МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЦЕНТРАЛЬНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (1 частина) для студентів денної та заочної форми навчання за спеціальностями F3 «Комп'ютерні науки», F7 «Комп'ютерна інженерія», F5 «Кібербезпека та захист інформації», G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка»

ЗАТВЕРДЖЕНО
На засіданні кафедри кібербезпеки та програмного забезпечення,
протокол № 10 від 25.03.2025 року

КРОПИВНИЦЬКИЙ
2025

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (1 частина) для студентів денної та заочної форми навчання за спеціальностями F3 «Комп'ютерні науки», F7 «Комп'ютерна інженерія», F5 «Кібербезпека та захист інформації», G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка» / уклад. В.В. Босько, Л.В. Константинова — Кропивницький: ЦНТУ, 2025. — 85 с.

Укладач: Босько В. В., Константинова Л.В.

Рецензенти:

Смірнов О. А.	доктор технічних наук, професор, професор кафедри кібербезпеки та програмного забезпечення Центральноукраїнського національного технічного університету;
Мацуй А. М.	доктор технічних наук, професор, доцент кафедри автоматизації виробничих процесів Центральноукраїнського національного технічного університету, академік Академії технічних наук України, член-кореспондент Академії Прикладних Наук

Вступ

Для вивчення дисципліни «Бази даних» пропонується виконання завдань, що представлено в цих методичних рекомендаціях. Завдяки теоретичному та практичному матеріалу користувач може ознайомитись з роботою з базами даних (БД) за допомогою системи керування базами даних (СКБД) MySQL.

Методичні рекомендації розраховані на студентів денної та заочної форм навчання, що вивчають навчальну дисципліну «Бази даних». Виконуючи завдання з лабораторних робіт, що представлено в рекомендаціях, користувачі зможуть закріпити теоретичний матеріал з наступних тем (розглянуто в таблиці 1).

Таблиця 1 - Теми лабораторних робіт

Теми лабораторних робіт	
1. Застосування DDL для роботи з БД	
2. Маніпулювання даними в БД за допомогою DML	
3. Застосування DQL при вибірці даних	
4. Керування базами даних за допомогою DCL	
5. Транзакції у роботі з БД	
6. Створення міжтабличних зв'язків за допомогою SQL, підтримка цілісності БД	
7. Типи з'єднань в MySQL	

До кожної теми пропонується виконання декількох завдань. Практичні приклади та ілюстрації, що представлено в рекомендаціях, полегшують засвоєння викладеного матеріалу з дисципліни «Бази даних». Розглядається робота з реляційними базами даних за допомогою запитів, основні прийоми роботи з БД за допомогою мови SQL, а саме, застосування мови визначення даних (DDL), мови маніпулювання даними (DML), вибірка

даних. Розглядаються прийоми управління доступом користувачів до бази даних та команди мови управління транзакціями (TCL). Пропонується також навчитись керувати зв’язками між таблицями, застосовувати первинні та зовнішні ключі, накладати обмеження на введення, видалення, зміну даних. Розглядаються також практичні завдання з різними типами з’єднань при вибірці даних, застосування агрегатних функцій та впорядкування даних.

Навчальна дисципліна «Бази даних» - двосеместрова, форма підсумкового контролю - залік й іспит відповідно.

У випадку відвідування студентом всіх лекцій, лабораторних занять, виконання і захисту завдань з самостійної роботи у встановлений термін проходження контролю, то отримується максимальна кількість балів з дисципліни згідно шкали оцінювання, що в таблиці 2.

У випадку виконання та захисту лабораторних робіт після встановленого терміну, одержані бали повинні перераховуватись з відповідним коефіцієнтом: для самостійної роботи студента -0,3; лабораторної роботи -0,7.

Таблиця 2 - Шкала оцінювання: національна та ECTS

Сума балів за всі види навчальної діяльності	Оцінка ECTS	Оцінка за національною шкалою
		для екзамену, курсової роботи
90 – 100	**A**	відмінно
82-89	**B**	добре
74-81	**C**	
64-73	**D**	задовільно
60-63	**E**	
35-59	**FX**	незадовільно з можливістю повторного складання
0-34	**F**	незадовільно з обов’язковим повторним вивченням дисципліни

Ознайомившись з теоретичним матеріалом та визначившись з основними поняттями [1], студент може приступити до виконання лабораторних робіт.

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

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

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

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

Студенту пропонується обрати та затвердити тему (предметну область), з якою буде працювати. Всі представлені завдання в рекомендаціях будуть пов’язані з цією темою. Перелік всіх тем дивитись в додатку А. Окрім виконання всіх завдань до всіх лабораторних робіт також потрібно відповісти на запитання, що знаходяться в кінці кожної роботи. Звіт обов’язково повинен містити хід виконання завдань, а також графічні матеріали, що підтверджують виконання цих завдань. Приклад оформлення звіту дивитись в додатку В.

Для роботи з БД необхідно обрати СКБД.

Як СКБД, пропонується застосовувати MySQL - вільну систему керування реляційними базами даних. **MySQL** - компактний

багатонитковий сервер БД з гарними характеристиками щодо швидкодії, простоти застосування та стійкості. Компіляція вихідних кодів можлива на різних платформах. MySQL входить до складу серверів WAMP, AppServ, LAMP та в портативні зборки серверів Денвер, XAMPP, VertrigoServ та ін. Зазвичай MySQL використовується як сервер, до якого звертаються локальні або віддалені клієнти, проте до дистрибутиву входить бібліотека внутрішнього сервера, що дозволяє включати MySQL в автономні програми. (Бажано встановити збірку).

Крім того, необхідні знання команд **SQL** (Structured Query Language – мова структурованих запитів).

Команди SQL поділяються на такі групи:

- Команди мови визначення даних - DDL (Data Definition Language). Ці SQL команди можна використовувати для створення, зміни та видалення різних об'єктів бази даних.
- Команди мови маніпулювання даними - DML (Data Manipulation Language). Ці SQL команди дозволяють користувачеві переміщати дані в базу даних і з неї.
- Команда мови запитів – DQL (Data Query Language). Вибірка даних.
- Команди мови управління даними - DCL (Data Control Language). За допомогою цих SQL команд можна управляти доступом користувачів до бази даних і використовувати конкретні дані (таблиці, представлення і т.д.).
- Команди мови управління транзакціями - TCL (Transaction Control Language). Ці SQL команди дозволяють визначити результат транзакції.

В Інтернеті існує велика кількість продуктів для розробки та адміністрування баз даних MySQL. Серед них є наступні, що часто застосовуються:

- Workbench,
- PHPMyAdmin,
- Navicat,

- HeidiSQL,
- EMS SQL Manager для MySQL,
- SQLyog…

Для виконання лабораторних робіт достатньо ознайомитись з панеллю керування phpMyAdmin. Хоча, якщо студенту зручніше працювати з іншим засобом, це не заборонено.

PhpMyAdmin - це програмне забезпечення, написане на PHP, яке забезпечує повноцінну, у тому числі віддалену, роботу з базами даних MySQL через браузер (входить до складу багатьох серверів для роботи з БД).

Щоб вправно працювати з MySQL та phpMyAdmin зручно зразу встановити пакет програм (наприклад, Open Server, XAMPP, Denwer та інші).

Open Server - це портативний локальний WAMP/WNMP сервер, що має декілька компонентів для роботи та дозволяє легко запускати та тестувати веб-додатки на Windows. Він включає такі компоненти, як Apache, PHP, MySQL, phpMyAdmin та інші (https://ospanel.io/download/).

Про програму MEMP, як її встановити та коротко про введення до MySQL: https://www.youtube.com/watch?v=Et6uzla8aK0.

Про phpMyAdmin: https://www.youtube.com/watch?v=BE1ajZ5S7hE.

SQLFiddle - це веб-сервіс, де ви можете налаштувати і працювати з невеликими прикладами SQL в різних системах (PostgreSQL, Oracle, MySQL і т. д.) без встановлення додаткового ПЗ (http://sqlfiddle.com/about.html).

Лабораторна робота №1

Тема: Застосування DDL для роботи з БД

Мета: Застосовуючи DDL оператори CREATE, DROP, ALTER навчитися створювати та визначати, видаляти та змінювати об'єкти БД

Завдання:

1. Створити нову базу даних за допомогою DDL.
2. За допомогою DDL створіть пробну таблицю з визначеним ім'ям та декількома полями різних типів та розмірів.
3. За допомогою DDL-оператора змініть структуру існуючої таблиці. Додайте нове поле типу INTEGER, збільшіть розмір існуючого поля, видаліть непотрібне поле.
4. Створіть індекс у таблиці БД.
5. Видаліть створений раніше індекс.
6. Видаліть непотрібну таблицю із своєї БД за допомогою DDL-оператору.
7. Створіть 2 головні пусті таблиці для своєї БД так, щоб для двох полів не можна було б встановити невизначене значення, встановіть первинні ключі та інші обмеження для таблиць, застосуйте поле з автоінкрементом.

Теоретичні відомості

Для зміни структури БД в SQL передбачено **DDL** (Data Definition Language) - мову визначення даних. За допомогою операторів DDL можливо наступне:

- Створити нову БД;
- Визначити структуру нової таблиці та створити цю таблицю;
- Видалити існуючу таблицю;
- Змінити визначення існуючої таблиці;
- Визначити представлення даних;

- Забезпечити умови безпеки БД;

- Створити індекси для доступа до таблиць;

- Керувати розміщенням даних на пристроях зберігання.

DDL базується на трьох командах SQL:

- CREATE-дозволяє визначити та створити об’єкт БД;
- DROP- застосовується для видалення існуючого об’єкту БД;
- ALTER -за допомогою якого можна змінити визначення об’єкта БД.

CREATE DATABASE – команда для створення бази даних.

Синтаксис:

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

db_name - ім'я БД;

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

CHARACTER SET, COLLATE - використовується для завдання стандартного кодування таблиці й порядку сортування.

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

CREATE DATABASE `my_db1`

або

CREATE DATABASE `my_db1` CHARACTER SET utf8 COLLATE utf8_general_ci;

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

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

- Максимальна довжина імені в символах не повинна перевищувати 64;

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

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

Створення БД за допомогою PhpMyAdmin:

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

На рисунку 1.1 зображено створення БД з назвою “my_db”:

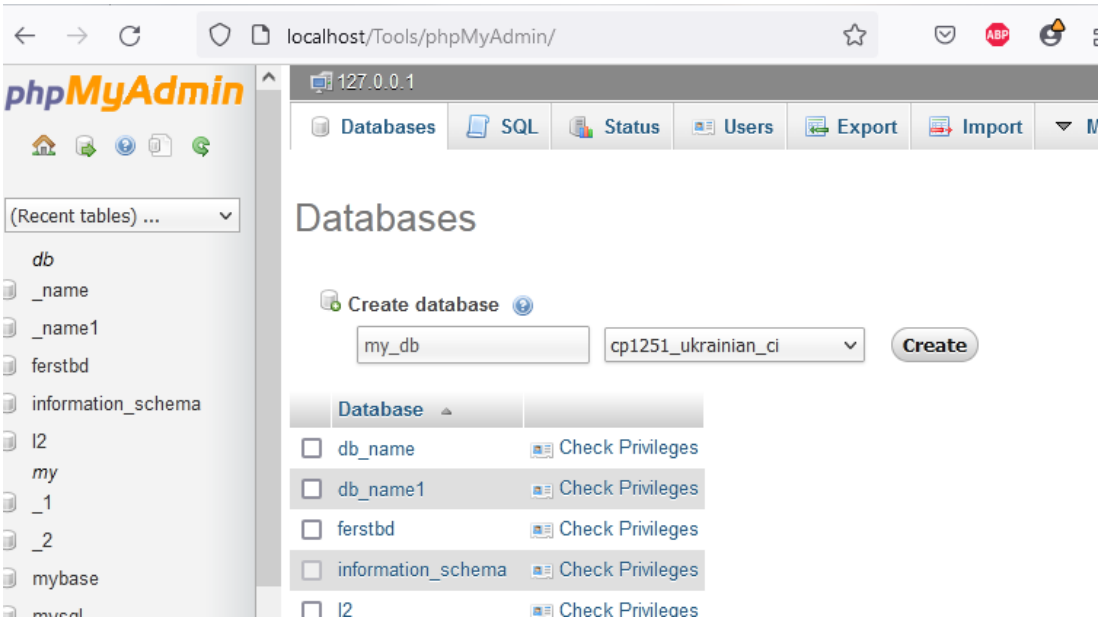


Рисунок 1.1 - Створення нової БД за допомогою phpMyAdmin

Створення таблиці в БД проводиться командою **CREATE TABLE**.

Синтаксис:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name1

[(create_definition,...)]

[table_options] [select_statement]

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

TEMPORARY - використовується для створення таблиці, що є тимчасовою протягом поточного сценарію. По закінченню роботи таблиця

видаляється. Вказана можливість з'явилася в 3.23 версії MySQL. В MySQL 4.0.2 для створення тимчасових таблиць також потрібні привілеї створення тимчасових таблиць.

IF NOT EXISTS - якщо цей параметр зазначено та проводиться спроба створити таблицю, що вже існує у поточній БД, то таблиця створена не буде й повідомлення про помилку не з'явиться. В іншому випадку таблиця також створена не буде, але команда викличе помилку. **create_definition** - визначає внутрішню структуру таблиці, що створюється (назви, типи полів, ключі (якщо є), індекси тощо).

Можливі синтаксиси **create_definition:**

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

PRIMARY KEY (index_col_name1,...)

KEY [index_name] (index_col_name1,...)

INDEX [index_name] (index_col_name1,...)

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

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

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

CHECK (expr)

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

type - задає тип даних для певного стовпця.

Можливі значення параметра type:

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

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

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

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

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

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

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

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

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

Означає те саме що і для INT

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

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

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

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

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

Означає те саме що і для DOUBLE

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

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

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

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

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

Це означає рядок певної довжини, при зберіганні завжди дописується прогалинами в кінці рядка до потрібного розміру. Діапазон аргументу length встановлено від 0 до 255 символів. Проміжки, що в кінці видаляються, якщо здійснюється виведення значення.

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

Означає рядок змінної довжини.

- **DATE**

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

- **TIME**

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

- **TIMESTAMP**

Означає часову мітку. Формат виведення значень в MySQL TIMESTAMP: YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD та YYMMDD.

- **DATETIME**

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

- **TINYBLOB**

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

- **BLOB**

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

- **MEDIUMBLOB**

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

- **LONGBLOB**

Теж стовпець типу BLOB, але має максимальну довжину 4294967295 (2 ^ 32 - 1) символів.

- **TINYTEXT**

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

- **TEXT**

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

- **MEDIUMTEXT**

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

- **LONGTEXT**

Теж стовпець типу TEXT, але має максимальну довжину 4294967295 (2 ^ 32 - 1) символів.

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

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

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

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

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

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

[AUTO_INCREMENT] - При введенні нового запису в таблицю поле з цим атрибутом автоматично отримає числове значення, більше самого великого значення для цього поля в поточний момент часу на 1.

Приклад 1 створює таблицю користувачів з 3 полями, де перше поле - унікальний ідентифікатор запису, друге поле - ім'я користувача, а третє поле - його вік:

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

Приклад 2 створює порожню таблицю STUDENTS:

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

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

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

Приклад 3:

```
ALTER TABLE STUDENTS
```

```
ADD COURS INTEGER,
SPEC CHAR (10);
```

До таблиці STUDENTS додаються два поля для зберігання інформації про курс та спеціальність студента.

Для видалення таблиці використовують команду DROP TABLE. Таблица з інформацією не може видалятись.Таблиця видаляється, якщо вона пуста.

Приклад 4:

```
…DROP TABLE STUDENTS;
```

Виконується видалення пустої таблиці STUDENTS.

Індексом називають впорядкований список полів чи груп полів в таблиці. Індекси - це корисний інструмент, який широко застосовується у всіх СКБД. Якщо створюється індекс у полі, БД запам’ятовує відповідний порядок всіх значень даного поля в області пам’яті. Таблиця, для якої створюється індекс повинна вже існувати і зберігати імена індексованих полів, при цьому ім’я індексів не може бути використано для чогось іншого в БД, а SQL сама вирішує коли він необхідний для роботи та користується ним автоматично.

Приклад 5 команда для створення індексу за полем, яке зберігає прізвище студента:

```
…CREATE INDEX SPRIIDX ON STUDENTS (SPRI);…
```

Приклад 6 для видалення створеного індексу за прізвищем студента, можна скористатись командою:

```
…DROP INDEX SPRIIDX
ON STUDENTS;…
```

Обмеження даних – це частина визначень таблиці, яка обмежує значення, які допускаються до введення в поля таблиці. Обмеження можна вказувати, коли створюється чи змінюється таблиця. Існують два основних види обмежень. Обмеження поля та обмеження таблиці. Обмеження поля ставлять у кінець фрагмента команди, яка оголошує його ім’я після типу даних. Обмеження таблиці ставлять у кінець оголошення імені таблиці. NOT NULL-оберігає поле від порожніх значень.

