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 «Організація баз даних» (2 частина) для студентів денної та заочної форми навчання за спеціальностями 123 «Комп'ютерна інженерія», 125 «Кібербезпека»

ЗАТВЕРДЖЕНО
кафедрою кібербезпеки та
програмного забезпечення,
протокол від 11.03.2020 року № 11

КРОПИВНИЦЬКИЙ
2020

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Організація баз даних» (2 частина) для студентів денної та заочної форми навчання за спеціальностями 123 «Комп'ютерна інженерія», 125 «Кібербезпека» / уклад. В.В. Босько, Л.В. Константинова — Кропивницький: ЦНТУ, 2020. — 60 с.

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

Рецензенти: Смірнов О. А., д-р техн. наук, професор;
Мелешко Є. В., канд. техн. наук.

Вступ

В методичних рекомендаціях представлено теоретичний і практичний матеріал для роботи з базами даних за допомогою СКБД MySQL. Дані методичні вказівки продовжують методичні вказівки до лабораторних робіт з навчальної дисципліни «Організація баз даних» (1 частина) для студентів денної та заочної форми навчання. Приводиться багато завдань, практичних прикладів та ілюстрацій, які допомагають краще засвоїти викладений матеріал з дисципліни «Організація баз даних». Розглядається робота з БД 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).*

Теми лабораторних робіт, що розглядаються та оцінювання знань

Пропонується виконати лабораторні роботи за наступними темами:

Теми лабораторних робіт	Д.ф. (години)
1. Керування базами даних за допомогою SQL. Застосування операторів DDL для роботи з БД.	2
2. Керування базами даних за допомогою SQL. Застосування операторів DML для роботи з БД.	2
3. Керування базами даних за допомогою SQL. DQL. Вибірка, читання даних, отримання підсумкових значень.	2
4. Керування базами даних за допомогою SQL. DCL.	2
5. Керування базами даних за допомогою SQL. TCL.	2
6. Первинні та зовнішні ключі, обмеження, керування зв'язками між таблицями за допомогою SQL.	2
7. Типи з'єднань в MySQL.	2
Всього:	14

Форма підсумкового контролю іспит.

Максимальну кількість балів студент може одержати у випадку відвідування всіх лекцій, лабораторних занять, виконання і захисту завдань з самостійної роботи у встановлений термін проходження контролю.

При виконанні та захисті лабораторних робіт після встановленого терміну, одержані бали перераховуються з коефіцієнтом: для самостійної роботи студента -0,3; лабораторної роботи -0,7.

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

Вибравши предметну область, над якою будете працювати, ви повинні виконати завдання до лабораторних робіт, а також відповісти на питання в кінці кожної лабораторної роботи. Звіт повинен містити хід виконання завдань а також графічні матеріали, що підтверджують виконання цих завдань.

MySQL — вільна система керування реляційними базами даних.

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* (*Тransaction Control Language*). Ці SQL команди дозволяють визначити результат транзакції.

Доступ до БД може здійснюватися:

- Через скрипти сайту;

- За допомогою програми phpMyAdmin.

Для виконання лабораторних робіт достатньо ознайомитись з панеллю керування phpMyAdmin.

PhpMyAdmin - це програмне забезпечення, написане на PHP, яке забезпечує повноцінну, у тому числі віддалену, роботу з базами даних MySQL через браузер (рекомендується використовувати систему Denwer, яку можна завантажити з веб-сайту: http://www.denwer.ru).

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).

Також можна застосувати Open Server - досить функціональний і простий у використанні продукт (https://ospanel.io/docs/, https://php-start.com/blog/open-server-intallation).

Лабораторна робота №1

Тема: Керування базами даних за допомогою SQL. Застосування операторів DDL для роботи з БД.

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

Зміст роботи за варіантом індивідуального завдання:

1. Створити нову базу даних за допомогою DDL.
2. За допомогою DDL створіть 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. Видаліть таблицю із своєї БД за допомогою DDL-операторів.
5. Створіть індекс у таблиці БД.
6. Створіть унікальний індекс для таблиці своєї БД.
7. Видаліть створений раніше індекс.
8. Створіть таблицю так, щоб для двох полів не можна було б встановити невизначене значення, та встановіть первинний ключ за допомогою необхідного обмеження.

