

Щоб створити просто запит на вибірку потрібно виконати команду (результат виконання див. рис. 4.1):

SELECT * від клієнтів;

* - вибірка по всіх полях.



clients_id	name	surname	patronymic	phoneNumber	address
1	Egor	Avilov	Olegovich	951234567	St. Petrova 3-2-123
2	Anton	Baburin	Valrivich	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	957685746	St. Ahsarova 2-1-12

Рисунок 4.1 – Результат простого запиту

Створення підсумкового запиту

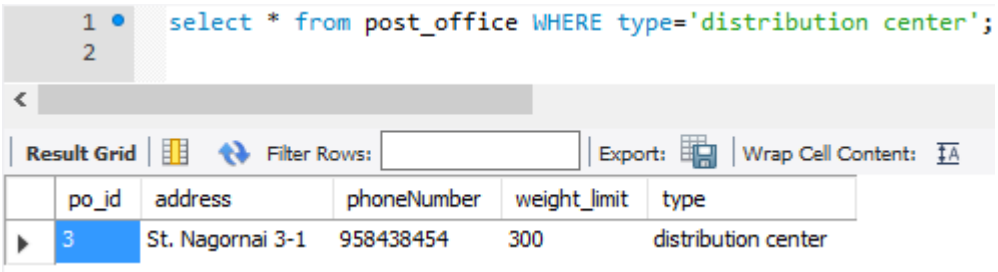
????

Рисунок 4.2 – Результат підсумкового запиту

Створення запиту з параметром

Виберемо, за допомогою запиту, із таблиці «Відділення», відділення які є розподільчими центрами (не включаючи у вибірку пункти прийому/видачі) рис. 4.3:

SELECT * FROM post_office WHERE type='distribution center';



po_id	address	phoneNumber	weight_limit	type
3	St. Nagornai 3-1	958438454	300	distribution center

Рисунок 4.3 – Результат запиту з параметром

Створення перехресного запиту

???

Створення запиту типу "зовнішнє об'єднання"

Для створення такого запиту додамо дані до таблиці «Журнал оформлення декларацій».

Додавати будемо запитом завантаження із файлу:

LOAD DATA LOCAL INFILE "F:\\registrations_journal.txt" INTO TABLE registrations_journal;

Дивитись файл "registrations_journal.txt" в дод. матеріалах.

Тепер створимо запит на об'єднання даних із таблицями: «Клієнти» (рис. 4) та «Журнал оформлення декларації» (рис. 4.5).

```
1 • select * from clients;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address
1	Egor	Avilov	Olegovich	459568	951234567	St. Petrova 3-2-123
2	Anton	Baburin	Valrivich	854574	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	125896	957685746	St. Ahsarova 2-1-12
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23

Малюнок 4.4 – Таблиця «Клієнти»

```
1 • select * from registrations_journal;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

rj_id	sender_id	sender_po_id	receiver_id	receiver_po_id	departure_date	sending_time
1	2	1	3	2	2016-04-21	12:11:00
2	3	2	1	4	2016-03-14	17:15:00
3	1	4	2	1	2016-02-11	18:00:00

Малюнок 4.5 – Таблиця «Журнал оформлення декларації»

Виберемо ID, Ім'я, Прізвище, номери паспортів, ID у журналі оформлення декларації клієнтів, які здійснили надсилання посилки (рис. 4.6):

SELECT clients.clients_id, clients.name,clients.surname, clients.passportNumber, registrations_journal.rj_id FROMregistrations_journal
INNER JOIN clients
ONsender_id=clients_id;

```
SELECT clients.clients_id, clients.name,
clients.surname, clients.passportNumber,
registrations_journal.rj_id
FROM registrations_journal
INNER JOIN clients
ON sender_id=clients_id;
```

Result Grid | Filter Rows: | Export:

clients_id	name	surname	passportNumber	rj_id
1	Egor	Avilov	459568	3
2	Anton	Baburin	854574	1
3	Bogdan	Zubkov	125896	2

Рисунок 4.6 – Результат запиту із зовнішнім об'єднанням

Розглянемо запит докладніше:

Після оператора вибірки SELECT йдуть назви полів, які ми хочемо бачити при виведенні на екран. Після оператора FROM слід таблиця з якої вибирати, а після INNER JOIN – з якою об'єднувати під час пошуку відповідностей, які вказані після ON. Щоб запит спрацював правильно, у запиті має бути ясно зазначено, за якими полями шукати відповідності між таблицями.

Використання майстра запитів

Стандартні запити в MySQL Workbench:

І так, стандартний запит на виведення таблиці ви можете зробити за допомогою середовища MySQL Workbench:

Шукаємо нашу БД, у ній нашу таблицю, при наведенні на неї, з'являються 3 значки, натискаємо на крайній праворуч (рис. 4.7):

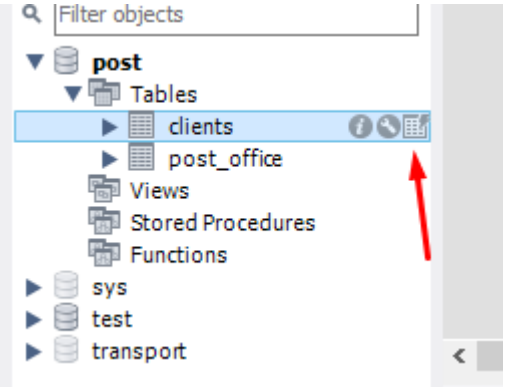


Рисунок 4.7 – Розташування таблиці

Результатом є висновок нашої обраної таблиці (рис. 4.8):

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address
2	Anton	Baburin	Valrivich	0	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	0	957685746	St. Ahsarova 2-1-12
4	Oleg	Isaev	Denisovich	0	9584758	St. Artema 12-3-145
5	Ivan	Manevich	Ivanovich	0	951855758	St. Motornai 1-4-23
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 4.8 – Виведення таблиці запитом засобами MySQL Workbench

Нещодавно додане поле pasportNumber заповнене нулями, додати значення ми можемо з конструктора (рис. 4.9):

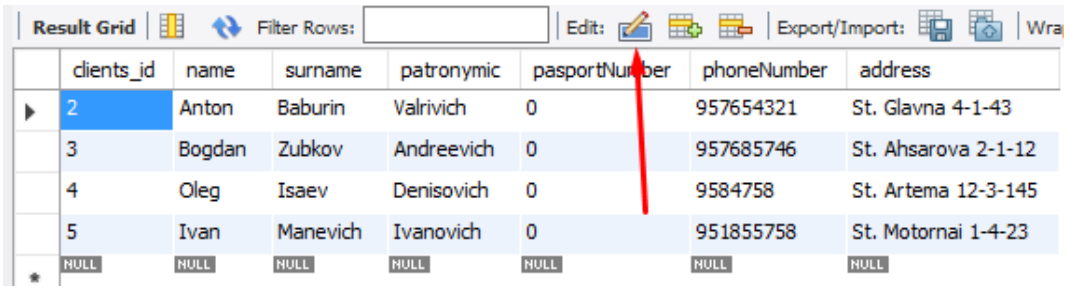


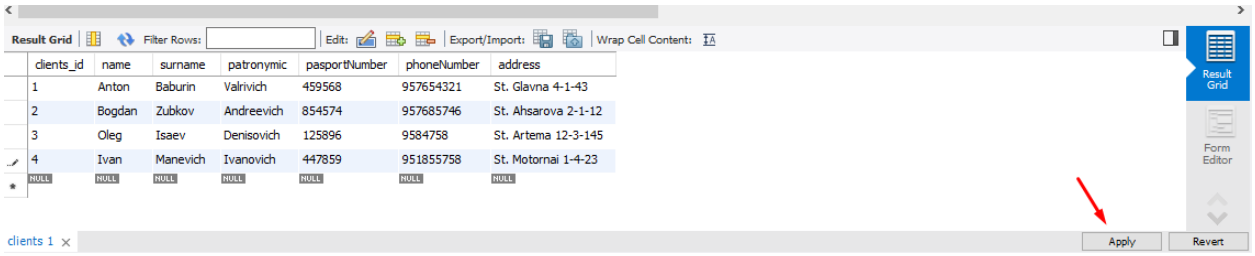
Рисунок 4.9 – Додати значення MySQL Workbench