Кожна таблиця повинна вміщувати одне або декілька полів, що однозначно визначають кожен запис в таблиці. Такі поля називають **первинним ключовим полем** таблиці (PRIMARE KEY).

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

Приклад 7:

```
CREATE TABLE YSP
(UNUM INTEGER NOT NULL PRIMARY KEY,
OCENKA INTEGER,
UDATE DATE,
SNUM INTEGER NOT NULL,
PNUM INTEGER NOT NULL,
UNIQUE (SNUM, PNUM));
```

На рисунку 2 видно, як створити таблицю з прикладу.

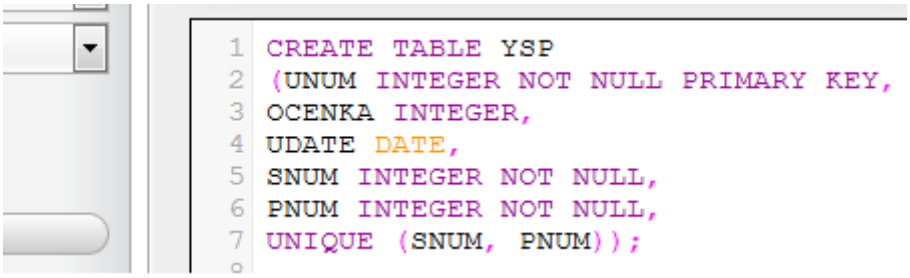


Рисунок 1.2 – Запит на створення таблиці

Таблиця, що успішно створилась, з'явилась у списку таблиць БД.

Якщо необхідно перейменувати поле таблиці використовуємо:

ALTER TABLE name_of_table CHANGE old_column_name new_column_name Тип(розмір);

```
…ALTER TABLE STUDENTS CHANGE SPB SPOB VARCHAR(20);…
```

Поле з ім'ям SPB зміниться на SPOB.

Приклад створення БД та таблиці у PHPMyAdmin просто і з поясненнями: https://www.youtube.com/watch?v=14XVx3m9t4E

Контрольні запитання до лабораторної роботи 1:

1. **Що включає в себе поняття БД?**
2. **Що собою представляє СКБД?**
3. **Який основний об’єкт реляційних БД?**
4. **Що представляє собою SQL?**
5. **На які підгрупи команд поділяється SQL?**
6. **Які можливості DDL?**
7. **На яких командах базується DDL?**
8. **За допомогою якої команди можна створити БД?**
9. **За допомогою якої команди можна створити таблицю?**
10. **За допомогою якої команди можна видалити таблицю?**
11. **За допомогою якої команди можна змінити визначення об’єкта БД?**
12. **Що означає команда CREATE TABLE?**
13. **Що означає команда ALTER TABLE?**
14. **Що означає команда DROP TABLE?**
15. **Яке поле таблиці називають ключовим?**
16. **Що називають індексом?**
17. **Що означає обмеження даних?**
18. **Що означає обмеження NOT NULL у кінці оголошення імені таблиці?**
19. **Як переіменувати поле таблиці?**

Лабораторна робота №2

Тема: Маніпулювання даними в БД за допомогою DML

Мета: Застосовуючи DML оператори INSERT, UPDATE, DELETE навчитися модифікувати, видаляти дані, вносити нову інформацію в БД

Завдання:

1. Побудувати запит (DML оператори), завдяки якому можна було б вносити нові записи до одної з Ваших таблиць. Два поля таблиці повинні заповнюватись певною інформацією, третє поле повинно мати значення NULL, а останні - за замовчуванням.
2. Запит на видалення зі спеціально для цього створеної таблиці всіх записів за допомогою SQL.
3. Запитом внесіть дані в наступну таблицю БД (5 записів). Вилучіть з таблиці ті записи, які у конкретному полі мають певне значення за допомогою DML.
4. Створіть запит на зміну значень де-якого поля на «пусто», яким у полі Дата (наприклад) відповідає значення 01.01.2024.
5. Зменшити за допомогою запиту (DML) значення числового поля на 25% в таблиці, якщо, його значення, наприклад, більше або дорівнює 180376.
6. Заповніть всі таблиці вашої БД інформацією (10 і більше рядків).

Теоретичні відомості

Запити на зміну використовуються для додавання, вилучення й поновлення записів, а також для збереження результуючого набору записів запиту у вигляді таблиці. Вносити, модифікувати та видаляти значення з таблиць відбувається за допомогою трьох команд **DML** (мова маніпулювання даними) **- INSERT, UPDATE, DELETE.**

Додавання інформації в таблицю

В SQL всі записи в таблицю вводяться за допомогою команди модифікації INSERT. Ім'я таблиці, в яку відбувається вставка, повинно бути попередньо визначене, а кожне значення, що вводиться повинне співпадати з типом даних поля, в який воно вставляється.

Приклад 1 - для внесення запису в таблицю з викладачами TEACHERS, можливо скористатись виразом такого виду:

```
...INSERT INTO TEACHERS
VALUES ('4006', 'Федченко', 'Світлана', 'Григорівна', '2019-09-09');…
```

Можна вказати стовпці, в які будуть зберігатись введені значення.

Приклад 2:

```
INSERT INTO TEACHERS (TDATE, TFAM, TIMA)
VALUES ('01-09-1999', 'Федченко', 'Світлана');
```

Для полів, які не вказані, в запиті автоматично встановлюються значення за замовчуванням.

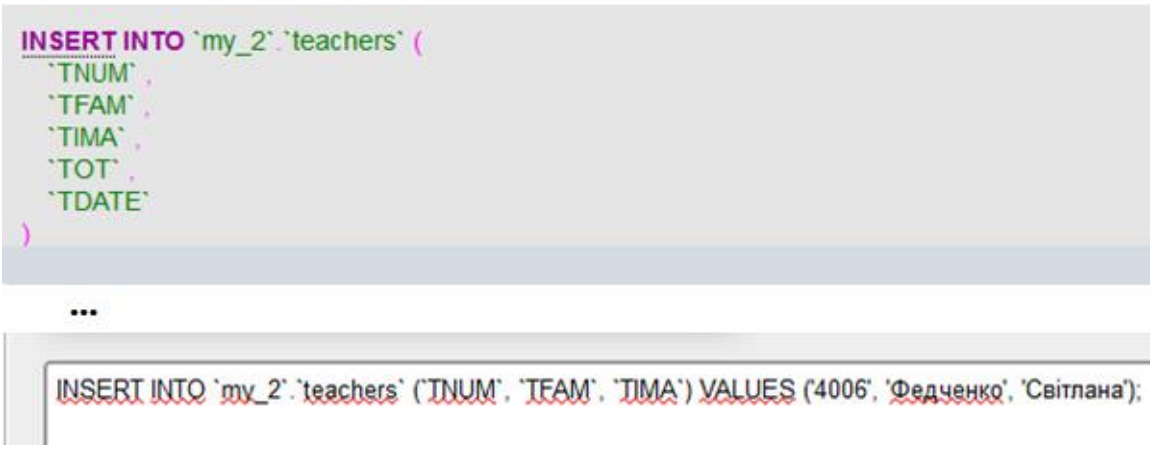


Рисунок 2.1 – Запит на додавання

На рисунку 2.1 отримали запит у режимі SQL, що додає інформацію до таблиці TEACHES БД MY_2.

Є можливість за допомогою команди INSERT отримувати чи вибирати значення з однієї таблиці та поміщати їх в іншу разом з запитом.

Видалення даних

Видалення рядків з таблиці можна здійснити командою модифікації DELETE. Треба зауважить, що команда може видаляти тільки цілі записи таблиці, а не індивідуальні значення деяких полів.

Приклад 3. Для видалення всього вмісту таблиці STUDENTS можливо виконати наступне:

```
…DELETE FROM 'STUDENTS';…
```

Для визначення рядків, які треба видалити за певної умови, застосовують предікат. **Приклад 4** - для видалення інформації, що стосується студента Нагорного можна використати наступну команду:

```
…DELETE FROM 'STUDENTS'
WHERE SNUM=3416;…
```

В якості предікату використовують номер студентського квитка.

Наприклад, на рисунку 2.2 зображена таблиця STUDENTS.

Для видалення таблиці необхідно застосувати SQL-запит, що зображено на рисунку 2.3. Необхідно підтвердити виконання дії, якщо необхідно. Після виконання запиту в повідомленні буде вказано кількість видалених рядків та час виконання операції.

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

Рисунок 2.2 - Таблиця STUDENTS

Рисунок 2.3 – Запит на видалення рядків з таблиці

Про оновлення даних

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

Приклад 5:

```
…UPDATE 'USP'
```

```
SET 'OCENKA'=5;…
```

Приведе до зміни в таблиці USP всіх оцінок на «5».

Для зміни єдиного значення можна застосовувати предікати (рисунок 2.4). Наприклад:

```
…UPDATE 'USP'
SET 'OCENKA'=5
WHERE PNUM=2003;…
```

```
1 UPDATE `students` SET
  `SFAM`='Перший' WHERE
  SNUM=200
```

Рисунок 2.4 – Запит з предікатом

За допомогою UPDATE можна модифікувати дані з декількох полів - SET взмозі визначити будь-яку кількість полів, відокремлених комами.

Модифікувати зразу кілька таблиць однією командою UPDATE не дозволено, тому не можна застосовувати назву таблиці через крапку в іменах полів

У рядку SET команди UPDATE можна застосовувати вирази, розташовуючи їх в списку для того поля, яке необхідно змінити.

Приклад 6:

```
…UPDATE 'STUDENTS'
SET STIP=STIP*3;…
```

Команда збільшує значення стипендії в 3 рази.

До недоліків команди UPDATE можна віднести неможливість посилатися на таблицю, яка задієна в будь-якому підзапиті з команди модифікації. Наприклад, неможливо одною командою виконати таку дію, як модифікація оцінок для студентів, у яких оцінки нижче середньої. Для цього необхідно виконати один запит (запит на вибірку):

```
…SELECT AVG (OCENKA)
FROM USP;…
```

а потім результат цього запиту застосувати для модифікації:

```
…UPDATE USP
SET OCENKA= OCENKA-1
WHERE OCENKA <4.2;…
```

Робота з записами в таблицях за допомогою phpMyAdmin на простих прикладах: https://www.youtube.com/watch?v=A7WyqmNVexs.

Контрольні запитання до лабораторної роботи 2:

1. Що представляє собою DML?

2. Які команди складають DML?

3. Перелічіть, які дії можливі для запитів на зміну?

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

5. Що виконується командою UPDATE у запитах?

6. Чи можливо за допомогою UPDATE виконувати модифікацію даних з декількох полів?

7. Яка команда SQL застосовується в запитах на вилучення значень у таблиці?

8. Які дії виконуються командою INSERT INTO у запитах SQL?

9. Виконання яких дій стає можливим командою DELETE у запитах SQL?

10. Якою командою SQL можливо змінити значення в таблиці?

11. За допомогою якої команди можливо видалити значення з таблиці?

Лабораторна робота № 3

Тема: Застосування DQL при вибірці даних

Мета: Застосовуючи SQL-оператори, та спеціальні засоби навчитися виконувати складну обробку даних за допомогою запитів та вдосконалювати їх виведення. Навчитися виконувати узагальнену групову обробку значень полів за допомогою агрегатних функцій в SQL запитах

Завдання:

1. За допомогою SQL- запитів вивести всі дані з таблиць БД. В інструкції застосовувати необов'язкове скорочення у вигляді символа «зірочка» (*).
2. Створити запит на вибірку значень для двох полів таблиці.
3. Створити SQL- запит для вибірки даних таблиці таким чином, щоб результат не мав дублікатів в окремому полі.
4. З таблиці з даними отримайте інформацію про всі об'єкти (постачальники, клієнти, хворі, студенти і т.д.), яким в полі, наприклад, код або номер відповідає певне значення (наприклад 2003), а в іншому полі - значення більше або дорівнює певному значенню (наприклад>=3).
5. За допомогою агрегатної функції у SQL- запиті підрахуйте кількість записів у таблиці вашої БД, не рахувати пусті значення, але враховувати дублікати.
6. Необхідно знайти максимальне значення збільшеного вдвічі значення поля, наприклад, стипендії або заробітної плати, або ціни товару, або іншого поля зі своєї БД за допомогою SQL- запиту.
7. Зробити вибірку даних з таблиці про студентів (чи інші об'єкти), впорядковуючи їх за розміром стипендії (або інше числове поле) у

порядку спадання, а для студентів, що мають однаковий розмір стипендії в алфавітному порядку їх прізвищ.

8. Зробити вибірку даних з двох таблиць об’єднанням (UNION), щоб було видно з якої таблиці кожне значення (Приклади 8,9).
9. Вивести всю інформацію про об’єкт з визначеним прізвищем, якщо невідомий його номер. Номер отримується з таблиці з даними про об’єкти, за допомогою вкладеного запиту, а потім застосувати результат до таблиці з основними даними (Приклад 11).
10. Зробити вибірку і порахувати кількість об’єктів (клієнтів, працівників чи студентів) яким відповідає значення іншого поля кількість більше 2 (Приклад з Having).

Теоретичні відомості:

Вибірка даних DQL (Data Query Language). Мова запитів DQL найбільш відома користувачам реляційної БД, незважаючи на те, що вона включає одну команду SELECT. Ця команда численними опціями і пропозиціями використовується для формування запитів до реляційної БД.

Синтаксис:

```
SELECT [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
[SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
[HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
  select_expression,...
  [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
  [FROM table_references
    [WHERE where_definition]
    [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
```

[HAVING where_definition]

[ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]

[LIMIT [offset,] rows]

[PROCEDURE procedure_name]

[FOR UPDATE | LOCK IN SHARE MODE]]

Приклад 1:

```
…SELECT SNUM, SFAM, SIMA, SOTCH, STIP
FROM STUDENTS;…
```

SELECT- ключове слово, яке повідомляє БД, що ця команда є запитом.

SNUM, SFAM, SIMA, SOTCH, STIP-список полів з таблиці, які вибираються запитом. Такий запит не впливає на інформацію в таблицях, він тільки показує дані.

FROM STUDENTS – ключове слово, яке супроводжується ім’ям таблиці, яку використовуємо у якості джерела інформації.

Приклад виконання запиту на рисунку 3.1.

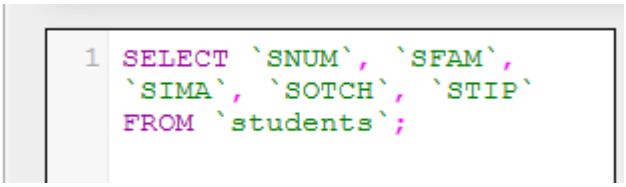


Рисунок 3.1 - Запит на вибірку даних

Результатом такого запиту на вибірку буде таблиця з даними.

Крапка з комою «;»-застосовується у всіх інтерактивних командах SQL для повідомлення БД що команда готова для виконання.

«*» -якщо необхідно отримати кожне поле таблиці.

Приклад 2:

```
…SELECT * FROM STUDENTS;…
```

що призведе до того ж результату, що і попередня команда. Якщо необхідно вивести тільки деякі поля з таблиці, просто необхідно виключити з списку непотрібні поля.

SFAM	SIMA	SOTCH	STIP
Перший	ццццц	ууууу	100
Струмеленко	Сергій	Петрович	100
Зозуленко	Павел	Романович	1000

Рисунок 3.2 - Результат роботи «*» у запиті

Якщо є необхідність уникнути дублювання застосовують DISTINCT-аргумент.

Наприклад, є таблиця з рисунку 3.2.

Приклад 3:

```
…SELECT DISTINCT STIP FROM STUDENTS;…
```

Цю інструкцію застосовують для отримання списку результатів без дублікатів.

На рисунку 3.3 отримали результат виконання такого запиту.

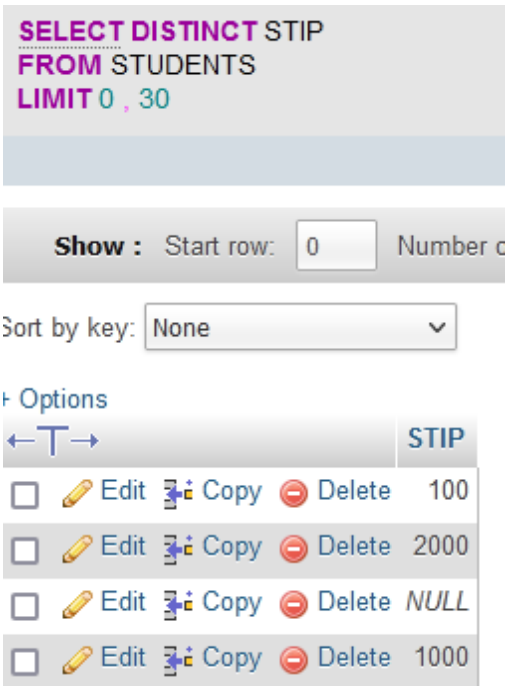


Рисунок 3.3 - Результат запиту на вибірку без дублікатів

Якщо застосувати ALL, то це буде мати протилежний ефект.