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.

Створення бази даних виконується за допомогою оператора **CREATE DATABASE**.

Синтаксис:

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

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

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

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

Якщо при створенні таблиці ці параметри не вказуються, то кодування і порядок сортування новостворюваної таблиці беруться зі значень, зазначених для всієї БД. Якщо заданий параметр *CHARACTER SET*, але не заданий параметр *COLLATE*, то використовується стандартний порядок сортування. Якщо заданий параметр *COLLATE*, але не заданий *CHARACTER SET*, то кодування визначає перша частина імені порядку сортування в *COLLATE*.

Кодування, задане в *CHARACTER SET*, повинне підтримуватися сервером (latin1 або sjis), а порядок сортування повинен бути допустимим для поточного кодування.

Приклади:

Наступний приклад створює БД "my_db":

CREATE DATABASE `my_db`

або

CREATE DATABASE `my_db` 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 бази даних:

- Максимальна довжина імені не повинна перевищувати 64 символи;

Дозволені будь-які символи, які допускаються в імені каталогу, за винятком / (слеш) і. (точка). Але не можна використовувати символи ASCII (0) і ASCII (255).

Створення БД за допомогою PhpMyAdmin:

Для цього необхідно перейти на вкладку Databases і в формі Create new database вказати назву БД та її кодування.

На рис. 1 зображено створення БД з назвою “my_db”:

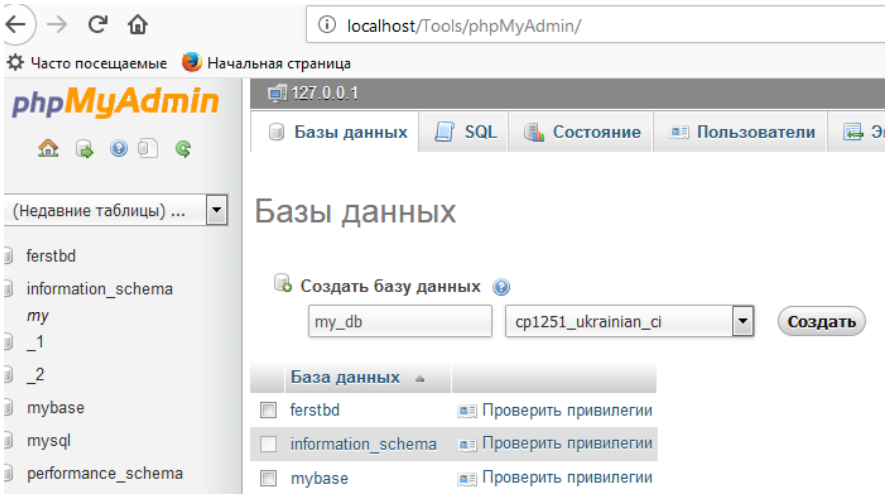


Рисунок 1 - Створення нової БД

Створення таблиці в БД проводиться командою **CREATE TABLE**.
Синтаксис:
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
[(create_definition,...)]
[table_options] [select_statement]
tbl_name - Визначає ім'я таблиці, яку буде створено в поточній БД. Починаючи з MySQL 3.22 введена можливість явно вказати БД, в якій буде створена нова таблиця, за допомогою синтаксису db_name.tbl_name.
TEMPORARY - Цей параметр використовується для створення тимчасової таблиці з ім'ям tbl_name протягом лише поточного сценарію. По закінченню виконання сценарію створена таблиця видаляється. Дана можливість з'явилася в MySQL 3.23. В MySQL 4.0.2 для створення тимчасових таблиць потрібні привілеї створення тимчасових таблиць.
IF NOT EXISTS - Якщо зазначений цей параметр і проводиться спроба створити таблицю з дублюючим ім'ям (тобто таблиця з таким ім'ям у поточній БД вже є), то таблиця створена не буде і повідомлення про помилку не з'явиться. В іншому випадку таблиця також створена не буде, але команда викличе помилку. Слід зазначити, що при створенні порівнюються тільки імена таблиць. Внутрішні структури не порівнюються.
create_definition - Визначає внутрішню структуру створюваної таблиці (назви і типи полів, ключі, індекси і т.д.)
Можливі синтаксиси **create_definition:**
col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]
PRIMARY KEY (index_col_name,...)
KEY [index_name] (index_col_name,...)
INDEX [index_name] (index_col_name,...)
UNIQUE [INDEX] [index_name] (index_col_name,...)
FULLTEXT [INDEX] [index_name] (index_col_name,...)

[CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...) [reference_definition]
CHECK (expr)

col_name - Визначає ім'я стовпця в створюваній таблиці;

type - Задає тип даних для стовпця ім'я_стовпця

Можливі значення параметра type:

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

Дуже мале ціле число. Діапазон зі знаком від -128 до 127. Діапазон без знаку від 0 до 255.

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

Мале ціле число. Діапазон зі знаком від -32768 до 32767. Діапазон без знаку від 0 до 65535.

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

Ціле число середнього розміру. Діапазон зі знаком від -8388608 до 8388607. Діапазон без знаку від 0 до 16777215.

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

Ціле число нормального розміру. Діапазон зі знаком від -2147483648 до 2147483647. Діапазон без знаку від 0 до 4294967295.

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

Синонім для INT

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

Велике ціле число. Діапазон зі знаком від -9223372036854775808 до +9223372036854775807. Діапазон без знаку від 0 до 18446744073709551615.

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

Число з плаваючою крапкою подвоєної точності нормального розміру. Допустимі значення: від -1,7976931348623157 E +308 до -2,2250738585072014 E-308, 0, і від 2, 2250738585072014E-308 до +1,7976931348623157 E +308. Якщо вказаний атрибут UNSIGNED, негативні значення неприпустимі.

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

Синонім для DOUBLE

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

Мале число з плаваючою крапкою звичайної точності. Допустимі значення: 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]**

«Неупаковане» число з плаваючою крапкою. Веде себе подібно колонці CHAR, яка містить цифрове значення. Термін «неупаковане» означає, що число зберігається у вигляді рядка і при цьому для кожного десяткового знака використовується один символ.

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

Рядок фіксованої довжини, при зберіганні завжди доповнюється пробілами в кінці рядка до заданого розміру. Діапазон аргументу length становить від 0 до 255 символів (від 1 до 255 у версіях, що передують MySQL 3.23). Кінцеві прогалини видаляються при виведенні значення. Якщо не заданий атрибут чутливості до регістру BINARY, то величини CHAR сортуються і порівнюються як незалежні від регістра відповідно до встановленого за замовчуванням алфавіту.

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

Рядок змінної довжини

- **DATE**

Дата. Підтримується інтервал від '1000-01-01 'до '9999-12-31'. MySQL виводить значення DATE у форматі 'YYYY-MM-DD', але можна встановити значення в стовпець DATE, використовуючи як рядки, так і числа.

- **TIME**

Час. Інтервал від '-838:59:59' до '838: 59:59 '. MySQL виводить значення TIME у форматі 'HH: MM: SS', але можна встановлювати значення в стовпці TIME, використовуючи як рядки, так і числа.

- **TIMESTAMP**

Часова мітка. Інтервал від '1970-01-01 00:00:00 'до деякого значення часу в 2037. MySQL виводить значення TIMESTAMP у форматах YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD або YYMMDD.

- **DATETIME**

Комбінація дати і часу. Підтримується інтервал від '1000-01-01 00:00:00 'до '9999-12-31 23:59:59'. MySQL виводить значення DATETIME у форматі 'YYYY-MM-DD HH: MM: SS', але можна встановлювати значення в стовпці DATETIME, використовуючи як рядки, так і числа.

- **TINYBLOB**

Стовпець типу BLOB з максимальною довжиною 255 (2 ^ 8 - 1) символів.