Так, додамо дані, для поля pasportNumber(значення записуємо відразу в таблицю) рис. 4.10:

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23
NULL	NULL	NULL	NULL	NULL	NULL	NULL

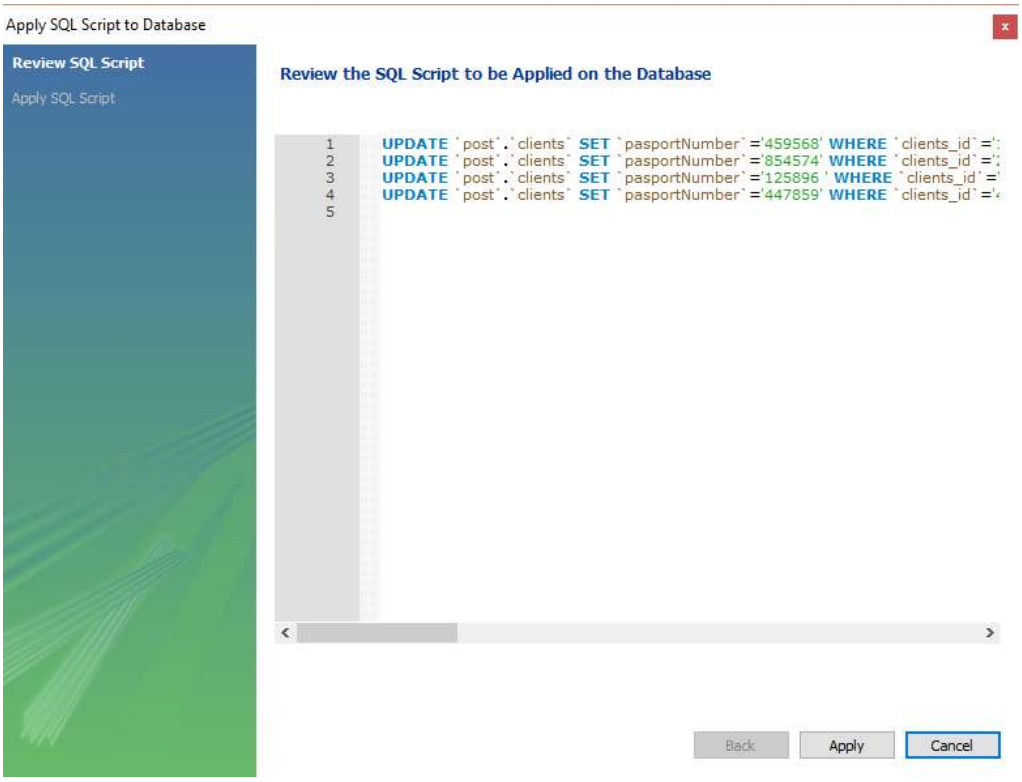
Рисунок 4.10 – Введення значення полів

Натискаємо Apply (рис. 4.11):



Малюнок 4.11 – Збереження введених даних у таблицю

Показується згенерований запит, натискаємо Apply (рис. 4.12):



Малюнок 4.12 – Згенерований запит

Наш запит пройшов успішно, натискаємо Finish (рис. 4.13):

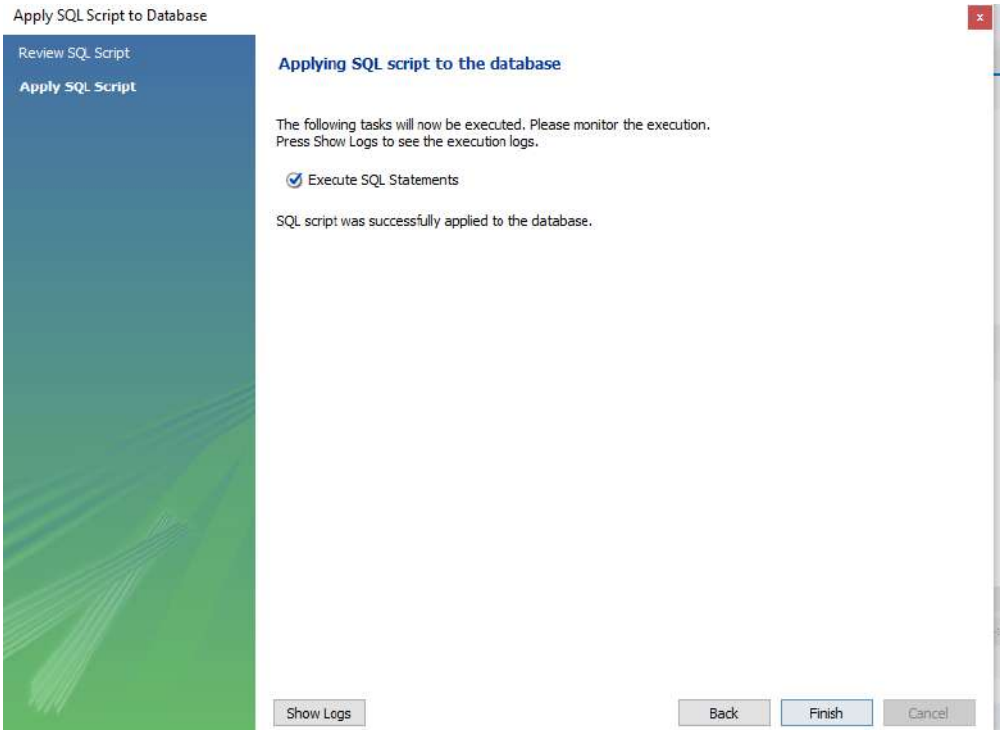


Рисунок 4.13 – Успішне завершення запиту

До нашої таблиці додалися дані (рис. 4.14):

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 4.14 – Додані дані у таблиці

Для додавання нового ряду (рис. 4.15):

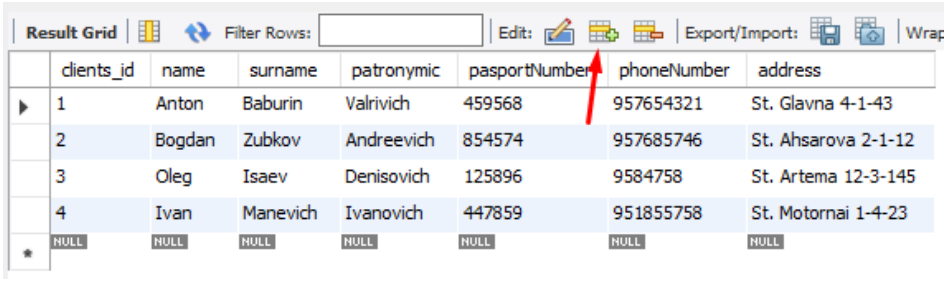


Рисунок 4.15 – Додавання нового ряду

Для видалення ряду (рис. 4.16):

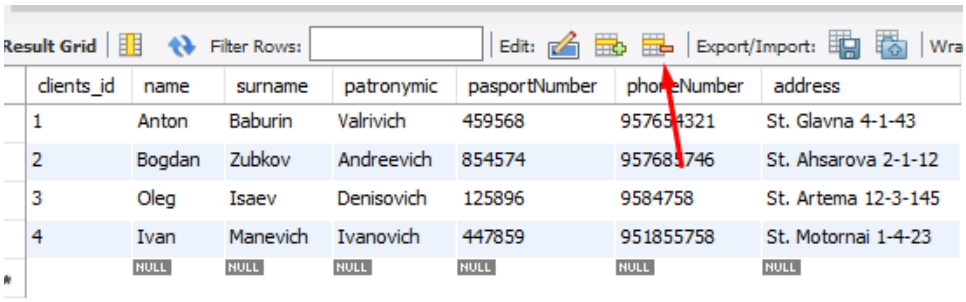


Рисунок 4.16 – Видалення ряду

Виділяємо ряд, який ви хочете видалити (рис. 4.17):