WHERE – інструкція команди SELECT, яка дозволяє встановлювати предікати, умова яких можливо буде виконуватись або не виконуватись для будь-якого запису таблиці. Команда отримує тільки ті записи з таблиці, для яких таке твердження буде вірним.

Приклад4:

```
...SELECT SFAM, STIP
FROM STUDENTS
WHERE STIP=45.50;…
```

виведуться тільки ті прізвища та розмір стипендії студентів, для яких у полі STIP буде значення 45.50.

Пов'язуючи предікати з булєвськими операторами (AND, OR, NOT) та реляційними операторами (=, >, <, >=, <=, <>), набагато збільшується можливість вибірки потрібних даних. Також можливо застосовувати спеціальні оператори IN, BETWEEN, LIKE, IS NULL (наприклад, рисунок 3.4).

```
1 SELECT SFAM, STIP
2 FROM STUDENTS
3 WHERE STIP>=100 AND
  SIMA='Сергій';
4
```

Результат:

SFAM	STIP
Струмеленко	100

Рисунок 3.4 – Результат запиту на вибірку з предікатами з булевими та реляційними операторами

Запити здатні виконувати узагальнену групову обробку значень полів, що реалізовується за допомогою агрегатних функцій. Агрегатні функції отримують одиночне значення для всієї групи таблиці. В SQL допускають наступні функції:

COUNT, SUM, AVG, MAX, MIN. Щоб знайти суму всієї отриманої стипендії в таблиці про студентів маємо **приклад 5**:

```
…SELECT SUM(STIP)
FROM STUDENTS;…
```

Функція COUNT застосовується для підрахунку кількості значень у стовпці. **Приклад 6**:

```
…SELECT COUNT(SNUM)
FROM USP;…
```

Якщо з COUNT застосовувати ALL (використовується за замовчанням) – не можна підрахувати значення NULL, але враховуються дублікати, а COUNT з «*» включає записи з NULL та дублікати.

Приклад запитів з агрегатними функціями та їх результати зображено на рисунку 3.5.

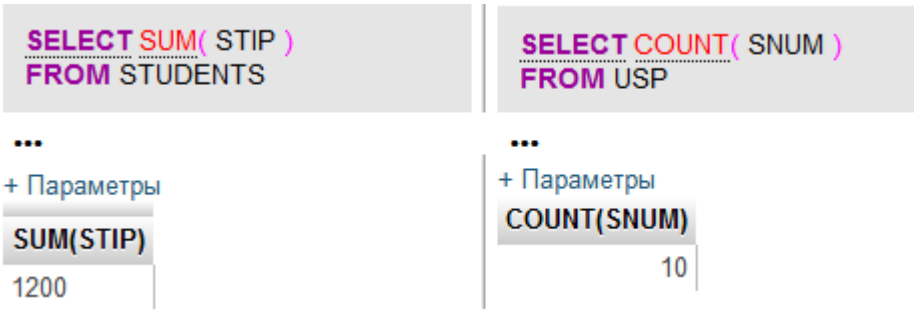


Рисунок 3.5 - Застосування агрегатних функцій у запитах

Для впорядкування виведення полів таблиць SQL має команду ORDER BY, що дозволяє сортувати виведення запиту згідно зі значеннями серед кількості полів. Якщо вказати декілька полів, то стовпці виведення впорядковуються один в середині іншого, при цьому треба визначити зростання (ASC) чи спадання (DESC) для кожного стовпця. **Приклад 7**:

```
…SELECT *
FROM STUDENTS
ORDER BY SFAM ASC;…
```

виведе таблицю з інформацією про студентів в алфавітному порядку прізвищ.

Однотабличні запити – це запити, які в якості джерела даних використовують тільки одну таблицю.

Часто виникає необхідність у виборі інформації з декількох таблиць. Це вирішується за допомогою багатотабличних запитів. Одним із варіантів такого виводу є об’єднання результатів декількох запитів, які виконуються незалежно один від одного.

Об'єднання є механізм, який використовується для об'єднання таблиць всередині оператора. Використовуючи особливий синтаксис, можна об'єднати кілька таблиць таким чином, що буде повертатися один

результат, і це об'єднання буде "на льоту" пов'язувати потрібні рядки з кожної таблиці.

Об'єднання створюється СКБД в разі потреби і зберігається тільки на час виконання запиту.

Для розташування декількох запитів разом та об’єднання їх виведення застосовують UNION, що поєднує виведення двох чи більше SQL запитів в єдиний набір рядків та стовпців (сполучення за допомогою Join розглядається в 7 л.р.).

Приклад 8 - щоб отримати інформацію про учасників конференції ЗВО застосовується запит:

```
SELECT SFAM, SIMA, SOTCH
FROM STUDENTS
WHERE NOT(ISNULL(KONF))
UNION
SELECT TFAM, TIMA, TOTCH
FROM TEACHERS
WHERE NOT(ISNULL(KONF));
```

Зверніть увагу, відсутність «;» після першої інструкції означає, що далі буде ще один або декілька запитів. Якщо необхідно з’єднати два або більше стовпців враховується, що вони повинні бути сумісними для об’єднання. Для кожного з запитів необхідне включення однакової кількості стовпців, у тому ж порядку, та при цьому повинна бути сумісність типів. Не можна застосовувати агрегатні функції в інструкції SELECT запиту на об’єднання. UNION автоматично виключає дублікати рядків з виведення. Можливо застосовувати константи та вирази у інструкції SELECT з UNION.

Результатом роботи запиту з прикладу 8 буде таблиця про учасників конференції, що містить інформацію з двох таблиць, це видно на рисунку 3.6.

SFAM	SIMA	SOTCH
Перший	ццццц	уууууу
Струмеленко	Сергій	Петрович
Терещенко	Анфіса	Петрівна
Степанов	Дмитро	Васильович
Федченко	Світлана	
Іванченко	Олена	Сергіївна
Кочерженко	Вікторія	

Рисунок 3.6 - Результат інструкції SELECT з UNION

Приклад 9:

```
SELECT 'Студент  ', SFAM
FROM STUDENTS
UNION
SELECT 'Викладач  ', TFAM
FROM TEACHERS;
```

Результат на рисунку 3.7.

+ Параметры

Студент	SFAM
Студент	Перший
Студент	Струмеленко
Студент	Зозуленко
Викладач	Терещенко
Викладач	Степанов
Викладач	Федченко
Викладач	Іванченко
Викладач	Кочерженко

Рисунок 3.7 - Результат запиту з UNION

Важливою особливістю запитів SQL є їх спроможність визначати зв'язки між таблицями й виводити інформацію з них в термінах цих зв'язків. Ці операції називаються об'єднанням. У багатотабличних запитах, таблиці, які представлені у вигляді списку в інструкції FROM, відокремлюються одна від одної комами. Предикат запиту може посилатись на будь-який стовпець будь-якої таблиці, та може використовуватись для зв'язку між ними. Тепер виникає необхідність

вживання імен стовпців та таблиць, оскільки в багатотабличному запиті можливе виникнення неоднозначності. Допускається створювати запити, що об'єднують більше двох таблиць. **Приклад 10** - необхідно вивести список оцінок, які виставив той чи інший викладач:

```
…SELECT TEACHERS.TFAM, USP.OCENKA
FROM TEACHERS, PREDMET, USP
WHERE TEACHERS.TNUM=PREDMET.TNUM
AND PREDMET.PNUM=USP.PNUM;…
```

Результат виконання такого запиту може мати такий результат (рисунок 3.8):

```
SELECT TEACHERS.TFAM, USP.OCENKA
FROM TEACHERS, PREDMET, USP
WHERE TEACHERS.TNUM = PREDMET.TNUM
AND PREDMET.PNUM = USP.PNUM
LIMIT 0 , 30
```

…

TFAM	OCENKA
Терещенко	12
Терещенко	0
Терещенко	8
Степанов	9
Степанов	6
Федченко	5
Федченко	9
Федченко	10
Іванченко	7

Рисунок 3.8 - Можливий результат запиту

Запити здатні керувати іншими запитами, це відбувається, якщо розмістити запит всередину іншого предиката, який використовує виведення внутрішнього запиту для встановлення вірного чи невірного значення предиката. **Приклад 11** - потрібно вивести всю інформацію про студента з прізвищем Поляков, якщо невідомий його номер. Номер отримується з таблиці з даними про студентів, а потім застосувати результат до таблиці успішності таким запитом:

```
…SELECT *
FROM USP
WHERE SNUM=
(SELECT SNUM
FROM STUDENTS
```

```
WHERE SFAM='Перший');…
```

Щоб виконати запит SQL спочатку оцінює внутрішній запит (його називають **підзапитом**) всередині речення WHERE. Підзапит повинен вибрати тільки одне поле, а тип даних цього поля повинен співпадати з тим значенням, з яким він порівнюється в предикаті.

Результат виконання запиту з підзапитом з прикладу 11 зображено на рисунку 3.9.

Спочатку виконується пошук номера студента у таблиці STUDENTS, це 1010, потім в таблиці успішності вибирається все потрібне для цього номера, а саме, на цей момент 3 рядки інформації.

```
SELECT *
FROM USP
WHERE SNUM = (

  SELECT SNUM
  FROM STUDENTS
  WHERE SFAM = 'Перший'
```

…

UNUM	OCENKA	UDATE	SNUM	PNUM
1	12	2020-01-01	1010	1
3	0	2020-01-02	1010	1
5	8	2020-01-03	1010	1

Рисунок 3.9 - Запит з підзапитом

EXISTS, ANY, ALL, SOME-спеціальні оператори, які завжди беруть підзапити у якості аргументів.

EXISTS-використовують, щоб вказати предикату на те, щоб виконувати виведення чи не виконувати виведення в підзапиті, при цьому EXISTS дає у якості результату значення ІСТИНА чи БРЕХНЯ. Наприклад, можливо вирішити, отримувати дані з таблиці успішності, якщо в ній присутні незадовільні оцінки. Це реалізується таким чином:

```
…SELECT *
FROM USP
WHERE USP.OCENKA=1
AND EXISTS
(SELECT *
FROM USP
WHERE USP.OCENKA=1);…
```

ANY, ALL та SOME, нагадують EXISTS, але відрізняються тим, що застосовуються разом з реляційними операторами, аналогічно IN в підзапитах. Наприклад:

```
…SELECT *
FROM USP
WHERE OCENKA<>ALL
(SELECT OCENKA
FROM USP
WHERE UDATE=2024-06-10);…
```

В цьому випадку підзапит вибере всі оцінки за 10.06.2024. Після цього основний запит виведе всі записи з оцінкою, яка не співпадає ні з одною з них.

Аналогічний результат можна отримати таким чином:

```
...SELECT *
FROM USP
WHERE NOT OCENKA=ANY
(SELECT OCENKA
FROM USP
WHERE UDATE=2024-06-10);…
```

Якщо запросити значення за допомогою команди ALL, не рівні набору значень, то це теж саме, що визнати факт відсутності цього значення у наборі.

Конструкція GROUP BY

Group by дозволяє визначати підмножину значень і застосовувати агрегатну функцію до цієї множини, дає можливість поєднувати поля й агрегатні функції в єдиній конструкції Select. Наприклад:

```
…SELECT Year, COUNT(NAME) FROM film_list
GROUP BY Year…
```

Такий запит видасть кількість фільмів кожного року.

Оператор HAVING

Оператор HAVING визначає критерії, що використовуються для видалення певних груп з виведення, має призначення, подібне до оператора WHERE, але використовується з агрегатними даними. Наприклад з замовниками:

```
…SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;…
```

Тут ми рахуємо кількість клієнтів-замовників (*customer_id*), групуючи їх за країнами, а потім повертаємо результат, якщо на країну є більше 1 клієнта рисунок 3.10.

COUNT(customer_id)	country
2	UK
2	USA

Рисунок 3.10 – Результат запиту з HAVING

HAVING додали через те, що оператор WHERE не підтримує агрегатні функції.

Оператори GROUP BY та JOIN більш детально буде розглянуто в лабораторній роботі №7.

Контрольні запитання до лабораторної роботи 3:

1. Що означає DQL?

2. Якою командою в запитах SQL здійснюється вибірка даних?

3. Який значок в SQL-запиті допомагає вивести дані всіх полів з таблиці?

4. Який засіб запобігає дублюванню інформації в SQL-запитах?

5. За допомогою якого засобу у SQL-запитах в результаті зберігається дублювання рядків виведення?

6. Що виконується запитом з командою SELECT...FROM...?

7. Що буде виконуватись в запиті, якщо застосовується команда ORDER BY?

8. Яка інструкція команди SELECT, дозволяє встановлювати предікати?

9. Яка функція застосовується для підрахунку кількості значень у стовпці таблиці в SQL-запиті?

10. Для чого застосовують UNION у запитах?

11. Які запити називають підзапитами?

12. Для чого використовують оператор GROUP BY?

13. Що за запити називають багататабличними?

14. Чи відрізняється оператор HAVING від WHERE?

Лабораторна робота № 4

Тема: Керування базами даних за допомогою DCL

Мета: Навчитися керувати доступом до інформації, що знаходиться в БД. Навчитися створювати нові ролі та контролювати розподілом привілеїв між користувачами БД

Завдання:

1. Створити нових користувачів вашої БД (з іменами User1, User2, User3).
2. Надати доступ User1 до всієї інформації вашої БД.
3. Надати доступ User2 тільки перегляд інформації БД.
4. Надати доступ User3 додавання, видалення, зміна даних в таблицях, перегляд інформації БД.
5. Позбавте прав доступу User1: не дозволяйте призначати або видаляти права доступу для інших користувачів.
6. Позбавте прав доступу User3 видалення даних в таблицях.
7. Відобразіть список користувачів БД.
8. Відобразіть привілеї користувачів.

Теоретичні відомості:

MySQL - це програмне забезпечення з відкритим вихідним кодом для управління базами даних, яке допомагає користувачам зберігати, організовувати і здійснювати доступ до інформації. Воно має безліч варіантів тонкої настройки прав доступу до таблиць і баз даних для кожного користувача.

Як тільки користувач починає застосовувати MySQL, йому надається ім’я користувача і пароль. Ці початкові облікові дані дають вам привілеї ‘root-доступу’. Користувач з правами доступу root має повний доступ до всіх баз даних і таблиць всередині цих баз.

DCL (Data Control Language або мова керування даними) - мова управління привілеями. Команди управління даними дозволяють управляти доступом до інформації, що знаходиться всередині БД. Як правило, використовується для створення об'єктів, пов'язаних з доступом до даних, а також служать для контролю над розподілом привілеїв між користувачами. Команди управління даними: **Grant** - додає роль (привілей) користувачу; **Revoke** - команда, яка видаляє роль.

Синтаксис команд GRANT и REVOKE:

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
   ON {tbl_name | * | *.* | db_name.*}
   TO user_name [IDENTIFIED BY [PASSWORD] 'password']
      [, user_name [IDENTIFIED BY 'password'] ...]
   [REQUIRE
      [{SSL| X509}]
      [CIPHER cipher [AND]]
      [ISSUER issuer [AND]]
      [SUBJECT subject]]
   [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
                  MAX_UPDATES_PER_HOUR # |
                  MAX_CONNECTIONS_PER_HOUR #]]
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
   ON {tbl_name | * | *.* | db_name.*}
   FROM user_name [, user_name ...]
```

У випадку, якщо потрібні більш жорсткі обмеження, існують способи створення користувачів з особливими наборами прав доступу.

Приклад 1 спочатку створення нового користувача з консолі MySQL:

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
```

Перевірити користувачів можливо на вкладці Користувачі (або users), як показано на рисунку 4.1.