- **BLOB**

Стовпець типу BLOB з максимальною довжиною 65535 (2^16 - 1) символів.

- **MEDIUMBLOB**

Стовпець типу BLOB з максимальною довжиною 16777215 (2^24 - 1) символів.

- **LONGBLOB**

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

- **TINYTEXT**

Стовпець типу TEXT з максимальною довжиною 255 (2 ^ 8 - 1) символів.

- **TEXT**

Стовпець типу TEXT з максимальною довжиною 65535 (2^16 - 1) символів.

- **MEDIUMTEXT**

Стовпець типу TEXT з максимальною довжиною 16777215 (2^24 - 1) символів.

- **LONGTEXT**

Стовпець типу TEXT з максимальною довжиною 4294967295 (2 ^ 32 - 1) символів. Слід враховувати, що в даний час протокол передачі даних сервер / клієнт і таблиці MyISAM мають обмеження 16 Мб на переданий пакет / рядок таблиці, тому поки не можна використовувати цей тип даних у його повному діапазоні.

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

Перерахування. Перераховується тип даних. Об'єкт рядка може мати тільки одне значення, вибране із заданого списку величин 'value 1', 'value 2', ..., NULL або спеціальна величина помилки. Список ENUM може містити максимум 65535 різних величин.

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

Набір. Об'єкт рядка може мати нуль або більше значень, кожне з яких має бути вибрано із заданого списку величин 'value 1', 'value 2', ... Список SET может містити максимум 64 елементи.

[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 вставці нового запису в таблицю поле з цим атрибутом автоматично отримає числове значення, більше самого великого значення для цього поля в поточний момент часу на 1. Дана можливість зазвичай використовується для генерування унікальних ідентифікаторів рядків. Стовпець, для якого застосовується атрибут AUTO_INCREMENT, повинен мати цілочисельний тип. У таблиці може бути тільки один стовпчик з атрибутом AUTO_INCREMENT. Так само цей стовпець повинен бути проіндексований. Відлік послідовності чисел для AUTO_INCREMENT починається з 1. Це можуть бути лише додатні числа.

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,
SFAM CHAR (20),
SIMA CHAR (10),
SOTCH CHAR (15));

Порядок розташування полів у таблиці визначається тим, в якій послідовності вони вказані в команді створення таблиці.

Команда ALTER TABLE-засіб для зміни визначення існуючої таблиці, якою можна додавати поля до існуючої таблиці(ADD), видаляти поля(DROP) чи змінювати їх розмір (ALTER). ALTER TABLE не діє, 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 по полю, яке зберігає прізвище студента:

CREATE INDEX SFAMIDX ON STUDENTS (SFAM);

Для створення унікальних індексів використовується ключове слово UNIQUE у команді CREATE INDEX. Фактично такий індекс буде первинним ключем таблиці.

Створити унікальні індекси можна так:

CREATE UNIQUE INDEX
SNAMIDX ON STUDENTS (SNAM);

Ця команда не виконається, якщо в полі SNAM трапляться неунікальні значення.

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

DROP INDEX SFAMIDX
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.

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 7:2

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.