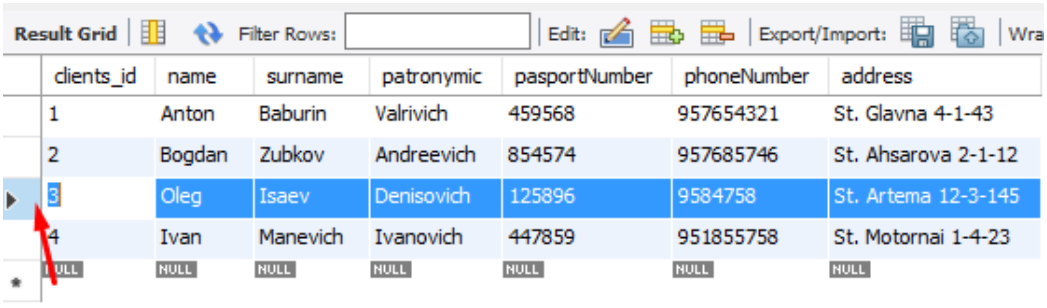


Рисунок 4.17 – Вибір ряду до видалення

І натискаємо видалення рада (рис. 4.18):

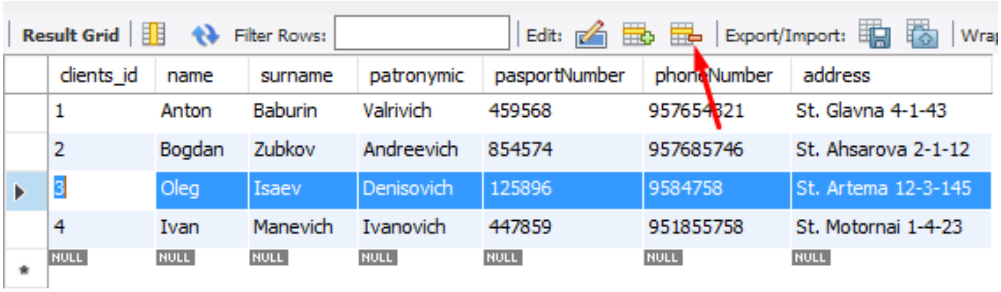
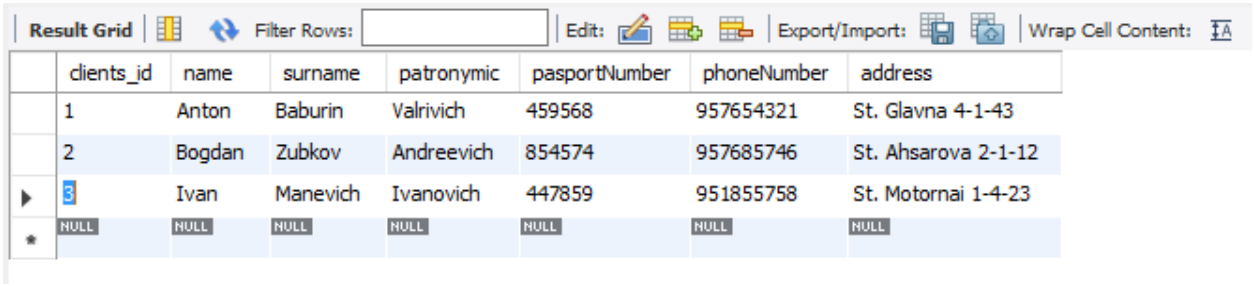


Рисунок 4.18 – Натискаємо видалення ряду

Ряд видалений (рис. 4.19):



Малюнок 4.19 – Ряд видалений

Для підтвердження видалення натискаємо Apply, а повернення таблиці до вихідного стану Revert (рис. 4.20):

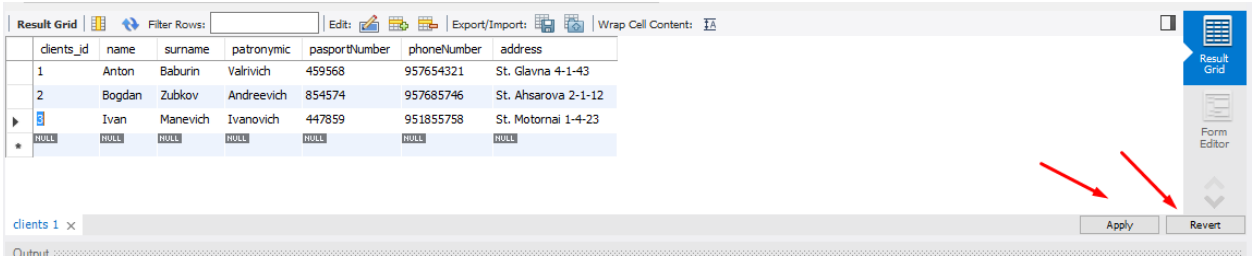


Рисунок 4.20 – Підтвердження зміни даних

4 Зміст звіту:

13. Мета роботи.
14. Опис предметної сфери.
15. Короткий опис виконаного завдання та використаних засобів.
16. *SQL*запити, що використовувалися.
17. Роздрук результатів виконання створених запитів.
18. Висновки про можливостіпрограмного пакета Workbench та мови запитів SQL для вирішення поставленого завдання.

Лабораторна робота 5

Тема: запити у MySQL.

Мета: створення запитів у MySQL

1. Постановка задачі

1. Створення запиту об'єднання таблиць.
2. Створення багатотабличного запиту. 3. Створення багатотабличного запиту із параметрично заданою умовою відбору. 4. Групові запити (агрегатні функції) 5. Запити об'єднання інструкцій SELECT (UNION, ORDER BY) 6. Запити на видалення даних на SQL.

2. Теоретична частина

Розглянуті до цього розділу SQL запити використовували одну таблицю. Але реальних додатках часто потрібно використовувати відразу кілька таблиць бази даних. Запити, які звертаються одночасно до кількох таблиць, називаються багатотабличними запитами. Саме таких запитах проявляється одне з переваг реляційних баз даних — зв'язок таблиць друг з одним.

Оскільки реляційні БД припускають розташування незалежних сутностей у різних таблицях, необхідний механізм зведення кількох таблиць воєдино одному запиті. Цей механізм відомий як з'єднання (join).

Внутрішні сполуки:

```
mysql> SELECT t1. field1, t1. field2, t2. field3
    FROM table1 t1 JOIN table2 t2
    ON t1. field4 = t2. field4;
```

„Де field1, field2, field3 – поля з двох таблиць, field4 – поле яке має зовнішній ключ і пов'язане з іншою таблицею.

Якщо певне значення стовпця є в одній таблиці, але його немає в іншій, з'єднання рядків не відбувається, і вони не включаються до результуючого набору. Такий тип з'єднання називають внутрішнім з'єднанням (inner join); це найбільш широко використовуваний тип з'єднання.

Якщо потрібно включити всі рядки тієї чи іншої таблиці незалежно від відповідності, можна скористатися зовнішнім з'єднанням (outer join).

Якщо імена стовпців, що використовуються для з'єднання двох таблиць, збігаються (що має місце в попередньому запиті), можна замість під блоку on застосувати підблок using:

```
SELECT t1. field1, t1. field2, t2. field3
FROM table1 t1 INNER JOIN table2 t2
USING (field4);
```

Оскільки using – скорочений запис, який може використовуватися тільки в певній ситуації, щоб уникнути плутанини я завжди віддаю перевагу підблоку on

З'єднання трьох та більше таблиць:

З'єднання трьох таблиць аналогічно з'єднанню двох, але є невелика хитрість. При з'єднанні двох таблиць є дві таблиці і один тип з'єднання в блоці від, а також єдиний підблок on, що визначає, як з'єднуються таблиці. При з'єднанні трьох таблиць присутні три таблиці та два типи з'єднання в блоці from, а також два підблоки on.