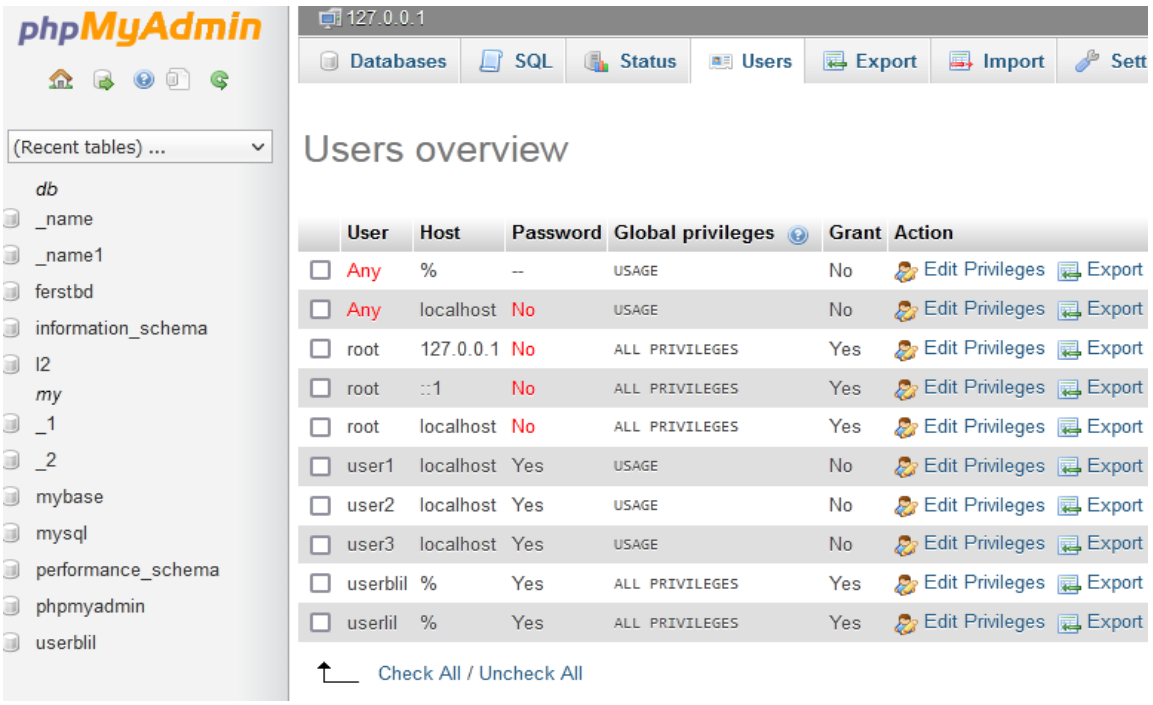


Рисунок 4.1 – Вкладка users

Потім необхідно надати користувачеві доступ до інформації, яка йому буде потрібна:

```
GRANT ALL PRIVILEGES ON *.* TO 'newuser2'@'localhost';
```

Саме ця команда дозволяє користувачеві читати, редагувати, виконувати будь-які дії над усіма базами даних і таблицями. Після завершення налаштування прав доступу нових користувачів, переконайтеся, що ви оновили всі права доступу:FLUSH PRIVILEGES (оновлення всіх прав доступу).

Завдяки командам GRANT й REVOKE дозволяється системним адміністраторам реєструвати та додавати користувачів MySQL; на чотирьох рівнях привілеїв їм надаються права та можливо позбавляти прав.

Про рівні привілеїв

Є **глобальний рівень.** Привілеї глобального рівня зберігаються в табличці mysql.user та стосуються всіх баз даних на зазначеному сервері БД.

Наступний іде **рівень БД.** Цього рівня привілеї стосуються в зазначеній БД всіх табличок. Вони містяться в таких таблицях mysql.db й mysql.host.

Потім іде **рівень таблиці**. Такого рівня привілеї стосуються загалом всіх стовпчиків вказаної таблиці й розміщуються в таблиці з назвою mysql.tables_priv.

І нарешті, **рівень стовпчика.** Такого рівня привілеї містяться в табличці mysql.columns_priv та застосовуються до певних вказаних стовпчиків зазначеної таблиці.

Список деяких з можливих варіантів прав доступу, що надаються користувачам:

ALL PRIVILEGES - це дасть користувачеві MySQL повний доступ до заданої БД (якщо база даних не вказана, то до всіх);

CREATE - дозволяється виконувати створення нових таблиць або баз даних;

DROP - дозволяє видаляти таблиці або бази даних;

DELETE - дозволяється видаляти рядки з таблиць;

INSERT - дозволяється вносити рядки в таблицю;

SELECT - дозволяє використовувати команду Select для читання з баз даних;

UPDATE - дозволить редагувати рядки таблиць;

GRANT OPTION - дозволить призначати або видаляти права доступу для інших користувачів.

Для призначення прав конкретного користувача можна використовувати наступну схему:

GRANT [тип прав] ON [назва бази даних].[Назва таблиці] TO '[ім'я користувача]'@'localhost';

На рисунку 4.2 зображено запит, за допомогою якого, користувачу user1 надаються привілеї повного доступу до всіх таблиць БД my_2.

```
1 GRANT ALL PRIVILEGES ON my_2.* TO
  'user1'@'localhost';
```

Рисунок 4.2 - Запит на повний доступ до всіх таблиць для user1

Перевірити результат запиту можливо на вкладці користувачів (або users), знайшовши користувача необхідно вибрать Редагування привілеїв і впевнитись у виконаних діях (рисунок 4.3), також можливо редагувати потрібне. Також можливо вибрать у вкладці Бази даних (DataBases) необхідну БД та вибрати Check privileges для перевірки привілеїв, наприклад як на рисунку 4.4, бачимо список всіх користувачів, що мають доступ до відповідної БД та їх привілеї.

Кожен раз, коли відбувається зміна прав доступу, необхідно застосовувати команду Flush Privileges.

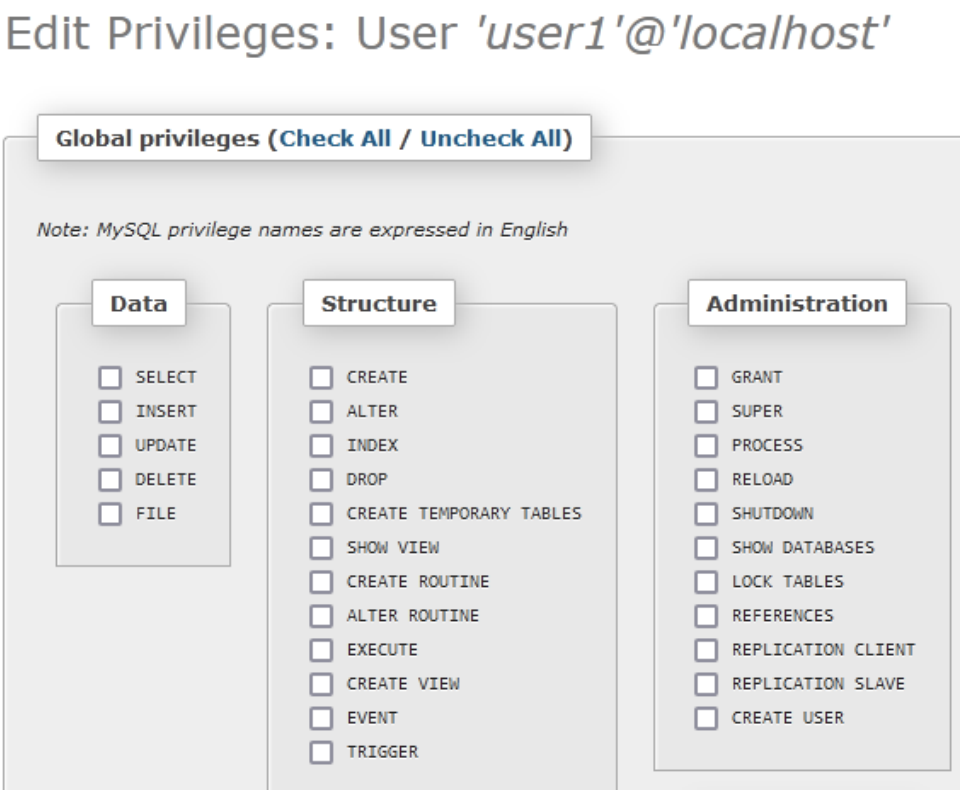
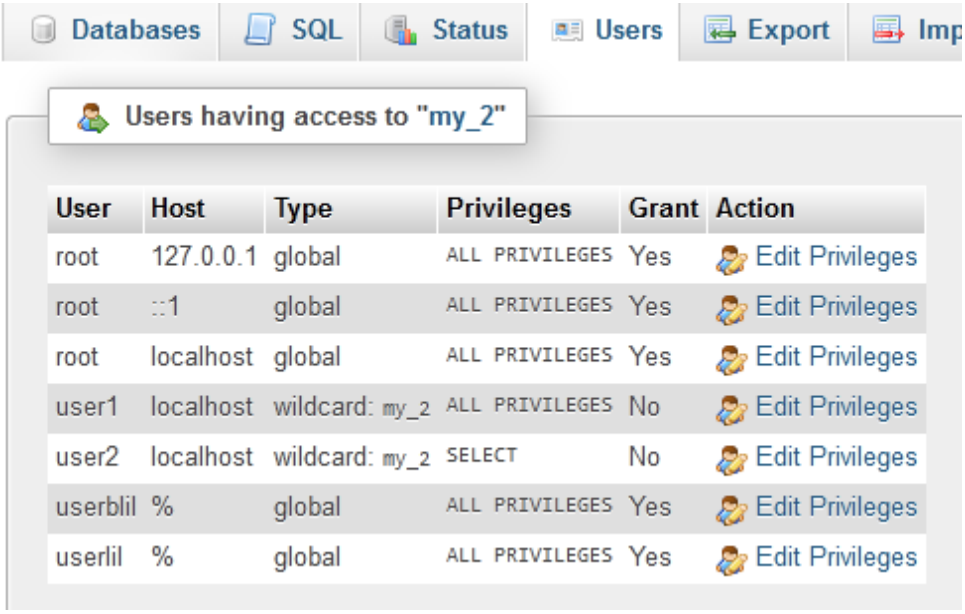


Рисунок 4.3 – Редагування привілеїв для user1



User	Host	Type	Privileges	Grant	Action
root	127.0.0.1	global	ALL PRIVILEGES	Yes	Edit Privileges
root	::1	global	ALL PRIVILEGES	Yes	Edit Privileges
root	localhost	global	ALL PRIVILEGES	Yes	Edit Privileges
user1	localhost	wildcard: my_2	ALL PRIVILEGES	No	Edit Privileges
user2	localhost	wildcard: my_2	SELECT	No	Edit Privileges
userblil	%	global	ALL PRIVILEGES	Yes	Edit Privileges
userlil	%	global	ALL PRIVILEGES	Yes	Edit Privileges

Рисунок 4.4 – Перелік всіх користувачів БД «my_2» та їх привілеїв

Не слід призначати привілеї ALTER звичайним користувачам. Це дає користувачу можливість зруйнувати систему привілеїв шляхом перейменування таблиць!

Позбавлення прав доступу практично ідентично їх призначенням:

REVOKE [тип прав] ON [назва бази даних].[назва таблиці] FROM ‘[им’я користувача]’@‘localhost’;

З використанням команди DROP відбувається видалення користувача:

Приклад 2:

```
…DROP USER ‘demo’@‘localhost’;…
```

Щоб відобразити список користувачів застосовують запит:

```
…SELECT `user` FROM `mysql` . `user`;…
```

Щоб відобразити привілеї користувача застосовують такий запит:

```
…SHOW GRANTS FOR  `username`@`hostname`…
```

Для тестування облікового запису створеного користувача, разлогіньтесь за допомогою команди:quit і залогініться знову, ввівши в термінаті наступну команду:

mysql -u [ім’я користувача]-p.

Контрольні запитання до лабораторної роботи 4:

1. Як визначається мова DCL?

2. Якою SQL командою SQL можна створити нового користувача БД?

3. Якою командою SQL надаються права користувачам БД?

4. Які рівні привілеїв користувачів БД існують в MySQL?

5. Які основні привілеї користувачів БД?

6. Яка з привілей надає користувачу повний доступ до БД?

7. Якою командою позбавляють прав користувачів?

8. Яким чином видаляють користувачів?

9. Які дії виконуються командою FLUSH PRIVILEGES?

Лабораторна робота №5

Тема: Транзакції у роботі з БД

Мета: Навчитися контролювати транзакціями при роботі з БД

Завдання:

1. Відключити режим autocommit.
2. Виконайте вибірку нових даних, намагайтесь змінити значення у таблиці, згідно відібраним.
3. Підтвердіть зміни, що відбулись у БД, збереженням.
4. Нова транзакція: намагайтесь виконати дії над даними своєї БД, але…не підтверджуйте
5. Ігноруйте зміни (відміна). Перевірте правильність виконання.

6. Створіть транзакцію з точкою збереження в потрібному місці та відмініть частину транзакції до іменованої точки збереження (за прикладом).

Теоретичні відомості:

Мова керування транзакціями TCL (Transaction Control Language.) - оператори, що дозволяють контролювати операцію транзакції.

За замовчуванням MySQL працює в режимі autocommit. Це означає, що при виконанні поновлення даних MySQL буде відразу записувати оновлені дані на диск.

Транзакція - послідовність операторів маніпулювання даними (читання, видалення, вставки, модифікації), яка розглядається СКБД, як одне ціле. Транзакція або успішно виконується, і СКБД фіксує зміни БД, які були виконані транзакцією у зовнішній пам'яті, або, у разі невдачі, жодна зміна не відбувається у БД [1].

При використанні таблиць, що підтримують транзакції (таких як InnoDB (дані в налаштуваннях за замовчуванням зберігаються в великих спільно використовуваних файлах), BDB), в MySQL можна відключити режим autocommit за допомогою наступної команди:

SET AUTOCOMMIT = 0

(Нетранзакційні таблиці, наприклад, таблиці типу MyISAM(для кожної таблиці створюється окремий файл) або Memory).

Після цього необхідно застосовувати команду COMMIT для запису змін на диск або команду ROLLBACK, яка дозволяє ігнорувати зміни, вироблені з початку даної транзакції.

Для деяких операторів не можна виконати відкат за допомогою ROLLBACK. В їх число входять оператори мови визначення даних (Data Definition Language - DDL), які створюють і знищують бази даних, а також створюють, видаляють і змінюють таблиці.

Необхідно проектувати свої транзакції таким чином, щоб вони не включали в себе ці оператори. Якщо ввести такий оператор на початку транзакції, яка не може бути відкатаною, і потім наступний оператор пізніше завершиться аварійно, повний ефект транзакції не може бути скасований оператором ROLLBACK.

Якщо необхідно переключитися з режиму AUTOCOMMIT тільки для виконання однієї послідовності команд, то для цього можна використовувати команду BEGIN або BEGIN WORK:

```
…BEGIN;
SELECT @A: = SUM (salary) FROM table1 WHERE type = 1;
UPDATE table2 SET summmary = @ A WHERE type = 1;
COMMIT;…
```

Відзначимо, що при використанні таблиць, які не підтримують транзакції, зміни будуть записані відразу ж, незалежно від статусу режиму autocommit.

При виконанні команди ROLLBACK після поновлення таблиці, що не підтримує транзакції, користувач отримає помилку (ER_WARNING_NOT_COMPLETE_ROLLBACK) у вигляді попередження. Всі таблиці, що підтримують транзакції, будуть перезаписані, але жодна таблиця, що не підтримує транзакції, не буде змінена.

При виконанні команд BEGIN або SET AUTOCOMMIT = 0 необхідно використовувати бінарний журнал MySQL для резервних копій замість більш старого журналу записи змін. Транзакції зберігаються в двійковому системному журналі як одна порція даних (перед операцією COMMIT), щоб гарантувати, що транзакції, за якими відбувається відкат, не заносяться.

Синтаксис команди SET TRANSACTION:

SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

Встановлює рівень ізоляції транзакцій.

За замовчуванням рівень ізоляції встановлюється для подальшої (не перший) транзакції. При використанні ключового слова GLOBAL дана команда встановлює рівень ізоляції за замовчуванням глобально для всіх нових з'єднань, створених від цього моменту. Однак для того, щоб виконати цю команду, необхідно привілей SUPER. При використанні ключового слова SESSION встановлюється рівень ізоляції за замовчуванням для всіх майбутніх транзакцій, які виконуються в поточному з'єднанні.

Встановити глобальний рівень ізоляції за замовчуванням для утиліти mysqld можна за допомогою опції --transaction-isolation.

SAVEPOINT ідентифікатор

ROLLBACK TO SAVEPOINT ідентифікатор

Починаючи з версій MySQL 4.0.14 і 4.1.1, innoDB підтримує SQL-оператори SAVEPOINT і ROLLBACK TO SAVEPOINT.

Оператор SAVEPOINT встановлює іменовану точку початку транзакції з ім'ям ідентифікатор. Якщо поточна транзакція вже має точку збереження з таким ім'ям, стара точка видаляється і встановлюється нова.

Приклад застосування точки збереження:

```
 …
 START TRANSACTION;
 SELECT total FROM accounts WHERE user_id = 5;
 SAVEPOINT accounts_3;
 UPDATE accounts SET total = total - 1000 WHERE user_id = 5;
ROLLBACK TO SAVEPOINT accounts_3;
…
```

Оператор ROLLBACK TO SAVEPOINT відміняє частину транзакції до іменованої точки збереження. Модифікації рядків, які виконувалися поточною транзакцією після цієї точки, скасовуються відкатом, однак InnoDB не знімає блокування рядків, які були встановлені в пам'яті після точки збереження. (Відзначимо, що для знову вставлених рядків інформація про блокування спирається на ідентифікатор транзакції, збережений в рядку, блокування не зберігається в пам'яті окремо. У цьому випадку блокування рядка знімається при скасуванні.) Точки збереження, встановлені в більш пізні моменти, ніж іменована точка, видаляються.

LOCK TABLES блокує таблиці для поточного потоку сервера. UNLOCK TABLES знімає будь-які блокування, утримувані поточним потоком. Всі таблиці, заблоковані в поточному потоці, неявно розблокуються, коли потік виконує інший оператор LOCK TABLES або коли закривається з'єднання з сервером.

LOCK TABLES не є оператором, безпечним щодо транзакцій, і неявно завершує транзакцію перед спробою блокувати таблиці.

Контрольні запитання до лабораторної роботи 5:

1. Що представляє собою TCL?

2. Які команди входять до складу TCL?

3. Яким чином можна відключити режим autocommit?

4. Як почати транзакцію?

5. Що необхідно зробити, щоб підтвердити всі дії транзакції і 6. зберегти зміни у БД?

6. Що необхідно зробити, щоб відмінити всі дії транзакції?

7. Для чого призначена команда COMMIT?

8. Для чого призначена команда ROLLBACK?

9. Для чого призначена команда SET TRANSACTION?

10. Яку дію виконує оператор SAVEPOINT

Лабораторна робота №6

Тема: Створення міжтабличних зв’язків за допомогою SQL, підтримка цілісності БД

Мета: Засобами SQL навчитися створювати зв’язки між таблицями. Набути навичок використання ключів для створення посилань з однієї таблиці на інші. Засвоїти правила, які регламентують введення даних в БД і маніпуляцію ними та обмеження, завдяки яким зберігається цілісність

Завдання:

1. Підготувати всі таблиці для БД, визначити необхідні обмеження (5 таблиць).
2. Визначити зовнішні ключі у підготовлених для зв’язку таблиць.
3. Представте графічне представлення отриманих зв’язків на схемі даних.
4. Створити не можливість видалення зв’язаних записів в батьківській таблиці без видалення запису в дочірній таблиці (ON DELETE CASCADE)
5. Застосуйте ON UPDATE CASCADE для зв’язаних таблиць.

Теоретичні відомості:

Реляційні таблиці розробляються таким чином, що вся інформація розподіляється по безлічі таблиць, причому для даних кожного типу створюється окрема таблиця. Ці таблиці співвідносяться (зв'язуються) між собою через загальні значення, і таким чином є реляційними (відносними) в реляційній конструкції [1].

Розподіл даних за таблицями забезпечує їх більш ефективне зберігання, спрощує маніпулювання даними та підвищує масштабованість.

Зовнішні ключі дозволяють встановити зв'язки між таблицями. Зовнішній ключ встановлюється для стовпців з залежної, підлеглої таблиці,

і вказує на один із стовпців з головної таблиці. Як правило, зовнішній ключ вказує на первинний ключ з пов'язаної головної таблиці.

Первинні ключі є стовпці, значення яких унікально ідентифікують кожен рядок таблиці. Стовпці, які допускають відсутність значень, не можуть використовуватися в якості унікальних ідентифікаторів.

В мові SQL багато з найбільш ефективних інструментів маніпуляції з даними засновані на таких методах, які забезпечуються за допомогою обмежень.

Реляційні бази даних зберігають дані в багатьох таблицях, кожна з яких містить дані, пов'язані з даними з інших таблиць. Для створення посилань з однієї таблиці на інші використовуються ключі (звідси термін цілісність на рівні посилань).

Розглянемо таблицю, що вміщує імена та адреси клієнтів магазина (таблиця 3).

Таблиця 3 – Customers (Клієнти)

CustomerID (Ідентифікатор клієнта)	Name	Address	City
1	Юрій Михайлов	5, вул. Садова	м. Київ
2	Михайло Згама	6, вул. Квіткова	м. Харків
3	Валентин Шевченко	7, вул. Вишнева	м. Кропивницький

У таблиці є ім’я – Customers(Клієнти), декілька стовпців з різного роду даними, а також рядки (кортежі) в яких записано відомості про клієнтів.

Таблиця 4 – Orders (Замовлення)

OrderID (ідентифікатор замовлення)	CustomerID (ідентифікатор клієнта)	Amout(Сума)	Date(Дата)
1	3	30.25	02-01-2020
2	1	12.95	15-01-2020
3	2	74.09	15-01-2020
4	3	5.55	01-02-2020

Як правило БД складаються із декількох таблиць, для яких ключі служать сполучними ланками. Так у таблиці 4 Orders (Замовлення) розміщено дані про замовлення, створені клієнтами.

Щоб реляційна база даних працювала належним чином, необхідно упевнитися в тому, що дані в таблиці введені правильно. Наприклад, якщо в таблиці Orders (таблиця 4) зберігається інформація про замовлення, а в Order Items - його детальний опис, ви повинні бути впевнені, що всі ідентифікатори замовлень, згадані в таблиці OrderItems, існують і в таблиці Orders. Аналогічно, кожен клієнт, згаданий в таблиці Orders, не повинен бути забутий і в таблиці Customers (таблиця 3).

Хоча можна проводити відповідні перевірки, перш ніж вводити нові рядки (виконуючи оператор SELECT для іншої таблиці, щоб впевнитися в тому, що потрібні значення правильні), краще уникати такої практики з наступних причин.

Якщо правила, що забезпечують цілісність бази даних, примусово здійснюються на клієнтському рівні, їх доведеться виконувати кожному клієнту (деякі з клієнтів напевно не захочуть цього робити).

Якщо СКБД виконує ці перевірки - це метод набагато ефективніший.

Цілісність баз даних – властивість даних, що визначає повноту і коректність інформації, яка вміщується в БД, і представляє собою систему правил, які виключають вільну зміну зв’язаних записів.

Обмеження, це правила, які регламентують введення даних в базу даних і маніпуляцію ними.

СКБД примусово забезпечують цілісність на рівні посилань за рахунок обмежень, що накладаються на таблиці бази даних. Більшість обмежень вводиться в визначеннях таблиць (за допомогою операторів CREATE TABLE або ALTER TABLE).

Існує кілька типів обмежень, і кожна СКБД забезпечує свій власний рівень їх підтримки.

Первинний ключ - - це особливе обмеження, що застосовується для того, щоб значення в стовпці (або наборі стовпців) були унікальними і ніколи не змінювалися. Іншими словами, це стовпець (або стовпці) таблиці, значення якого однозначно ідентифікують кожен рядок таблиці. Це полегшує безпосереднє маніпулювання окремими рядками і взаємодію з ними. Без первинних ключів було б дуже важко оновлювати або видаляти певні рядки, не зачіпаючи при цьому інші.

Будь-який стовпець таблиці може бути визначений як первинний ключ якщо він задовольняє наступним умовам:

- Ніякі два рядки не можуть мати однакові значення первинного ключа.
- Кожен рядок має якесь значення первинного ключа (не повинно бути дозволено використання значень NULL.)
- Стовпець, що містить значення первинного ключа, не може бути модифікований або оновлений.
- Значення первинного ключа не можуть бути використані повторно. Якщо якийсь рядок видалено з таблиці, його первинний ключ не може бути призначений іншому рядку.

Зовнішній (Foreign) **ключ** - це стовпець однієї таблиці, значення якого збігається зі значеннями стовпчика, що є первинним ключем іншої таблиці. Зовнішні ключі - дуже важлива частина механізму забезпечення посилальної цілісності даних. Для розуміння зовнішніх ключів розглянемо наступний приклад.

Таблиця Orders (таблиця 4) містить єдиний рядок для кожного замовлення, зафіксованого в БД. Інформація про клієнта зберігається в таблиці Customers (таблиця 3). Замовлення в таблиці Orders пов'язані з певними рядками в таблиці Customers за рахунок ідентифікатора клієнта. Ідентифікатор клієнта є первинним ключем в таблиці Customers; кожен клієнт має унікальний ідентифікатор. Номер замовлення є первинним ключем в таблиці Orders; кожне замовлення має свій унікальний номер.

Значення в стовпці таблиці Orders, що містить ідентифікатори клієнтів, не обов'язково унікальні. Якщо клієнт зробив кілька замовлень, може бути кілька рядків з тим же самим ідентифікатором клієнта.

Зовнішні ключі допомагають примусово зберігати цілісність посилальних даних, вони можуть виконувати багато інших важливих функцій. Після того як зовнішній ключ визначено, СКБД не дозволить видаляти рядки, пов'язані з рядками в інших таблицях. Наприклад, ви не зможете видалити інформацію про клієнта, у якого є замовлення. Єдиний спосіб видалити інформацію про такого клієнта полягає в попередньому видаленні пов'язаних з ним замовлень (для чого, в свою чергу, потрібно видалити інформацію про предмети цих замовлень). Оскільки потрібно настільки методичне й цілеспрямоване видалення, Foreign ключі допомагають в запобіганні випадкового видалення даних.

У деяких СКБД підтримується можливість, що отримала назву каскадне видалення. Якщо така функція реалізована, можна видаляти всі пов'язані з цим рядком дані при видаленні його з таблиці. Наприклад, якщо можливо каскадне видалення й ім'я клієнта видаляється з таблиці Customers, всі пов'язані з його замовленням рядки видаляються автоматично.

Додавання та видалення зовнішнього ключа.

Наприклад необхідно створити дві таблиці, які потім будуть пов’язані.

Це виконується за допомогою наступних запитів:

```
…CREATE TABLE Customers
(
    Id INT PRIMARY KEY AUTO_INCREMENT,
    Age INT,
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL
);
CREATE TABLE Orders
(
    Id INT PRIMARY KEY AUTO_INCREMENT,
    CustomerId INT,
    CreatedAt Date
);…
```

Для додавання зовнішнього ключа до стовпчика CustomerId таблиці Orders виконується запит:

```
…ALTER TABLE Orders
ADD FOREIGN KEY(CustomerId) REFERENCES Customers(Id);…
```

Для додавання обмежень можливо вказати для них ім’я, застосовуючи оператор CONSTRAINT, після якого вказується ім’я обмеження:

```
…ALTER TABLE Orders
ADD CONSTRAINT orders_customers_fk
FOREIGN KEY(CustomerId) REFERENCES Customers(Id);…
```

В цьому випадку обмеження зовнішнього ключа називається orders_customers_fk. За цим іменем можливо видалити це обмеження:

```
…ALTER TABLE Orders
DROP FOREIGN KEY orders_customers_fk;…
```

Наприклад, для створення зовнішнього ключа PNUM під час зв’язування таблиць, що розглядались раніше PREDMET і USP застосовується запит, що на рисунку 6.1.

```
1 Alter table USP
2 ADD CONSTRAINT USP_Predmet_fk
3 FOREIGN KEY (PNUM) REFERENCES
  Predmet(PNUM);
4
```

Рисунок 6.1 - Приклад застосування запиту для створення зовнішнього ключа

Щоб візуально побачити зв’язок між таблицями можна вибрать Дизайнер з вкладки меню (рисунок 6.2)

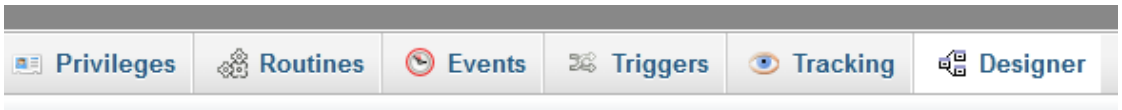


Рисунок 6.2 - Рядок меню з дизайнером в кінці

Результатом може бути схема, наприклад, як на рисунку 6.3.

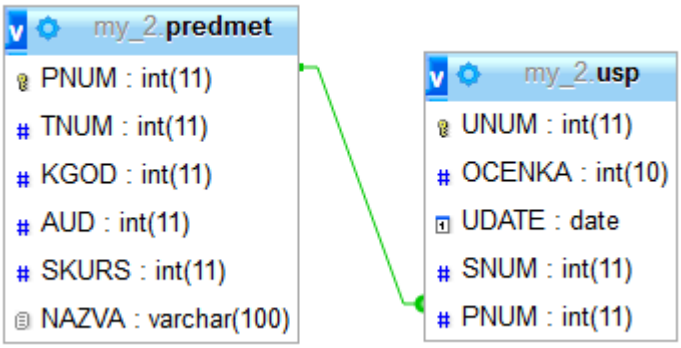


Рисунок 6.3 – Зв’язок на схемі

Щодо загального синтаксису для визначень FOREIGN KEY рівня таблиць: [CONSTRAINT ім’я_обмеження]

FOREIGN KEY (стовпчик1, стовпчик2, ... стовпчикN)

REFERENCES головна_таблиця (стовпчик_головної_таблиці1, стовпчик_головної_таблиці2, ... стовпчик_головної_таблиціN)

[ON DELETE дія]

[ON UPDATE дія]

При створенні обмежень для зовнішнього ключа після FOREIGN KEY вказується стовпець таблиці, що планується як зовнішній ключ. А після ключового слова REFERENCES вказується ім'я пов'язаної таблиці, а потім в дужках ім'я пов'язаного стовпця, на який буде вказувати зовнішній ключ. Після висловлення REFERENCES йдуть вирази ON DELETE і ON UPDATE, які задають дію при видаленні й оновленні рядку з головної таблиці відповідно.

ON DELETE и ON UPDATE

За допомогою виразів ON DELETE і ON UPDATE можна встановити дії, які виконуються відповідно при видаленні та зміні пов'язаного рядка з головної таблиці. Як дії можуть використовуватися такі опції:

• CASCADE: автоматично видаляє або змінює рядки з залежною таблиці при видаленні або зміні пов'язаних рядків в головній таблиці.

• SET NULL: при видаленні або оновленні пов'язаного рядка з головної таблиці встановлює для стовпця зовнішнього ключа значення NULL. (В

цьому випадку стовпець зовнішнього ключа повинен підтримувати установку NULL)

• RESTRICT: відхиляє видалення або зміну рядків в головній таблиці при наявності пов'язаних рядків у залежній таблиці.

• NO ACTION: те ж саме, що і RESTRICT.

• SET DEFAULT: при видаленні пов'язаного рядку з головної таблиці встановлює для стовпця зовнішнього ключа значення за замовчуванням, яке задається за допомогою атрибуту DEFAULT. Незважаючи на те, що дана опція в принципі доступна, проте двигун InnoDB не підтримує цей вислів.

Каскадне видалення

Каскадне видалення дозволяє при видаленні рядка з головної таблиці автоматично видалити всі пов'язані рядки з залежної таблиці. Для цього застосовується опція CASCADE:

```
...CREATE TABLE Orders
(
    Id INT PRIMARY KEY AUTO_INCREMENT,
    CustomerId INT,
    CreatedAt Date,
    FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE
);…
```

Подібним чином працює і вираз ON UPDATE CASCADE. При зміні значення первинного ключа автоматично зміниться значення пов'язаного з ним зовнішнього ключа. Однак оскільки первинні ключі змінюються дуже рідко, та й в принципі не рекомендується використовувати в якості первинних ключів стовпці із змінними значеннями, то на практику вираз ON UPDATE використовується рідко.

Встановлення NULL

При встановленні для зовнішнього ключа опції SET NULL необхідно, щоб стовпець зовнішнього ключа допускав значення NULL:

```
    …CREATE TABLE Orders
(
    Id INT PRIMARY KEY AUTO_INCREMENT,
    CustomerId INT,
    CreatedAt Date,
```

```
    FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE SET NULL
);…
```

Приклад створення бази даних детальніше можна розглянути у Додатку Б.

Контрольні запитання до лабораторної роботи 6:

1. Що означає поняття первинного ключа?

2. Як визначити зовнішній ключ.

3. Що означає поняття цілісності БД?

4. Що означає поняття обмежень БД та які обмеження ви знаєте?

5. Яким чином можна створити первинний ключ?

6. Якими способами за допомогою SQL визначають зовнішні ключі?

7. Яким чином створюється зв'язок?

8. Як можливо здійснити видалення зв'язку між таблицями?

9. Для чого використовують опцію CASCADE?

10. Що означає поняття цілісності даних?

11. Що означає вираз ON UPDATE?

12. Що означає вираз ON DELETE?

13. Що означає поняття каскадне видалення?

14. Що означає поняття каскадне відновлення?

Лабораторна робота №7

Тема: Типи з’єднань в MySQL

Мета:. Навчитись реалізовувати засоби SQL для виконання сполучення таблиць у запитах на вибірку та при цьому групування й агрегування даних

Завдання:

1. Створити 4 зв’язані таблиці, відповідно до своєї схеми даних.
2. Заповнити таблиці відповідними даними (не менше 10 рядків).
3. Виконати запити на вибірку з операціями з’єднання з використанням псевдонімів для таблиць:

А) Внутрішнє з’єднання;

Б) Лівостороннє з’єднання;

В) Правостороннє з’єднання;

Г) Повне з’єднання;

Д) Перехресне з’єднання;

4. Провести групування даних в визначеному порядку з використанням конструкції ORDER BY й ключових слів та провести агрегування.

Теоретичні відомості:

Join - операція з'єднання таблиць в SQL, яка сполучає дві таблиці в реляційній базі даних, утворюючи нову тимчасову таблицю, яку інколи називають «з'єднаною таблицею».

Згідно з ANSI-стандартом, в SQL існують такі типи з'єднання: внутрішнє - INNER, зовнішнє –OUTER та перехресне - CROSS. Зовнішнє з'єднання поділяється на ліве –LEFT OUTER, праве – RIGHT OUTER та повне – FULL OUTER. Особливим випадком є з'єднання таблиці з собою, що має назву самоз'єднання (англ. self-join).