Створення багатотабличного запиту (зокрема об'єднання трьох таблиць) з параметрично-заданим умовою відбору.

SELECTt1. field1, t2. field2, t3.field3.1, t3.field3.2

FROMtable1 t1INNER JOINtable2 t2

ON t1. field4 = t2. field4

INNER JOIN table3 t3

ON t1.field5 = t3.field5 (поля за якими встановлено зв'язок між таблицями)

WHERE t1.field6 = 'value';

Запит, що використовує три або більше таблиць, можна уявити, як снігова грудка, що котиться з гори. Перші дві таблиці запускають цей ком, а кожна наступна таблиця вносить у нього свій внесок. Снігова куля можна розглядати як проміжний результуючий набір (intermediate result set), який підбирає все більше і більше стовпців у міру з'єднання з наступними таблицями.

Угруповання та агрегати:

Зазвичай, дані зберігаються з найнижчим рівнем деталізації, який може знадобитися будь-якому користувачеві бази даних. Якщо Чаку для бухгалтерського обліку потрібно подивитися операції одного клієнта, БД повинна бути таблиця, що зберігає операції окремого клієнта. Однак це не означає, що всі користувачі повинні працювати з даними у вигляді, в якому вони зберігаються в БД. Основна увага приділена угрупованню та агрегуванню даних, які забезпечують користувачам можливість працювати з даними на вищому рівні деталізації, ніж той, з яким вони зберігаються у БД.

Агрегуючі функціїдозволяють отримувати з таблиці зведену (агреговану) інформацію, виконуючи операції над групою рядків таблиці. Для завдання у**SELECT**запиті агрегувальних операцій використовуються такі ключові слова

- COUNTвизначає кількість рядків або значень поля, вибраних за допомогою запиту і не є значеннями NULL;

- SUMобчислює арифметичну суму всіх вибраних значень даного поля;
- AVGобчислює середнє для всіх вибраних значень даного поля;
- MAXобчислює найбільше зі всіх обраних значень даного поля;
- MINобчислює найменше зі всіх вибраних значень даного поля

У SELECT-запиті агрегують функції використовуються аналогічно іменам полів, при цьому останні (імена полів) використовуються як аргументи цих функцій.

*Функція AVG*призначена для підрахунку середнього значення поля на множині записів таблиці.

Приклади використання оператора та агрегатних функцій див. у розділі 2 (Практична частина).

UNION:

Ви можете помістити численні запити разом і об'єднати їх вивод, використовуючи пропозицію UNION. Пропозиція UNION поєднує виведення двох або більше SQL запитів у єдиний набір рядків та стовпців.

КОЛИ ВИ МОЖЕТЕ РОБИТИ ОБ'ЄДНАННЯ МІЖ ЗАПИТами?

Коли два (або більше) запити піддаються об'єднанню, їх стовпці виводу мають бути сумісні для об'єднання. Це означає, що кожен запит повинен вказувати однакову кількість стовпців і в тому ж порядку, що і перший, другий, третій, і так далі, і кожен повинен мати тип, сумісний з кожним. Значення сумісності типів - змінюється. ANSI слідкує за цим дуже суворо і тому числові поля повинні мати однаковий числовий тип і розмір, хоча деякі імена ANSI для цих типів є - синонімами. (Див. Додаток B для подробиць про ANSI числових типів.) Крім того, символьні поля повинні мати однакову кількість символів (значення призначеного номера, не обов'язково таке ж як номер, що використовується). Добре, що деякі SQL програми мають більшу гнучкість, ніж це, визначається ANSI. Типи, не визначені ANSI, такі як DATA та BINARY, зазвичай мають збігатися з іншими стовпцями такого ж нестандартного типу. Довжина рядка може стати проблемою. Більшість програм дозволяють поля змінної довжини, але вони не обов'язково використовуватимуться з UNION. З іншого боку, деякі програми (і ANSI теж) вимагають, щоб символьні поля були точно рівної довжини. У цих питаннях ви повинні проконсультуватися з документацією вашої програми. Інше обмеження на сумісність - це коли пусті значення (NULL) заборонені в будь-якому стовпці об'єднання, причому ці значення необхідно заборонити і для відповідних стовпців в інших запитах об'єднання. Порожні значення (NULL) заборонені з обмеженням NOT NULL, яке обговорюватиметься в Розділі 18. Крім того, ви не можете використовувати UNION у підзапитах, а також не можете використовувати агрегатні функції у пропозиції SELECT запиту об'єднання. (Більшість програм нехтують цими обмеженнями.)

Перехресні з'єднання:

Якщо дійсно потрібно отримати декартове твір двох таблиць, має бути задано перехресне з'єднання:

SELECT t2. field1, t1. field1, t2. field2

FROM table1 t1 CROSS JOIN table2 t2;

Інші перехресні запити дивись у конспекті лекцій, або у п 2.

Запити на видалення даних на SQL:

Запит на видалення всіх записів з таблиці mysql:

TRUNCATE TABLE table_name

Тут все просто, ми пишемо команду TRUNCATE TABLE, яка видаляє всі записи з таблиці - table_name.

А тепер трохи складніше, запит на видалення одного або більше записів із бази:

mysql_query("DELETE FROM `table_name`");

Цей запит видалить також усі записи в таблиці table_name, оскільки у запиті відсутня умова (WHERE).

А тепер спробуємо трохи інший запит:

mysql_query("DELETE FROM `table_name` WHERE `id`='7'");

Цей запит видалить запис у якого id дорівнює 7, так як у нас стоїть умова (WHERE `id` = '7').

Нарешті хочу загострити увагу вже досвідченіших користувачів, на тому, що оператор TRUNCATE TABLE працює швидше ніж DELET без заданої умови, тобто запит:

mysql_query("TRUNCATE TABLE `table_name`");

буде працювати швидше ніж запит:

mysql_query("DELETE FROM `table_name`");

І ще хотілося б додати про оптимізатор запитів – те, що оптимізатор СУБД MySQL автоматично використовує оператор TRUNCATE TABLE <ім'я_таблиці>, якщо оператор DELETE не містить WHERE-умови чи конструкції LIMIT.

3. Практична частина

1. Створення запиту на об'єднання таблиць

Багатотабличні запити створюються за допомогою JOIN. У БД MySQL JOIN поєднує стовпці під час вибірки.

Отже, візьмемо наші 2 таблиці clients і post_office і напишемо найпростіший багатотабличний запит на об'єднання таблиць по полю id (рис. 1).

```
use post;
select * from clients c join post_office p on c.clients_id=p.id_office;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address	id_office	adress	telephone	weight_limit	type
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43	1	St. Pobedi	973332211	20	department
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145	3	Av. Asharova	678831223	60	Admission point
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23	4	St. Nachimova 3-2	977595158	30	department

Рисунок 1 – Об'єднання таблиці за допомогою JOIN

2. Створення багатотабличного запиту

Для даного типу запиту використовуємо INNER JOIN (CROSS JOIN (внутрішнє (перехресне) об'єднання)), цей тип об'єднання дозволяє витягувати рядки, які обов'язково присутні у всіх таблицях, що об'єднуються (рис. 2).