Візуально про кожен тип Join можна оглянути на рисунку 7.1

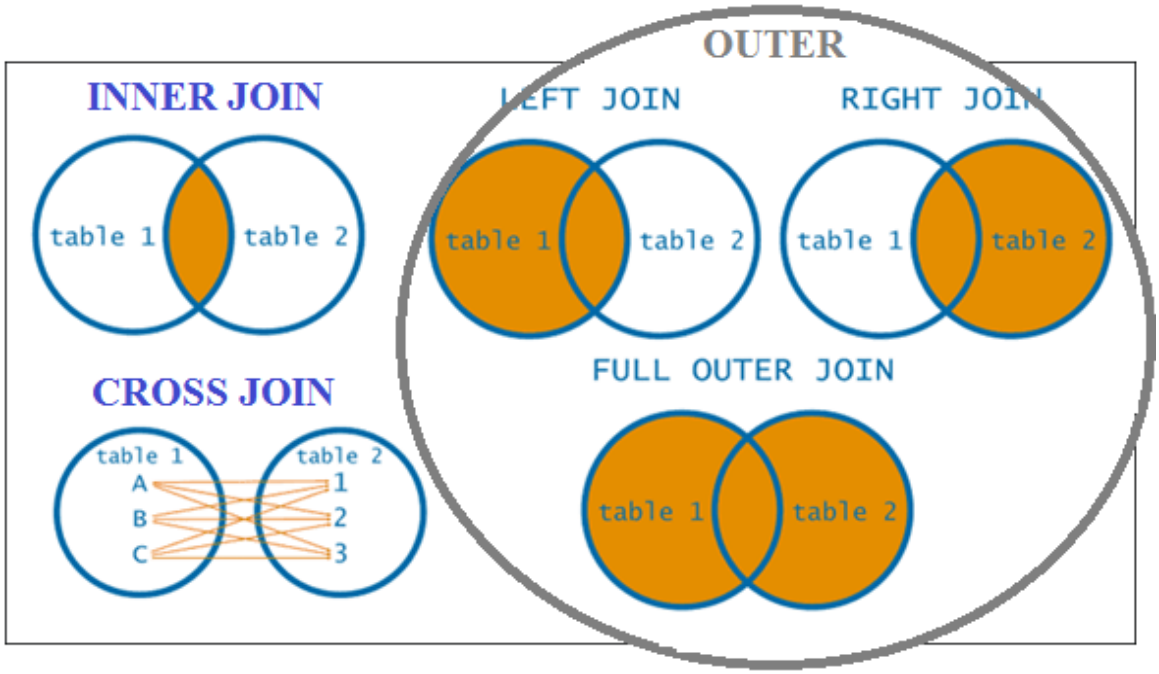


Рисунок 7.1 – Візуальне подання типів з'єднання

При **внутрішньому** з'єднанні з'єднуються записи двох таблиць (1 і 2) враховуючи предикат з'єднання. Виконується обчислєння декартового добутку всіх записів таблиць. Як результат, всі записи таблиці 1 буде поєднано з кожним із записів таблиці 2. Зауважимо, що після виконаного в таблиці отримаємо лише ті записи, які задовільняють предикат з'єднання.

Результат **лівого зовнішнього** з'єднання для таблиць 1 і 2 включає всі записи з лівої таблиці (1), навіть якщо умова з'єднання не знаходить відповідностей у правій таблиці (2). Це означає, що якщо порівняння не виявляє відповідних записів у таблиці 2, результат з'єднання все одно поверне рядки, але значення стовпців з таблиці 2 будуть порожніми. Інакше кажучи, ліве зовнішнє з'єднання включає всі записи з лівої таблиці, а також значення стовпців з правої таблиці або NULL, якщо немає відповідності за умовою з'єднання.

Якщо працює **праве зовнішнє** з'єднання для таблиць 1 і 2, то результат буде містити з правої таблиці - всі записи (2), навіть якщо в умові з'єднання немає збігів з кортежами лівої таблиці (1). Тобто, праве зовнішнє з'єднання повертає всі значення з правої таблиці й додає значення з лівої таблиці або NULL, якщо за предикатом з'єднання немає збігу.

Якщо розглядать **повне зовнішнє** з'єднання, то при цьому сполучаються результати правого та лівого зовнішніх з'єднань. Отримуємо в результаті таблицю, що містить всі записи з двох таблиць, і позначені NULL-значеннями у випадку відсутності збігів з кожного боку.

Оператор **CROSS JOIN**, ще називають оператором декартового з'єднання, він по'єднує дві таблиці. Через те, що оператор є симетричним, тому порядок вказування таблиць не є принциповим. В результаті отримується таблиця, що має у своєму заголовку об'єднання заголовків таблиць, що по'єднуються. А тіло таблиці результату складається так: кожен рядок одної таблиці по'єднується з рядком іншої таблиці, виконуючи в результаті всі можливі поєднання рядків з двох таблиць.

Наприклад, якщо для виконання роботи застосувати раніше створені таблиці (orders, customers), що розглядаються на рисунках 7.2 та 7.3, можна за допомогою відповідних запитів виконати вибір необхідної інформації саме в тому вигляді, як потрібно.

order_id	customer_id	amount	date
1	3	30.25	2023-01-02
2	1	12.95	2024-01-15
3	3	45.00	2020-01-16
4	1	78.00	2020-01-02
6	2	25.25	2020-01-07

Рисунок 7.2 – Таблиця orders

customer_id	name	address	city
1	Юрій Михайлович	вул.Садова 5	м.Київ
2	Михайло Згама	вул.Квіткова.6	м.Харків
3	Валентин Шевченко	вул.Вишнева 7	м.Кропивницький

Рисунок 7.3 - Таблиця customers

Пошук замовлень з двох таблиць які зробив Юрій Михайлович:

```
…SELECT orders.customer_id, orders.amount, orders.date
FROM customers, orders
WHERE customers.name = 'Юрій Михайлович'
andcustomers.customer_id = orders.customer_id;…
```

Результат виконання запиту розглянуто на рисунку 7.4.

customer_id	amount	date
1	12.00	2020-01-07
1	78.00	2020-01-02

Рисунок 7.4 - Результат виконання

При роботі з БД додаємо ще 3 таблиці для виконання необхідних запитів. Таблиці розглянуто на рисунках 7.5, 7.6, 7.7.

```
…CREATE TABLE books
(
isbnchar(15) NOT NULL PRIMARY KEY,
authorchar(45),
titlechar (100),
pricefloat(4,2)
);…
```

←T→	isbn	author	title	price
	5-8459-0046-8	Майкл Морган	JAVA2. Для Розробників	34.99
	6-8459-0046-8	Кристоф Негус	Linux. Для Розробників	25.99
	7-8459-0046-8	Марина Смоліна	Corel. Для Розробників	25.99
	8-8459-0046-x	Родерік Сміт	Мережі Linux	49.99

Рисунок 7.5 - Таблиця books

```
CREATE TABLE order_items (
orderidintunsigned NOT NULL,    isbnchar(15) NOT NULL,
quantitytinyintunsigned,
    PRIMARY KEY(orderid,isbn)
);…
```

←T→	orderid	isbn	quantity
	1	5-8459-0046-8	2
	2	5-8459-0046-8	1
	3	6-8459-0046-8	1
	3	7-8459-0046-8	1
	4	8-8459-0046-x	3

Рисунок 7.6 - Таблиця order_items

```
…CREATE TABLE book_reviews (
isbnchar(13) NOT NULL PRIMARY KEY,
reviewtext);…
```

Таким чином таблицю створили.

←T→	Isbn	Review
	5-8459-0046-8	Книга Моргана краща по JAVA

Рисунок 7.7 - Таблиця book_reviews

Далі заповнимо таблиці значеннями оператором INSERT (по одному запиту) або використовуючи графічний інтерфейс MySql:

```
…INSERT INTO books (`isbn`, `author`, `title`, `price`)  VALUES (
("5-8459-0046-8","Майкл Морган","JAVA2.Для Розробників",34.99),
("6-8459-0046-8","Кристоф Негус","Linux.Для Розробників",25.99),
("7-8459-0046-8","Марина Смоліна","Corel. Для Розробників",25.99),
    ("8-8459-0046-x","РодерікСміт","МережіLinux",49.99)
);…
```

Продовжуємо з іншими таблицями.

```
…INSERT INTO order_items VALUES (
(1,"5-8459-0046-8",2),
(2,"6-8459-0046-8", 1),
(3,"6-8459-0046-8",1),
(3,"7-8459-0046-8", 1),
    (4, "8-8459-0046-x",3)
);
```

Наступні введення даних:

```
    INSERT INTO `book_reviews` (`isbn`, `review`) VALUES ('5-8459-0046-8',
'Книга Моргана краща по JAVA'); …
```

Виконаємо запит: хто робив хоча б одне замовлення по курсу JAVA:

1) Для цього перше що потрібно - це написати, що ми хочемо отримати в таблиці:

SELECT customers.name…

2) Об’єднати усі 4 таблиці, бо замовлені книги - в таблиці order, а ім’я замовника - зв’язане з таблицею custmers.

3) Зроблені замовлення в таблиці Order_items, а опис книг для запиту пошуку по опису відповідно в таблиці books.

Тому виконаємо запит INNER JOIN, замінивши записом через кому:

…FROM customers, orders, order_items, books…

4) Далі необхідно пройтися по зв’язкам таблиць та задати пошук:

…WHERE customers.customer_id = orders.customer_id

AND orders.order_id = order_items.orderid

AND order_items.isbn = books.isbn

andbooks.title like '%Java%';

…

Загальний вигляд запиту буде таким:

```
…SELECT customers.name
FROM customers, orders, order_items, books
WHERE customers.customer_id = orders.customer_id
AND orders.order_id = order_items.orderid
AND order_items.isbn = books.isbn
andbooks.title like '%Java%';…
```

А результат на рисунку 7.8.

name	title
Валентин Шевченко	JAVA2. Для Розробників
Юрій Михайлович	JAVA2. Для Розробників
Леонід Макаров	Розробка Веб-додатків за допомогою PHP і MySQL та ...

Рисунок 7.8 – Результат запиту

Видно, що в усіх книгах в назві присутня назва «Java».

Далі виконаємо розповсюджений в MySQL тип з’єднання «LEFT JOIN».

Попередні запити були на пошук кортежів, в яких була відповідність в таблицях. Але інколи буває, що потрібно знайти і рядки, наприклад, які не зробили жодного замовлення, або книги, які ніхто не замовляв. Найпростіше тут виконати лівостороннє з’єднання, при якому виконується

пошук рядків за вказаною умовою з’єднання двох таблиць. Якщо у вказаній справа таблиці немає підходящого кортежу, то до результату додається рядок з нульовими значеннями у відповідних стовпчиках.

Приклад:

```
…
SELECT customers.customer_id, customers.name, orders.order_id
FROM customers LEFT JOIN orders
ONcustomers.customer_id = orders.customer_id;
…
```

Далі виконаємо виведення тільки тих клієнтів, які нічого не замовляли, це перевірка на значення NULL в полі первинного ключа правої таблиці (order_id), бо поля з реальними значеннями не можуть мати значення NULL.

```
…
SELECT customers.customer_id, customers.name
FROM customers LEFT JOIN orders
USING (customer_id)
WHERE orders.order_id is NULL;
…
```

Використання псевдонімів

Для цього використовується конструкція AS (alias). Їх можна створити на самому початку запиту, а потім використовувати за необхідності. Часто псевдоніми використовують в якості коротких імен.

Розглянемо запит:

```
…
SELECT c.name
FROM customersas c, ordersas o, order_items asoi, booksas b
WHERE c.customer_id = o.customer_id
AND o.order_id = oi.orderid
AND oi.isbn = b.isbn
AND b.title LIKE '%Java%';
…
```

Такий запит більш короткий, крім цього, псевдоніми можна привласнювати стовпчикам. Псевдоніми таблиць необхідні коли потрібно з’єднати таблицю з самою собою (наприклад для пошуку в цій же таблиці рядків, які мають однакові значення). Так, якщо нам буде необхідно знайти клієнтів, які проживають в одному місті, можна тій самій таблиці присвоїти різні псевдоніми й виконати пошук:

```
…
SELECT c1.name, c2.name, c1.city
FROM customersas c1, customersas c2
WHERE c1.city = c2.city
```

```
    AND c1.name != c2.name
…
```

Функції агрегування MySQL

Нерідко буває необхідно визначити, скільки рядків відноситься до певного набору або яке середнє значення будь-якого стовпця таблиці, скажімо, середня сума замовлення в грошовому еквіваленті. Для цього в MySQL використовуються функції агрегування. Дані функції можна застосовувати як до таблиці в цілому, так і до груп даних всередині таблиці.

Функції, що найбільш часто використовуються на практиці наведено в таблиці 5.

Таблиця 5 – Функції агрегування в MySQL

AVG(стовпчик)	Середня величина значень у вказаному стовпці.
COUNT(елементи)	При вказуванні стовпця отримується кількість не нульових значень. Якщо перед назвою стовпця помістити оператор DISTINCT, видається тільки кількість різних значень в стовпці. Якщо вказати COUNT(*) то підрахунок рядків буде проведено незалежно від нульових значень.
MIN(стовпець)	Мінімальне значення у вказаному стовпці.
MAX(стовпець)	Максимальне значення у вказаному стовпці.
STD(стовпець)	Стандартне відхилення у вказаному стовпці.
STDDEV(стовпець)	Аналогічно попередньому.
SUM(стовпець)	Сума значень у вказаному стовпці.

Розглядається приклад з AVG:

```
…USE books;
SELECT AVG(amount) FROM ORDERS;…
```

Результат на рисунку 7.9:

AVG(amount)
36.750000

Рисунок 7.9 – Результат прикладу з AVG

Для отримання більш детальної інформації виконаємо наступний запит:

```
...SELECT customer_id, AVG(amount)
FROM orders
GROUP BY customer_id;…
```

Це дозволить отримати середню суму замовлення по групам, наприклад, за номерами клієнта, щоб виявити, хто робить найбільш великі замовлення. Вказавши разом з конструкцією GROUP BY разом із функцією агрегування, зміниться поведінка функції і тепер ми отримаємо не середню суму всіх замовлень, а інформацію по середній сумі замовлень усіх клієнтів (кожним customer_id).

Результат на рисунку 7.10:

customer_id	AVG(amount)
1	45.000000
2	25.250000
3	37.500000
4	30.250000

Рисунок 7.10 – Результат запиту

Про оператор **GROUP BY**

Для обчислення сумарних значень на основі даних однієї або декількох таблиць можна використати оператор GROUP BY, що має такий синтаксис:

GROUP BY {column1} [, ...]

Наприклад, наступний запит зв'язує дві таблиці, сортує їх по полю Customeri, для кожного значення Customeri створює один рядок у результуючому наборі даних й обчислює кількість значень поля Orderi для кожного значення Customeri:

```
SELECT Customers.Customeri, COUNT (Orders.Orderi) FROM Customers INNER JOIN Orders ON Customers.Customeri = Orders.Customeri GROUP BY Customers.Customeri
```

У наведеному вище прикладі запиту використано в операторі SELECT агрегатну функцію COUNT, що обчислює кількість значень.

Додаткові приклади можливо продивитись за посиланнями (просто, про загальне чи окремі речі про JOIN):

https://www.youtube.com/watch?v=XKoW3Bs-0eI

https://www.youtube.com/watch?v=nUyO7saJ3Xc

https://www.youtube.com/watch?v=E9-CIfJZCGk

https://www.youtube.com/shorts/8bl2LS1rLmw

Контрольні запитання до лабораторної роботи 7:

1. **Які типи з’єднань існують в MySQL?**
2. **Що означає операція Join у реляційній базі даних?**
3. **На які види з'єднань поділяється зовнішнє об’єднання даних?**
4. **Що означає застосування CROSS JOIN?**
5. **Яким чином можна організувать внутрішнє з'єднання даних?**
6. **Що означає термін самоз’єднання?**
7. **Що таке псевдоніми (alias)?**
8. **Які функції агрегування часто розглядаються?**
9. **Який синтаксис для JOIN при об’єднанні 3 і більше таблиць?**

Додаток А

Індивідуальні теми для виконання завдань з БД

1. Автоматизована система продажу комп’ютерної техніки та комплектуючих матеріалів.
2. База даних для агентства працевлаштування.
3. Автоматизована система «Автосалон».
4. База даних для Автошколи.
5. Автоматизована система для роботи фірми з обслуговування та ремонту офісної техніки.
6. Розробка інформаційної системи «Телефонний довідник».
7. Автоматизована система «Географічний довідник».
8. Автоматизація роботи книжкового магазину.
9. Автоматизація роботи авіа агентства.
10. Автоматизація роботи завідуючого військовим складом.
11. Інформаційна система інституту досліджень паранормальних явищ.
12. Побудова інформаційної системи «Туристичне агентство».
13. Розробка автоматизованої системи обліку успішності в школі.
14. Розробка програми для автоматизації учбового процесу.
15. Тестуюча система з дисципліни «Бази даних».
16. Автоматизована система психологічного тестування людини.
17. Інформаційна система «Розклад занять у навчальних закладах».
18. Автоматизована система «Обласна стоматологічна поліклініка».
19. Інформаційна система для обліку досліджень патологоанатомічного відділення лікарні.
20. Автоматизована система роботи музею.
21. Інформаційна система для роботи цеху на заводі.
22. Каталогізатор відеоінформації.
23. Розробка інформаційної системи «Адресне бюро».