```
use post;
SELECT * FROM clients INNER JOIN post_office;
```

ult Grid | Filter Rows: | Export: | Wrap Cell Content:

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address	id_office	adress	telephone	weight_limit	type
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43	1	St. Pobedi	973332211	20	department
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12	1	St. Pobedi	973332211	20	department
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145	1	St. Pobedi	973332211	20	department
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23	1	St. Pobedi	973332211	20	department
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43	3	Av. Asharova	678831223	60	Admission point
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12	3	Av. Asharova	678831223	60	Admission point
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145	3	Av. Asharova	678831223	60	Admission point
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23	3	Av. Asharova	678831223	60	Admission point
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43	4	St. Nachimova 3-2	977595158	30	department
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12	4	St. Nachimova 3-2	977595158	30	department
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145	4	St. Nachimova 3-2	977595158	30	department
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23	4	St. Nachimova 3-2	977595158	30	department

Рисунок 2 – Багатотабличний запит із використанням INNER JOIN

Також є безліч інших форм внутрішнього об'єднання:

SELECT * FROM clients CROSS JOIN post_office;
SELECT * FROM clients JOIN post_office;
SELECT * FROM clients, post_office;

3. Створення багатотабличного запиту з параметричною умовою відбору

Як правило, декартове твір таблиць потрібно нечасто, частіше потрібно вибрати ті записи, які зіставлені один одному. Зробити це можна, якщо задати умову відбору, використовуючиONабоUSING.

Форма запитів з використанням:

Select * з table1inner join table2 using(id);

id – загальне поле таблиць table1 та table2.

Запит з on демонструвався у 1 пункті.

Якщо об'єднувати таблиці через кому, то не можна використовувати конструкціїONiUSINGтому умова може бути задана тільки в конструкціїWHERE (рис.3).

```
use post;
SELECT * FROM clients, post_office where post_office.id_office=clients.clients_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

clients_id	name	surname	patronymic	pasportNumber	phoneNumber	address	id_office	adress	telephone	weight_limit	type
1	Anton	Baburin	Valrivich	459568	957654321	St. Glavna 4-1-43	1	St. Pobedi	973332211	20	department
2	Bogdan	Zubkov	Andreevich	854574	957685746	St. Ahsarova 2-1-12	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
3	Oleg	Isaev	Denisovich	125896	9584758	St. Artema 12-3-145	3	Av. Asharova	678831223	60	Admission point
4	Ivan	Manevich	Ivanovich	447859	951855758	St. Motornai 1-4-23	4	St. Nachimova 3-2	977595158	30	department

Рисунок 3 – Запит з використанням where

І так запити із внутрішнім об'єднанням можна створювати такими способами:

select * from table1, table2, [table3, ...] [where Умова1 [Умова2 ...];
select * from table1 [iner | cross] join table2 [(onУмова1 [Умова2 ...]) | (using(Поле))]
Також за допомогою JOIN можна об'єднати таблицю саму на себе (рис. 4):

```
use post;
SELECT * FROM post_office join post_office as p;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id_office	adress	telephone	weight_limit	type	id_office	adress	telephone	weight_limit	type
1	St. Pobedi	973332211	20	department	1	St. Pobedi	973332211	20	department
2	Av. Kolochkovskaua	985544221	40	Admissionpoint	1	St. Pobedi	973332211	20	department
3	Av. Asharova	678831223	60	Admission point	1	St. Pobedi	973332211	20	department
4	St. Nachimova 3-2	977595158	30	department	1	St. Pobedi	973332211	20	department
1	St. Pobedi	973332211	20	department	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
2	Av. Kolochkovskaua	985544221	40	Admissionpoint	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
3	Av. Asharova	678831223	60	Admission point	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
4	St. Nachimova 3-2	977595158	30	department	2	Av. Kolochkovskaua	985544221	40	Admissionpoint
1	St. Pobedi	973332211	20	department	3	Av. Asharova	678831223	60	Admission point
2	Av. Kolochkovskaua	985544221	40	Admissionpoint	3	Av. Asharova	678831223	60	Admission point
3	Av. Asharova	678831223	60	Admission point	3	Av. Asharova	678831223	60	Admission point
4	St. Nachimova 3-2	977595158	30	department	3	Av. Asharova	678831223	60	Admission point
1	St. Pobedi	973332211	20	department	4	St. Nachimova 3-2	977595158	30	department
2	Av. Kolochkovskaua	985544221	40	Admissionpoint	4	St. Nachimova 3-2	977595158	30	department
3	Av. Asharova	678831223	60	Admission point	4	St. Nachimova 3-2	977595158	30	department
4	St. Nachimova 3-2	977595158	30	department	4	St. Nachimova 3-2	977595158	30	department

Рисунок 4 – Об'єднання таблиці самої себе

4. Групові запити

Виведення кількості записів у таблиці clients рис.5.

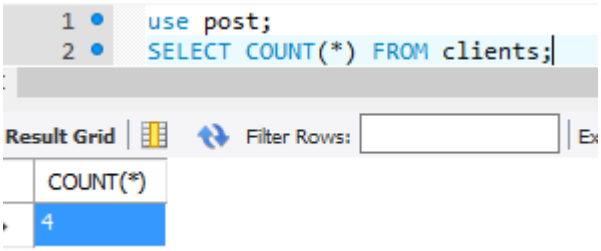


Рисунок 5 – Кількість записів у таблиці clients

Виведення середнього значення weight_limit із таблиці post_office рис.6.

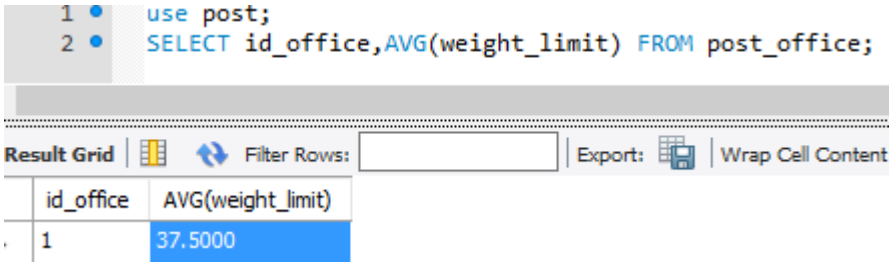


Рисунок 6 – Середнє значення weight

Створення групового запиту з допомогою group by (рис. 7).

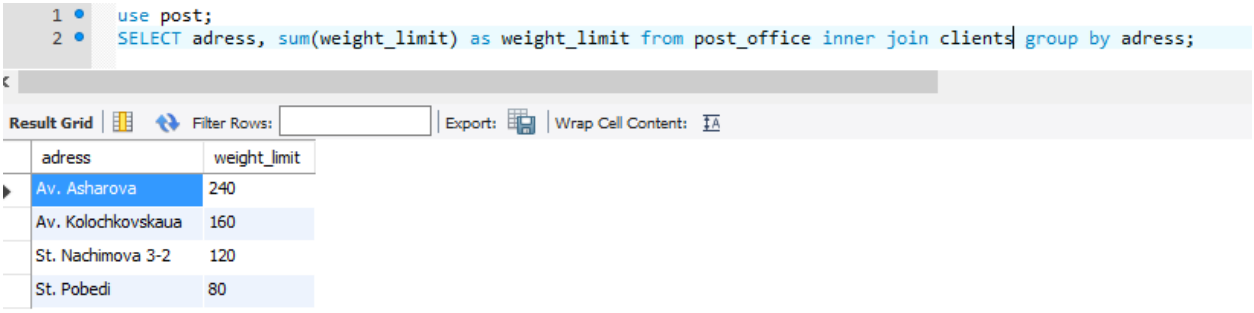


Рисунок 7 – Груповий запит, створений за допомогою group by

5. Запити на об'єднання інструкцій SELECT (UNION, ORDER BY)

Запит із використанням UNION (рис. 8).

```
1  use post;
2  SELECT clients_id id, address adress from clients union select id_office id, adress adress from post_office;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

id	adress
1	St. Glavna 4-1-43
2	St. Ahsarova 2-1-12
3	St. Artema 12-3-145
4	St. Motornai 1-4-23
1	St. Pobedi
2	Av. Kolochkovskaua
3	Av. Asharova
4	St. Nachimova 3-2

Рисунок 8 – Запит з використанням union

Запит з використанням ORDER BY:

```
1  use post;
2  SELECT
3      name, surname, adress
4  FROM
5      clients join post_office
6  ORDER BY surname;
```

Result Grid | Filter Rows: | Export:

name	surname	adress
Anton	Baburin	Av. Kolochkovskaua
Anton	Baburin	Av. Asharova
Oleg	Isaev	Av. Asharova
Oleg	Isaev	St. Nachimova 3-2
Oleg	Isaev	St. Pobedi
Oleg	Isaev	Av. Kolochkovskaua
Ivan	Manevich	Av. Asharova
Ivan	Manevich	St. Nachimova 3-2
Ivan	Manevich	St. Pobedi
Ivan	Manevich	Av. Kolochkovskaua
Bogdan	Zubkov	St. Nachimova 3-2
Bogdan	Zubkov	St. Pobedi
Bogdan	Zubkov	Av. Kolochkovskaua
Bogdan	Zubkov	Av. Asharova

Рисунок 9 – Запит з використанням ORDER BY

6. Запити на видалення даних на SQL
   Запит на видалення даних із таблиці:
   DELETE FROM post.post_office WHERE id_office=4;

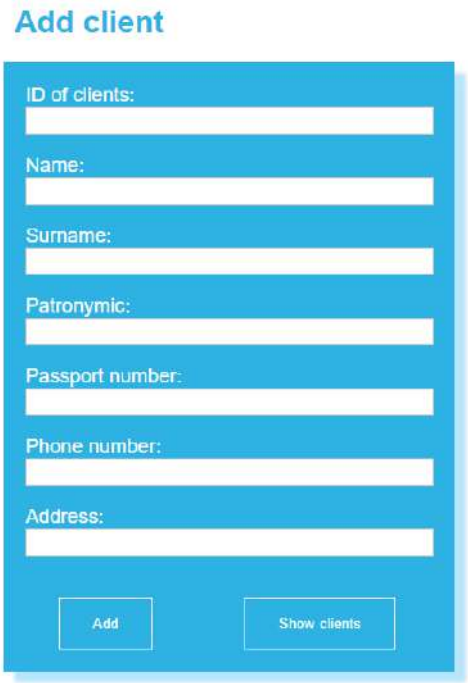
4. Зміст звіту

1. Мета роботи.
2. Опис предметної сфери.
3. Опис виконання завдання.
4. Роздрук результатів виконання лабораторної роботи.

Лабораторна робота №6

Тема: Робота з формами

Форма для додавання клієнта див. 6.1.



Малюнок 6.1 – Форма додавання клієнта

Використовуючи клавішу показати клієнтів, ми можемо вивести таблицю "clients" на екран (рис. 6.2):

clients_id	name	surname	patronymic	passportNumber	phoneNumber	address
1	Egor	Avilov	Olegovich	459568	951234567	St. Petrova 3-2-123
2	Anton	Baburin	Valrivich	854574	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	125896	957685746	St. Ahsarova 2-1-12
4	Ivan	Manevich	Ivanovich	440011	951855758	St. Motornai 1-4-23

Малюнок 6.2 - Виведення таблиці "clients"

Після введення у форму даних і натискання кнопки «Add», внизу побачимо повідомлення про те, що все пройшло добре (рис. 6.3):

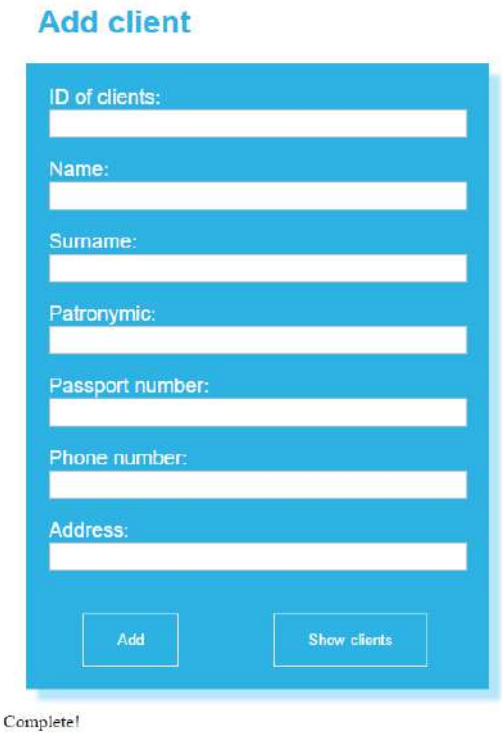


Рисунок 6.3 – Повідомлення про додавання клієнта до таблиці

Тепер натиснемо кнопку "Show clients", щоб подивитися на доданого клієнта (рис. 6.4):

clients_id	name	surname	patronymic	passportNumber	phoneNumber	address
1	Egor	Avilov	Olegovich	459568	951234567	St. Petrova 3-2-123
2	Anton	Baburin	Valrivich	854574	957654321	St. Glavna 4-1-43
3	Bogdan	Zubkov	Andreevich	125896	957685746	St. Ahsarova 2-1-12
4	Ivan	Manevich	Ivanovich	440011	951855758	St. Motornai 1-4-23
5	Michael	Sokolov	Andreevich	457864	664885158	St. New Bowaria 2-4-1

Рисунок 6.4 - Виведення таблиці з новим клієнтом

Як бачимо, у таблиці з'явився новий клієнт.

Лабораторна робота №7

Тема: «Тригери»

Мета: отримати навички створення тригерів у MySQL Workbench.

1. Постановка задачі

9. Створення простого тригера за допомогою MySQL Workbench
10. Створення простого тригера за допомогою SQL-запитів

2. Теоретична частина

Тригери –це процедури спеціального виду, 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 трireру проводяться до виконання операції зміни таблиці;

AFTER - дії трireру 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

3. Практична частина

1. Створення простого тригера за допомогою MySQL Workbench

У базі даних «гуманітарна_допомога» необхідно реалізувати тригер вставки, який ітеруватиме кількість збройних конфліктів у конкретному регіоні. Для цього працюватимемо з таблицями «доп_інф_о_державі» та «регіон».

id_государства	id_региона	фамилия_главы_государства	площадь_территории	количество_населения	вооружённый_конфликт
58	5	Гуайдо	916455	30761000	да
91	34	Ковинд	3287263	1340468000	да
95	35	Кхан	678500	5288522	да
380	7	Порошенко	923768	198369140	да
509	13	Моиз	603700	42000000	да
675	8	О'Нил	27750	9893934	
963	12	Асад	462840	7344638	да
NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 1 – Таблиця «доп_інф_про_державі»

id_региона	название_региона	количество_государств	количество_вооружённых_конфликтов	количество_насе
4	Северная Америка	2	0	354830825
5	Южная Америка	15	0	387489196
7	Центральная и Восточная Европа	44	0	145800000
8	Австралия и Океания	17	0	40117432
11	Западная Африка	16	0	362201579
12	Ближний Восток	19	0	436005542
13	Центральная Америка	21	0	89503839
34	Южная Азия	8	0	1891454121
35	Юго-Восточная Азия	11	0	641775797

Рисунок 2 – Таблиця «регіон»

Основна робота буде ввестися в таблиці «доп_інф_о_державі», а в таблиці «регіон» лише відображатимуть результат роботи тригера. Так що необхідно зайти в налаштування таблиці «доп_інф_о_державі та перейти до розділу Triggers, що показано на рис.3

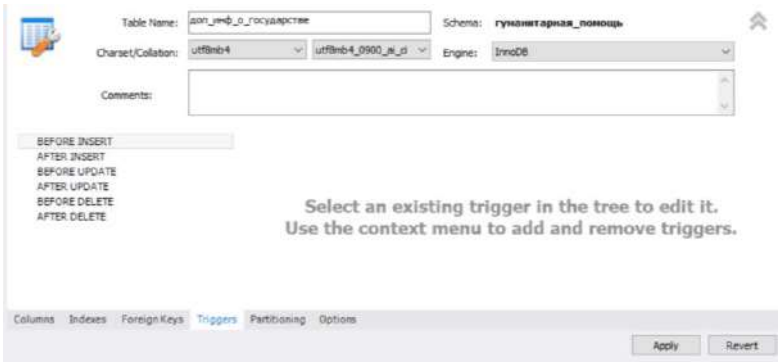


Рисунок 3 – Розділ Triggers таблиці «доп_інф_про_державі»