24. Інформаційна система «Автовокзал».

25. Автоматизована система оптового складу комп’ютерної техніки.

26. Автоматизована система обліку будівельних матеріалів на складі.

27. Розробка інформаційної системи для бібліотеки.

28. Інформаційно-обчислювальна система «Інтернет-провайдер».

29. Розробка програми для автоматизації роботи куратора у ВУЗах.

30. Автоматизована система обліку продукції на складі мототехніки.

31. Автоматизована система обліку матеріалів на аптечних складах.

32. Автоматизована система обліку продукції м’ясокомбінату.

33. Інформаційна система для роботи деканату.

34. Розробка автоматизованої системи розрахунку електроенергії, яка споживається населенням.

35. Автоматизована система для роботи пенсійного фонду.

36. Автоматизована система обліку заробітної плати для малого підприємства.

37. База даних для сайту про авіацію та космонавтику.

38. Розробка автоматизованої системи для СТО.

39. Інформаційна система «Розклад руху потягів залізничного вокзалу».

40. Автоматизована система роботи мережі магазинів.

41. Автоматизована система «Відділ кадрів».

42. Автоматизована система роботи спортивного клубу.

43. Інформаційна система для професійної футбольної ліги.

44. Автоматизована система «Радіостанція».

45. Автоматизована система організації фестивалів.

46. Розробка програми для автоматизації робочого місця робітника відділу банка.

47. Розробка автоматизованої системи розрахунку населення за послуги газопостачання.

48. Автоматизована система для сплати комунальних платежів.

49. Автоматизована система для реєстрації абітурієнтів у ВУЗах.
50. База даних для інтернет магазину одягу.
51. База даних для магазину квітів.
52. База даних для продуктового магазину.
53. Інформаційна система косметичної фірми.
54. Розробка бази даних для агентства нерухомості.
55. Інформаційна система для піцерії.
56. Автоматизація роботи ветеринарної клініки.
57. База даних для салону краси.
58. База даних для сайту з продажу меблів.
59. Автоматизація роботи фірми з пошиття та ремонту взуття.
60. БД для сайту фірми з виготовлення та ремонту меблів.
61. Інформаційна система дитячого розважального центру.
62. Розробка бази даних для сайту знайомств.
63. База даних для сайту з продажу канцелярських товарів.
64. База даних для сайту з продажу програмного забезпечення.
65. База даних для сайту кінотеатру.
66. База даних для дитячого дошкільного закладу.
67. База даних для танцювального колективу.
68. Розробка програмного забезпечення для навчання з дисципліни «Бази даних».
69. База даних для населеного пункту.
70. База даних для музичної школи.
71. База даних типографій міста Кропивницький.
72. База даних для госпіталя.
73. База даних для сайту волонтерської організації.
74. Інформаційна система для Державної податкової адміністрації.
75. База даних фірми з організації свят.

76. Електронний кабінет платника для Державної фіскальної служби України.

77. База даних для іподрому.

78. База даних для сайту краєзнавчого музею.

79. База даних магазину зоотоварів.

80. Інформаційна система орнітологічного клубу.

81. База даних для роботи рибного господарства.

82. Автоматизована система для роботи фермерського господарства.

83. База даних для обліку підприємницької діяльності.

84. Інформаційна система «Зоопарк».

85. База даних для Державної служби охорони.

86. Інформаційна система для телефонної компанії.

87. Автоматизація розрахунку навантаження викладачів учбового закладу.

88. База даних для сайту учбового закладу.

89. Інформаційна система з продажу комп'ютерних ігор.

90. Автоматизація продажу для магазину дитячих іграшок.

91. База даних для компанії з розробки програмного забезпечення.

92. База даних для виставки агропродукції.

93. Інформаційна система для школи мистецтв.

94. База даних для КРЕП.

95. База даних для магазину ювелірних виробів.

96. База даних для роботи юридичної компанії.

97. База даних для магазину побутової техніки.

98. Програмне забезпечення для автоматизації роботи бухгалтера.

99. Автоматизація продажу продукції підприємства.

100. База даних для сайту про архітектуру та дизайн.

Додаток Б

Приклад для створення простої бази даних

Для побудови простої БД «Книги» планується створення двох таблиць (зв’язаних) відповідно до рисунку 8.1. В кожну таблицю необхідно додавать дані, Таблиця Customers – головна, Orders – підлегла.

За допомогою операторів SQL вносяться відповідні дані в кожну таблицю по 5 кортежів в кожну. Виконується зв'язок таблиць. Створюються відповідні запити різних типів для виконання робіт з БД.

Customers (Клієнти)

CustomerID (Ідентифікатор клієнта)	Name	Address	City
1	Юрій Михайлов	5,вул. Садова	м. Київ
2	Михайло Згама	6, вул. Квіткова	м. Харків
3	Валентин Шевченко	7, вул. Вишнева	м. Кропивницький

Orders(Закази)

OrderID (ідентифікатор заказа)	CustomerID (ідентифікатор клієнта)	Amout(Сума)	Date(Дата)
1	3	30.25	02-01-2020
2	1	12.95	15-01-2020
3	2	74.09	15-01-2020
4	3	5.55	01-02-2020

Рисунок 8.1 – Таблиці Customers та Orders для БД «Книги»

В кожному замовленні з таблиці Orders є вказівник на клієнта із таблиці Customers.

Частіше за все, бази даних складаються із деякої кількості таблиць, для яких ключі служать сполучними ланками зв’язків. Так, на рисунку 8.1 показані таблиці, що складають БД. В ній розміщено дані про закази книг, створені клієнтами.

Для реалізації планів роботи з БД необхідно зробити вибір програмного забезпечення. В цьому випадку завантажується спочатку

OpenServer (можливо застосовувати інший пакет програм). Застосовується PhpMyAdmin.

Після завантаження в PhpMyAdmin вибирається вкладка SQL для виконання SQL запитів та створюється нова БД з назвою «books».

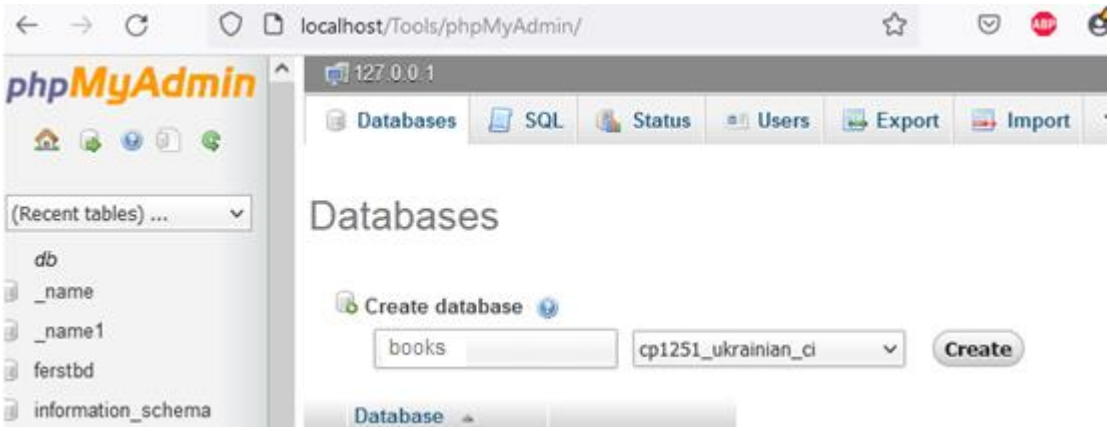


Рисунок 8.2 – Створення нової БД в PhpMyAdmin

За необхідністю виконуються відповідні SQL-запити.

Для видалення непотрібних можливих існуючих таблиць виконуються запити:

```
DROP TABLE IF EXISTS Customers;
DROP TABLE IF EXISTS Orders;
```

Та виконується запит для створення відповідної бази даних:

```
CREATE DATABASE books
```

Для переходу до потрібної БД застосовується запит:

USE books;

Після створення БД починається наповнення її об’єктами, а саме, деякими таблицями: «Customers» й «Orders» за допомогою мови DDL.

Згідно плану за допомогою запиту DDL у вкладці SQL виконується створення таблиці «Customers» з відповідними полями, як показано на рисунку 8.1:

```
…CREATE TABLE customers(
customer_id INT unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(45) NOT NULL,
address CHAR(55) NOT NULL,
city CHAR(25) NOT NULL
);…
```

Після виконання запиту система повинна повернути поле запиту та час виконання запиту.

Аналогічно, наступним запитом створюється таблиця «Orders»:

```
…CREATE TABLE orders(
order_id INT unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
customer_id INT unsigned NOT NULL,
amount float(6,2),
date date NOT NULL
);…
```

Після створення двох таблиць (між ними буде організовано зв'язок), одна з них буде «дочірня» (зберігає дані з батьківської таблиці) а перша буде називатись «батьківська». Це будуть відношення між даними створені за допомогою зовнішніх ключів. Таким чином в цій базі даних створюється відношення між значеннями в таблиці «Orders» та значеннями в таблиці «Customers».

Зв'язок між таблицями виконується за ідентифікатором клієнта (Customer_id) за допомогою зовнішніх ключів (Foreign key). Синтаксис надається нижче відповідно до документації MySQL[9]:

[CONSTRAINT[*symbol*]] FOREIGN KEY [*index_name*] (*col_namex*,...)

REFERENCES *tbl_name* (*col_name1*,...)

[ON DELETE *reference_option*]

[ON UPDATE *reference_option*]

reference_option:

RESTRICT|CASCADE|SETNULL|NOACTION

Дані до таблиць краще не додавать поки не будуть виконані всі підготовчі дії. Виконуються запити на зміну таблиці для додавання FOREIGN KEY:

```
…ALTER TABLE orders
ADD FOREIGN KEY(customer_id ) REFERENCES customers(customer_id)
ON DELETE CASCADE;…
```

Після виконання відповідного запиту перейшовши в розділ Структура (Structure) відповідної таблиці потім – Зв’язки, можна побачити організацію зв’язків міжтабличних та відповідні встановлені обмеження (рисунок 8.3). Також можна здійснити необхідні зміни, додати FOREIGN KEY чи виправити існуючий.

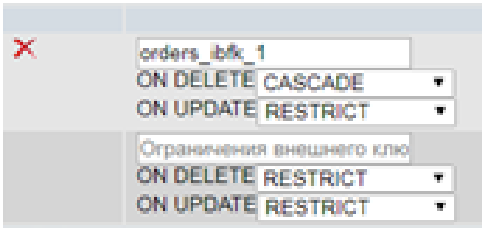


Рисунок 8.3 – Встановлені обмеження

Як результат ми отримали дві пов'язані таблиці. Після того, як виконані всі підготовчі роботи, всі об'єкти необхідні для роботи створені, можна додавать дані до таблиць у відповідній послідовності.

При внесенні даних в таблиці, спочатку повинна заповнюватися батьківська таблиця (клієнти), з причини, що дані в полі Customers_id таблиці Orders будуть залежати від даних батьківської таблиці. Так в таблиці Orders в полі запису Customer_Id буде не коректним і помилковим внесення даних ще не існуючого ID клієнта.

Наступним етапом є заповнення таблиць бази даних інформацією використовуючи DML оператор INSERT, що має наступний вигляд:

INSERT [INTO] таблиця [(стовпець1, стовпець2,….)]

VALUES (значення1, значення2, значення3,…).

Заповнюється таблиця клієнти за прикладом [6], що наведено нижче:

```
…INSERT INTO `customers` (`customer_id`, `name`, `address`, `city`) VALUES
(NULL, 'Михайло Згама', 'вул. Квіткова.6', 'м.Харків');
INSERT INTO `customers` (`customer_id`, `name`, `address`, `city`) VALUES
(NULL, 'Валентин Шевченко', 'вул.Вишнева 7', 'м.Кропивницький');…
```

Після заповнення таблиці «клієнти» приступаємо заповнювати таблицю Orders («закази»).

```
…INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`) VALUES
(NULL, '3', '30', '2020-01-01'), (NULL, '1', '12', '2020-01-07');
INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`) VALUES
(NULL, '3', '45', '2020-01-16'), (NULL, '1', '78', '2020-01-02');…
```

Для перевірки спробуємо додати в таблицю «замовлення» дані з неіснуючим ідентифікатором клієнта (order_id = 5):

```
INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`) VALUES
(NULL, '5', '45', '2024-01-16')…
```

Звичайно, система видасть повідомлення про помилку, тому що клієнта з таким ідентифікатором, поки що, не існує, й відповідний запит ми зможемо виконати тільки при появі клієнта з Custmer_id = 5.

Такий підхід дозволяє зберегти цілісність даних в таблицях БД. Це добре видно при роботі в графічному інтерфейсі, коли виконується додавання даних, система автоматично пропонує можливі варіанти для заповнення таблиці «Замовлення» поле «customer_id» рисунок 8.4. Як зображено на рисунку варіантів крім тих що є в таблиці «Клієнти» система не пропонує.

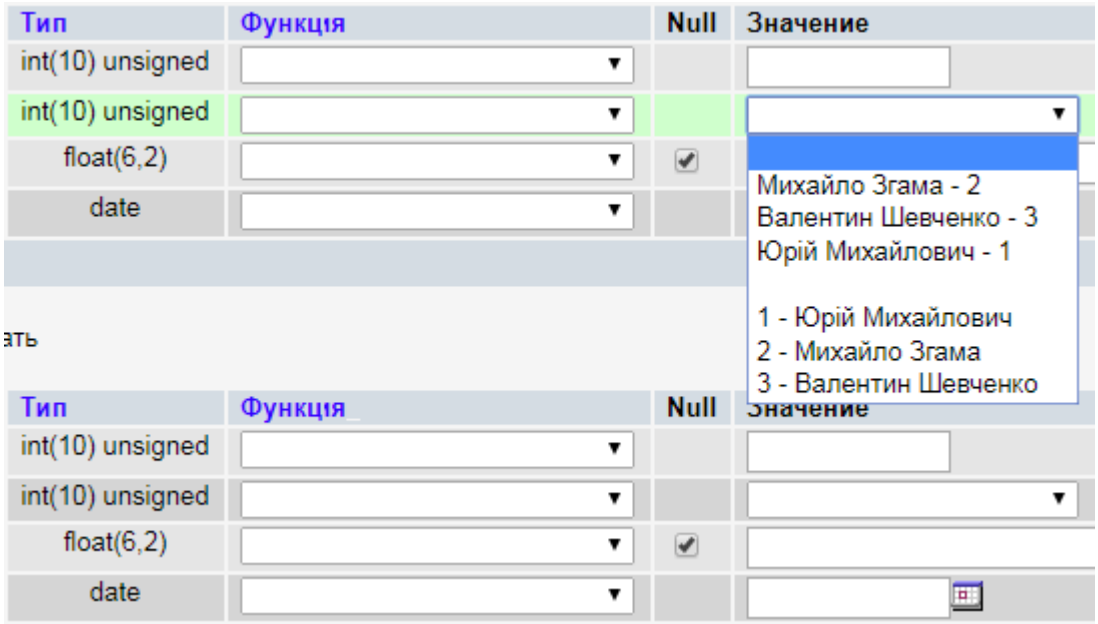


Рисунок 8.4 – Введення даних у таблиці, що пов’язані за допомогою графічного інтерфейсу

Таким чином, було наповнено таблиці необхідними даними. Передбачивши всі можливі варіанти та обмеження, проста БД була побудована з виконанням необхідних умов. За необхідності її можна розширити, змінити чи доповнити іншими об’єктами та даними.

Додаток В

Приклад звіту з виконаної лабораторної роботи

Міністерство освіти та науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

ЗВІТ
ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №7
з навчальної дисципліни «Бази даних»
«Типи з’єднань в MySQL»

ВИКОНАЛА: студентка
групи КБ-2??
Шевченко Є. А.
ПЕРЕВІРИВ: викладач
Константинова Л. В.

Кропивницький 2025

МЕТА:Навчитись реалізовувати засоби SQL для виконання запитів із декількох зв’язаних таблиць.

Тема індивідуального завдання: “База даних для турагентства”.

Завдання:

Виконати різні запити на вибірку із зв’язаних таблиць як наведено в теоретичному матеріалі.

1) Створити 4 таблиці.
2) Заповнити даними.
3) Виконати об’єднання з використанням псевдонімів для таблиць:
   А) Декартів добуток;
   Б) Повне з’єднання;
   В) Перехресне з’єднання;
   Г) Внутрішнє з’єднання;
   Д) З’єднання по рівності;
   Є) Лівостороннє з’єднання;
4) Провести групування даних в визначеному порядку з використанням конструкції ORDER BY та ключових слів та провести агрегування.