Також на рис.3 видно, що можна створювати шість різних видів тригерів. У цьому випадку нас цікавить тригер BEFORE INSERT. Після вибору виду тригера відкривається редактор, у якому міститься оголошення тригера, і навіть початок і кінець тіла тригера, що можна побачити на рис.4

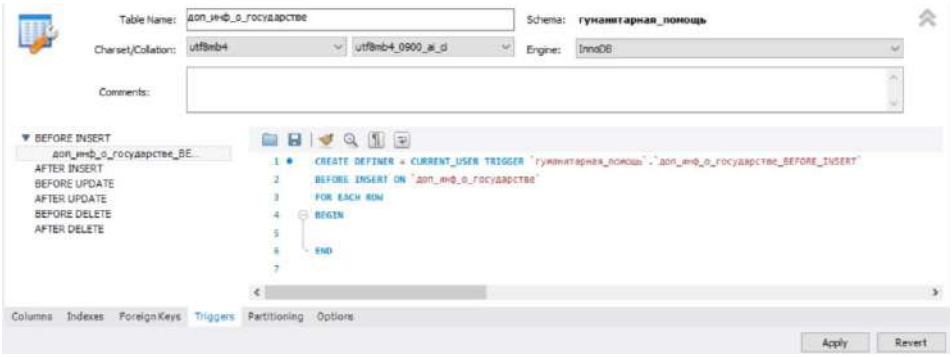


Рисунок 4– Заготівля для тригера

Тепер формуємо тіло тригера. Нам необхідно, щоб за наявності збройного конфлікту у держави, у регіоні, що містить цю державу, збільшувалася кількість збройних конфліктів. Реалізацію тіла тригера можна побачити на рис.

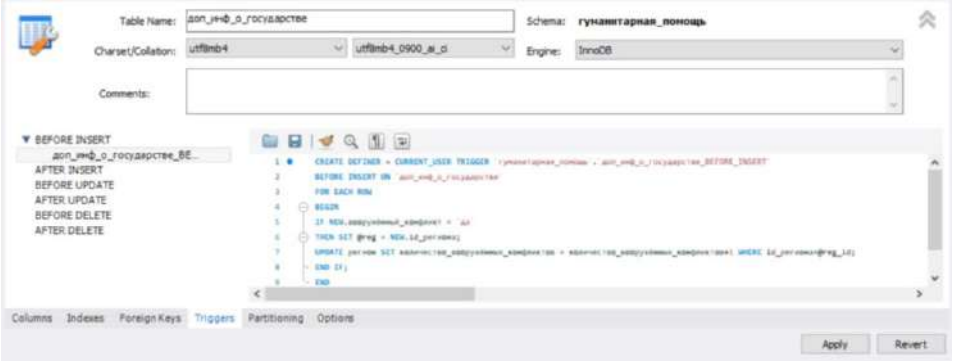


Рисунок 5 – Готовий тригер

Розберемо тіло тригера. Воно починається з оператора умови IF, що якщо поле «озброєний_конфлікт» дорівнюватиме значенням «так», а це може приймати значення так/ні, то будуть виконуватися такі дії поки блок IF не закінчиться ключовою фразою END IF. Тепер задаються дії, що йдуть після умови за допомогою оператора THEN. Створюється тимчасова змінна @id_reg (всі тимчасові змінні починаються зі знака @), до якої заноситься нове значення поля «id_регіону» нашої таблиці, яке має зв'язок «1:1» із таблицею «регіон» за допомогою оператора NEW. Далі ми вказуємо, що потрібно оновити таблицю «регіон» за допомогою оператора UPDATE та встановити нове значення поля «кількість_збройних_конфліктів», як «кількість_збройних_конфліктів»+1 за допомогою оператора SET там, де збігаються коди регіонів наших таблиць.

Тепер потрібно натиснути кнопку Apply, після чого висвітиться вікно, що містить наш тригер, написаний SQL-запитами, що продемонстровано на рис.6

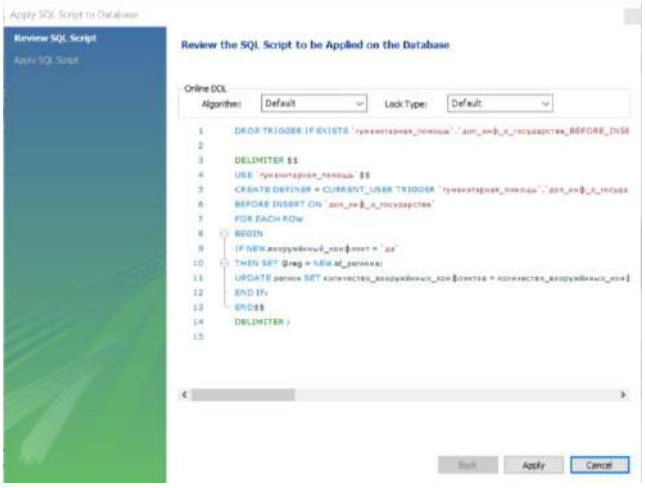


Рисунок 6 – Тригер, що створюється у вигляді SQL-запитів

Після створення тригера, переходимо до таблиці «доп_інф_о_державі» та прописуємо запит SHOW TRIGGERS, який продемонструє всі тригери даної таблиці.

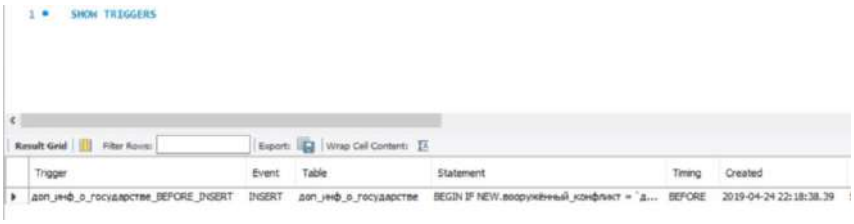


Рисунок 7 – Список тригерів таблиці «доп_інф_про_державі»

Як бачимо на рис.7, у таблиці є тригер, спрацьовує до вставки (BEFORE INSERT) даних у таблицю.

Тепер перевіримо наш тригер на коректність. Для цього виберемо з таблиці рядок, у якого значення поля «озброєний_конфлікт» дорівнює “ ” і змінимо його на “так”. Результат виконаних дій наведено на рис.8

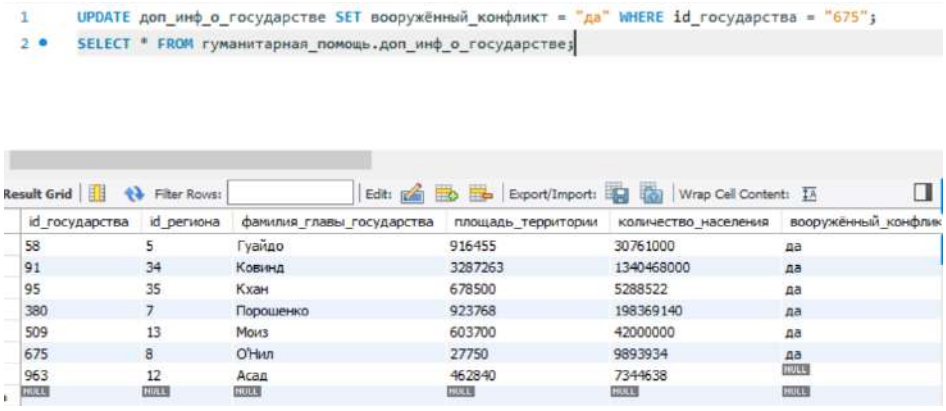


Рисунок 8 – Таблиця «доп_інф_о_державі» зі зміненим значенням поля «озброєний_конфлікт» у рядку зі значенням поля «id_держави»=675

Перейдемо до таблиці «регіон» та оновимо її, після чого побачимо, що у рядка з «id-регіону»=8 змінилося на одиницю значення поля «кількість_збройних_конфліктів», що продемонстровано на рис.9, отже, розроблений нами тригер працює коректно.

```
1 •  SELECT * FROM гуманитарная_помощь.регионы;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id_региона	название_региона	количество_государств	количество_вооружённых_конфликтов	количество_на
4	Северная Америка	2	0	354830825
5	Южная Америка	15	0	387489196
7	Центральная и Восточная Европа	44	0	145800000
8	Австралия и Океания	17	1	40117432
11	Западная Африка	16	0	362201579
12	Ближний Восток	19	0	436005542
13	Центральная Америка	21	0	89503839
34	Южная Азия	8	0	1891454121
35	Юго-Восточная Азия	11	0	641775797

Рисунок 9 – Таблиця «регіон» зі значенням поля «кількість_збройних_конфліктів», змінених тригером

2. Створення простого тригера за допомогою SQL-запитів

   Для баз даних тригери, переважно, створюються з допомогою SQL-запросов. Як приклад, буде створено запит, зворотний вже розробленому. Тобто після зміни значення поля «збройний конфлікт» у таблиці «доп_інф_о_державі» з “так” на “ні”/“ ”, у таблиці «регіон» значення поля «кількість_збройних_конфліктів» буде зменшувати на 1.

```
1   DELIMITER //
2   CREATE TRIGGER after_доп_инф_о_государстве_delete
3   AFTER DELETE ON доп_инф_о_государстве
4   FOR EACH ROW
5   BEGIN
6   IF вооружённый_конфликт = "да"
7   THEN SET @reg_id = id_региона;
8   UPDATE регион SET количество_вооружённых_конфликтов = количество_вооружённых_конфликтов-1 WHERE id_региона=@reg_id;
9   END IF;
10  END
11
```

Рисунок 10 – Тригер на зміну даних у таблиці «регіон» після видалення їх з таблиці «дод_інф_про_державі»

У SQL-запиті на рис.10, тригер починається з ключового слова DELIMITER, після якого ставиться подвійний сліш (//). Далі за допомогою оператора CREATE TRIGGER створюється тригер і даємо йому назву aftrer_ доп_інф_про_державі _delete. Після цього всі дії аналогічні тим, які робилися в

попередньому прикладі, тільки замість інкрементування буде декрементування значення поля «кількість_збройних_конфліктів» таблиці «регіон».

Перевіряємо кількість тригерів у таблиці за допомогою оператора SHOW TRIGGER, що наведено на рис.11

Рисунок 11 – Тригери таблиці «доп_інф_про_державі»

Тепер перевіримо працездатність тригера, повернувши полю, значення якого змінювалося, первісне значення, що показано на рис.12

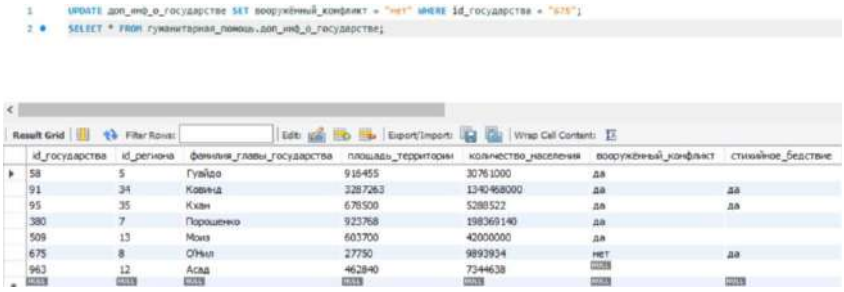


Рисунок 12 –Таблиця «доп_інф_про_державі» з новозміненим значенням поля «озброєний_конфлікт» біля рядка зі значенням поля «id_держави»=675

І робимо запит у таблиці «регіон», після чого значення поля «кількість_збройних_конфліктів» у рядка зі значенням поля «id_регіона»=8 зменшиться на одиницю, що показано на рис.13

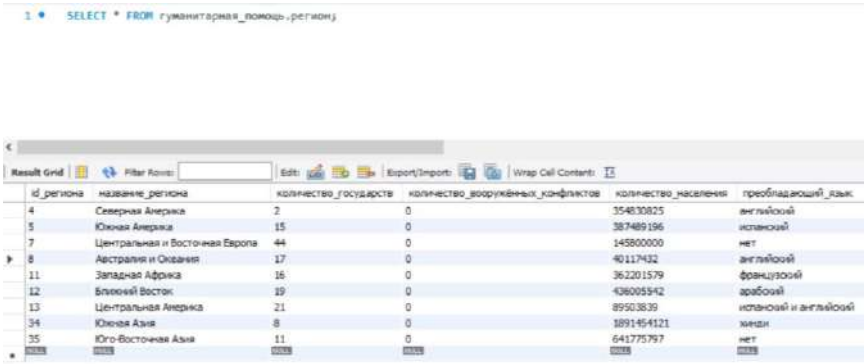


Рисунок13 - Таблиця «регіон» зі значенням поля «кількість_збройних_конфліктів», знову змінених тригером

Завдання

Для таблиць бази даних, що розробляється, реалізувати тригери (2-4 шт. різних типів) за допомогою інструментів MySQL Workbench і за допомогою SQL-запитів

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Організація баз даних : навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2- ге вид. виправ. і доп. – Одесса : Фенікс, 2019. – 246 с.
2. Ярцев В. П. Організація баз даних та знань : навч. посібник / Ярцев В. П. – Київ : ДУТ, 2018. – 214 с.
3. Трофименко О. Г. Сучасні інформаційні та комунікаційні системи та технології : метод. вказівки для практ. занять / Трофименко О. Г., Логінова Н. І. – Одесса : ВЦ НУ «ОЮА», 2016. – 120 с.
4. Трофименко О. Г. Бази даних: створення та опрацювання баз даних : методичні вказівки до лабораторних, практичних занять та самостійної роботи студентів / Трофименко О. Г., Буката Л. М., Прокоп Ю. В. – Одесса, 2016. – 96 с.
5. Трофименко О. Г. СУБД ACCESS створення та опрацювання : навч. посібник / Трофименко О. Г. Буката, Л. М. – Одесса, 2016. – 226 с.
6. Організація баз даних та знань : лабораторний практикум для студентів напряму 050101 «Комп’ютерні науки» ден. та заоч. форм навч. / уклад.: О. М. Мякшило та ін. – Київ : НУХТ, 2015. – 86 с.
7. Берко А. Ю. Системи баз даних та знань. Книга 2. Системи управління базами даних та знань : навч. посібник / Берко А. Ю., Верес О. М., Пасічник В. В. – Львів : Магнолія-2006, 2012. – 584 с.
8. Гайна Г. А. Основи проектування баз даних : навч. посібник / Гайна Г. А. – Київ : Кондор, 2008. – 200 с.
9. Завадський І. О. Основи баз даних : навч. посібник / Завадський І. О. – Київ : Видавець І. О. Завадський, 2011. – 192 с.
10. Зарицька О. Л. Бази даних та інформаційні системи : метод. посібник / Зарицька О. Л. – Житомир : Вид- во ЖДУ ім. І. Франка, 2009. – 132 с.

Зміст

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

з курсу: «Організація та проектування баз даних»

для студентів спеціальності

123 «Комп’ютерна інженерія»

Укладачі:

ФІЛОНЕНКО Алевтина Михайлівна

ЧЕРНУШЕНКО Радислав Володимирович

БРЕЧКО Вероника Олександрівна

ЧЕРНИХ Олена Петрівна

Відповідальний за випуск

Роботу до друку рекомендував

В авторській редакції

План 2024 р., поз. 173

Підп. до друку Гарнітура Times New Roman.

Видавничий центр НТУ «ХПІ»,

вул. Кирпичова, 2, м. Харків, 61002

Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.

Електронна версія