Хід роботи:

1,2 Було створено деякі таблиці. В результаті всього ми маємо п’ять таблиць, заповнених даними:

Таблиця «Bookings»:

	BookingID	ClientID	TourID	Booking_date	Payment_status
Редагувати Копіювати Видалити	31	1	3	2024-05-05	оплачено
Редагувати Копіювати Видалити	32	2	3	2024-05-06	не оплачено
Редагувати Копіювати Видалити	33	3	3	2024-05-07	оплачено

Таблиця «Clients»:

	ClientID	Name	Email	Phone
Редагувати Копіювати Видалити	1	John Smith	john@example.com	+1234567890
Редагувати Копіювати Видалити	2	Emma Johnson	emma@example.com	+1987654321
Редагувати Копіювати Видалити	3	Michael Davis	michael@example.com	+1122334455

Таблиця «Countries»:

	ID	Countrie_name	Capital	Area	Population	Date_of_foundation
Редагувати Копіювати Видалити	233727	Україна	Київ	603500	41879904	1991-08-24
Редагувати Копіювати Видалити	293782	США	Вашингтон	9833520	331002651	1776-07-04
Редагувати Копіювати Видалити	10929819	Italy	Rome	301340	60483973	1861-03-17
Редагувати Копіювати Видалити	27367368	France	Paris	551695	67300000	843-08-01
Редагувати Копіювати Видалити	28378237	England	London	130279	55980000	1066-01-01
Редагувати Копіювати Видалити	47564567	Japan	Tokyo	377975	125584838	660-01-01
Редагувати Копіювати Видалити	49854578	Germany	Berlin	357022	83149300	1871-01-18
Редагувати Копіювати Видалити	57678768	Canada	Ottawa	9984670	41903825.5125	1867-07-01
Редагувати Копіювати Видалити	76775678	Australia	Canberra	7692024	25687041	1901-01-01
Редагувати Копіювати Видалити	86774875	Spain	Madrid	505990	47329981	1469-01-01
Редагувати Копіювати Видалити	98843767	China	Beijing	9596961	1401814396	1949-10-01

Таблиця «Regions»:

Name	Country	Main_city
New England	United States	Boston
Île-de-France	France	Paris
Bavaria	Germany	Munich
Lazio	Italy	Rome
Andalusia	Spain	Seville
Ontario	Canada	Toronto
New South Wales	Australia	Sydney
Kanto	Japan	Tokyo
Beijing-Tianjin-Hebei	China	Beijing

Таблиця «Tours»:

	TourID	Destination	Price	Duration	Start_date	End_date	Description
Редагувати Копіювати Видалити	1	Paris	1500.00	7	2024-06-15	2024-06-22	Explore the beautiful city of Paris.
Редагувати Копіювати Видалити	2	Rome	1800.00	10	2024-07-01	2024-07-10	Discover the ancient ruins and rich history of Rom...
Редагувати Копіювати Видалити	3	Tokyo	2200.00	14	2024-08-05	2024-08-19	Experience the vibrant culture and modern marvels ...
Редагувати Копіювати Видалити	4	Париж	1500.00	7	2024-06-15	2024-06-22	Досліджуйте прекрасне місто Париж.
Редагувати Копіювати Видалити	5	Рим	1800.00	10	2024-07-01	2024-07-10	Відкрийте для себе древні руїни і багату історію Р...
Редагувати Копіювати Видалити	6	Токіо	2200.00	14	2024-08-05	2024-08-19	Відчуйте живу культуру і сучасні дива Токіо.

3. Виконую з’єднання з використанням псевдонімів для таблиць

A:

```
SELECT * FROM equipment AS e, instruments AS i;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

Показати все | Число рядків: 25 Фільтрувати рядки: Шукати в таблиці

+ Параметри

equipment_id	equipment_name	type	state	person_id	equipment_person_id	instrument_id	instrument_name	description	person_id	instrument_person_id
1	Лазерний дальномер	Оптичні прилади	Новий	NULL	NULL	1	Калашников АК-47	Автомат Калашникова, російський автомат зброї, шир...	NULL	NULL
2	Польовий телефон	Зв'язок	Використаний	NULL	NULL	1	Калашников АК-47	Автомат Калашникова, російський автомат зброї, шир...	NULL	NULL
3	Навігаційний прилад	Оптичні прилади	Старий	NULL	NULL	1	Калашников АК-47	Автомат Калашникова, російський автомат зброї, шир...	NULL	NULL

Б:

```
SELECT * FROM Personal AS p LEFT JOIN List AS l ON p.person_id = l.person_id UNION ALL SELECT * FROM Personal AS p RIGHT JOIN List AS l ON p.person_id = l.person_id;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

Показати все | Число рядків: 25 Фільтрувати рядки: Шукати в таблиці Сортувати за ключем: Жодного

+ Параметри

person_id	name	rank	branch	service_start_date	item_id	item_name	quantity	location	person_id	list_person_id
1	John Smith	Капітан	Піхота	2010-05-15	NULL	NULL	NULL	NULL	NULL	NULL
2	Anna Johnson	Майор	Танкові війська	2008-10-20	NULL	NULL	NULL	NULL	NULL	NULL
3	Michael Williams	Прапорщик	Повітряні сили	2015-03-08	NULL	NULL	NULL	NULL	NULL	NULL
4	Jessica Brown	Старшина	Морська піхота	2012-09-03	NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	1	Боєприпаси 7.62 мм	1000	Сховище	NULL	NULL
NULL	NULL	NULL	NULL	NULL	2	Медичний набір	20	Медпункт	NULL	NULL
NULL	NULL	NULL	NULL	NULL	3	Рюкзаки	50	Склад	NULL	NULL
NULL	NULL	NULL	NULL	NULL	4	Камуфляжні костюми	200	Склад	NULL	NULL

В:

```
SELECT * FROM Personal CROSS JOIN List;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

Показати все | Число рядків: 25 Фільтрувати рядки: Шукати в таблиці Сортувати за ключем: Жодного

+ Параметри

person_id	name	rank	branch	service_start_date	item_id	item_name	quantity	location	person_id	list_person_id
1	John Smith	Капітан	Піхота	2010-05-15	1	Боєприпаси 7.62 мм	1000	Сховище	NULL	NULL
2	Anna Johnson	Майор	Танкові війська	2008-10-20	1	Боєприпаси 7.62 мм	1000	Сховище	NULL	NULL
3	Michael Williams	Прапорщик	Повітряні сили	2015-03-08	1	Боєприпаси 7.62 мм	1000	Сховище	NULL	NULL
4	Jessica Brown	Старшина	Морська піхота	2012-09-03	1	Боєприпаси 7.62 мм	1000	Сховище	NULL	NULL
1	John Smith	Капітан	Піхота	2010-05-15	2	Медичний набір	20	Медпункт	NULL	NULL
2	Anna Johnson	Майор	Танкові війська	2008-10-20	2	Медичний набір	20	Медпункт	NULL	NULL
3	Michael Williams	Прапорщик	Повітряні сили	2015-03-08	2	Медичний набір	20	Медпункт	NULL	NULL
4	Jessica Brown	Старшина	Морська піхота	2012-09-03	2	Медичний набір	20	Медпункт	NULL	NULL
1	John Smith	Капітан	Піхота	2010-05-15	3	Рюкзаки	50	Склад	NULL	NULL
2	Anna Johnson	Майор	Танкові війська	2008-10-20	3	Рюкзаки	50	Склад	NULL	NULL
3	Michael Williams	Прапорщик	Повітряні сили	2015-03-08	3	Рюкзаки	50	Склад	NULL	NULL
4	Jessica Brown	Старшина	Морська піхота	2012-09-03	3	Рюкзаки	50	Склад	NULL	NULL

Г:

```
SELECT * FROM Personal AS p INNER JOIN List AS l ON p.person_id = l.person_id;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

person_id	name	rank	branch	service_start_date	item_id	item_name	quantity	location	person_id	list_person_id

Д:

```
SELECT * FROM shootingskills AS s JOIN personal AS p ON s.person_id = p.person_id;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

skill_id	skill_name	accuracy	level	person_id	shooting_person_id	person_id	name	rank	branch	service_start_date

Є:

```
SELECT * FROM personal AS p LEFT JOIN shootingskills AS s ON p.person_id = s.person_id;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

Показати все | Число рядків: 25 Фільтрувати рядки: Шукати в таблиці Сортувати за

+ Параметри

person_id	name	rank	branch	service_start_date	skill_id	skill_name	accuracy
1	John Smith	Капітан	Піхота	2010-05-15	NULL	NULL	NULL
2	Anna Johnson	Майор	Танкові війська	2008-10-20	NULL	NULL	NULL
3	Michael Williams	Прапорщик	Повітряні сили	2015-03-08	NULL	NULL	NULL
4	Jessica Brown	Старшина	Морська піхота	2012-09-03	NULL	NULL	NULL

4.Провести групування даних в визначеному порядку з використанням конструкції ORDER BY та ключових слів та провести агрегування.

```sql
SELECT co.Countrie_name, COUNT(b.BookingID) AS TotalBookings
FROM Countries AS co
JOIN Bookings AS b ON co.ID = b.ClientID
GROUP BY co.Countrie_name
ORDER BY TotalBookings ASC;

```

MySQL повернула пустий результат (тобто нуль рядків). (Запит виконувався 0.0002 секунди.)

```
SELECT co.Countrie_name, COUNT(b.BookingID) AS TotalBookings FROM Countries AS co JOIN Bookings AS b ON co.ID = b.ClientID GROUP BY
co.Countrie_name ORDER BY TotalBookings ASC;
```

Профілювання [Порядкове редагування] [Редагувати] [Тлумачити SQL] [Створити PHP код] [Оновити]

Countrie_name	TotalBookings

Операції з результатами запиту

Створити подання

Контрольні запитання і відповіді на них

1. Які типи з’єднань існують в MySQL?У MySQL існують такі типи з'єднань: внутрішнє з'єднання (INNER JOIN), ліве з'єднання (LEFT JOIN або LEFT OUTER JOIN), праве з'єднання (RIGHT JOIN або RIGHT OUTER JOIN) та повне з'єднання (FULL JOIN або FULL OUTER JOIN).

2. Що означає операція Join у реляційній базі даних? Операція JOIN у реляційній базі даних об'єднує рядки з двох або більше таблиць на основі спільних значень у вказаних стовпцях.

3. Яке з’єднання називають внутрішнім? Внутрішнє з'єднання (INNER JOIN) називається так, оскільки воно повертає тільки ті рядки, для яких є спільні значення в обох таблицях.

4. Яке з’єднання називають зовнішнім? Зовнішнє з'єднання (OUTER JOIN) називається так, оскільки воно повертає всі рядки однієї таблиці, а також ті рядки іншої таблиці, для яких немає відповідних значень у першій таблиці.

5. Яке з’єднання називають перехресним? Перехресне з'єднання (CROSS JOIN) повертає декартовий добуток рядків з усіх таблиць у запиті, тобто кожен рядок першої таблиці комбінується з кожним рядком другої таблиці.

6. Що таке псевдоніми (alias)? Псевдоніми (alias) - це тимчасові назви, які можна надати стовпцям або таблицям у запиті, щоб зробити їх ідентифікатори більш зрозумілими або коротшими.

7. Агрегування даних. Які функції агрегування розглядаються? Функції агрегування включають такі: SUM(), AVG(), COUNT(), MAX(), MIN() та інші. Вони дозволяють обчислювати суми, середнє значення, кількість рядків, максимальне та мінімальне значення у вибірці даних.

8. На які з'єднання поділяється зовнішнє з'єднання? Зовнішнє з'єднання поділяється на ліве з'єднання (LEFT JOIN або LEFT OUTER JOIN), праве з'єднання (RIGHT JOIN або RIGHT OUTER JOIN).

Висновок:

Застосовуючи різні відповідні команди для різних типів з’єднань я навчилась реалізовувати засоби SQL для виконання запитів на вибірку з декількох зв’язаних таблиць. Застосовувала при цьому функції агрегування, групувала дані, сортувала, застосовувала псевдоніми.

Рекомендовані джерела інформації

1. Бази даних: навч. посіб. / Босько В.В., Константинова Л.В., Поліщук Л.І., Конопліцька-Слободенюк О.К.; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2024. – 226 с.

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

3. Сидоренко В.В., Константинова Л.В., Смірнов С.А. Організація баз даних Навчальний посібник. - Кропивницький: ЦНТУ, 2018. – 274 с. [Електронний ресурс] - Режим доступу: http://dspace.kntu.kr.ua/jspui/bitstream/123456789/10527/1/NavPosOBD.pdf (дата звернення: 7.11.2023).

4. Пасічник В.В., Резніченко В.А. Організація баз даних та знань.-К: Видавнича група BHV, 2006.-384с.:іл.

5. Моделювання даних та маніпулювання даними /Навчальні ресурси СумДУ University online learning ecosystem. СумГУ, 2015 [Електронний ресурс] - Режим доступу: https://elearning.sumdu.edu.ua/free_content/lectured:a8104441b8e00905159c1ff04257b014dd456247/20151109195846/162252/index.html (дата звернення: 20.03.2024).

6. Пєтух А.М., Романюк О.В., Романюк О.Н. Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних, ВНТУ, 2016 [Електронний ресурс] - Режим доступу: https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/index.htm (дата звернення: 12.03.2024).

7. Клієнт-серверна архітектура. QATestLab. 28.05.2020. [Електронний ресурс] - Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 20.03.2024).

8. Database Replication 101: Everything You Need To Know. January 24th, 2024. [Електронний ресурс] - Режим доступу:

https://www.astera.com/type/blog/database-replication-101/ (дата звернення: 20.03.2024).
9. ISO/IEC 9075-1:2023(en). Information technology - Database languages SQL [Електронний ресурс] - Режим доступу: https://www.iso.org/obp/ui/en/#iso:std:76583:en (дата звернення: 25.03.2024).
10. Features of SQL databases / LinkedIn. 27.05.2023. [Електронний ресурс]/ - Режим доступу: https://www.linkedin.com/pulse/features-sql-databases-database-designer-sql-mysql (дата звернення: 24.03.2024).
11. Stephane Faroult with Peter Robson «The Art of SQL». Sebastopol, Calif.: O'Reilly Media Inc. (2006).
12. 10 SQL Skills for Programmers and Developers / Indeed. 11.03.2023 [Електронний ресурс] - Режим доступу: https://www.indeed.com/career-advice/resumes-cover-letters/sql-skills (дата звернення: 25.03.2024).
13. Paul DuBois, MySQL, 5th Edition, Mar 29, 2013 by Addison-Wesley Professional.
14. SQL Підручник [Електронний ресурс] – Режим доступу: https://w3schoolsua.github.io/sql/index.html#gsc.tab=0 (дата звернення: 7.04.2024).
15. Методичні вказівки до лабораторних робіт з дисципліни «Бази даних і знань» для студентів напряму підготовки «Управління інформаційною безпекою» / [уклад. : Ю. Є. Яремчук, Д. П. Присяжний, І. О. Дьогтєва, О. В. Салієва] ; ВНТУ [Електронний ресурс] - Режим доступу: https://web.posibnyky.vntu.edu.ua/fmib/37yaremchuk_metodvkaz_labrob_bazi_danih_znan_upravlinnya_informacijnoyu_bezpekoyu/07.html (дата звернення: 7.04.2024).
16. Documentation. MySQL, 2024. [Електронний ресурс] - Режим доступу: https://dev.mysql.com/doc/ (дата звернення: 21.03.2024).
17. Бази даних: метод. вказівки до виконання комп'ютерного практикуму для студентів спеціальності "Електронні комунікації та радіотехніка " /

Уклад.: Суліма С.В., Глоба Л.С., Скулиш М.А.. – К.: КПІ ім. Ігоря Сікорського, 2023. – 54 с.

18. Efficient MySQLPerformance: Best Practices and Techniques. 1st Ed.DanielNichter. O'Reilly, 2021. 276p.

19. Learning MySQL: Get a HandleonYourData. 2nd Ed.Vinicius M. Grippa, SergeyKuzmichev.O'Reilly, 2021. 550 p.

20. 97 Things Every Data Engineer Should Know: Collective Wisdom from the Experts. Tobias Macey. O'Reilly, 2021. 264 p.

21. Beginning Spring Data: Data Access and Persistence for Spring Framework 6 and Boot 3 1st ed. Edition. Andres Sacco.Apress, 2022. 439 p.

22. Learning PHP, MySQL & JavaScript. A Step-by-Step Guide to Creating Dynamic Websites. 6th Ed.Robin Nixon.O'Reilly, 2021. 826 p.

23. PHP & MySQL: Novice to Ninja 7th Edition. Tom Butler. SitePoint, 2022. 686 p.

24. PostgreSQL Query Optimization: The Ultimate Guide to Building Efficient Queries. Anna Bailliekova, Henrietta Dombrovskaya, Boris Novikov. Apress, 2021. 344 p.