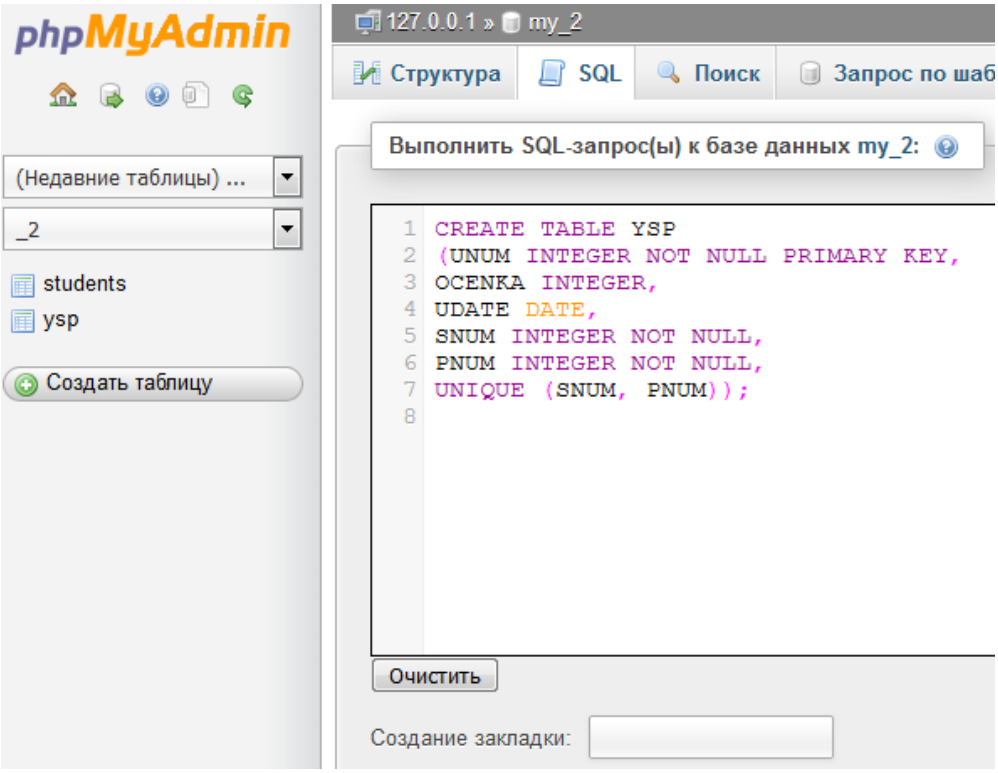


Рисунок 2

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 SOT SOTCH VARCHAR(20);

Поле з ім'ям SOT зміниться на SOTCH.

Контрольні питання до лабораторної роботи 1:

1. **Що представляє собою SQL?**
2. **На які підгрупи команд поділяється SQL?**
3. **Які можливості DDL?**
4. **На яких командах базується DDL?**
5. **За допомогою якої команди можна створити БД?**
6. **За допомогою якої команди можна створити таблицю?**
7. **За допомогою якої команди можна видалити таблицю?**
8. **За допомогою якої команди можна змінити визначення об'єкта БД?**
9. **Що означає команда CREATE TABLE?**
10. **Що означає команда ALTER TABLE?**
11. **Що означає команда DROP TABLE?**
12. **Які дії над таблицею треба виконати перед тим як видалити її?**
13. **Що називають індексом?**
14. **Що означає обмеження даних?**
15. **Коли не можна створити унікальний індекс в таблиці з даними?**
16. **Що означає обмеження NOT NULL у кінці оголошення імені таблиці?**
17. **Як переіменувати поле таблиці?**

Лабораторна робота №2

Тема: Керування базами даних за допомогою SQL. Застосування операторів DML для роботи з БД.

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

Зміст роботи за варіантом індивідуального завдання:

1. За допомогою DML у вікні SQL побудувати запит, завдяки якому можна було б вносити нові записи до вашої таблиці. Два поля таблиці повинні заповнюватись певною інформацією, третє поле повинно мати значення NULL, а останні -за замовчанням.
2. Вилучіть з спеціально для цього створеної таблиці всі записи за допомогою SQL.
3. Вилучіть з таблиці ті записи, які у конкретному полі мають певне значення за допомогою DML.
4. За допомогою DML змініть ті значення де-якого поля на «пусто», яким у полі Дата відповідає значення 01.01.2020.
5. Зменшити за допомогою DML значення числового поля на 25% в таблиці, якщо, його значення, наприклад, більше або дорівнює 32500.

Теоретичні відомості

Запити на зміну за допомогою SQL

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

Додавання інформації в БД

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

Приклад 1 - для додавання запису в таблицю викладачів TEACHERS, можна скористатись наступним виразом:

```
INSERT INTO TEACHERS
VALUES (‘4006’, ‘Федченко’, ‘Світлана’, ‘Геннадієвна’, ‘2019-09-09’);
```

Можна вказати стовпці, в які необхідно здійснити вставку значень.

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 замовчанням.

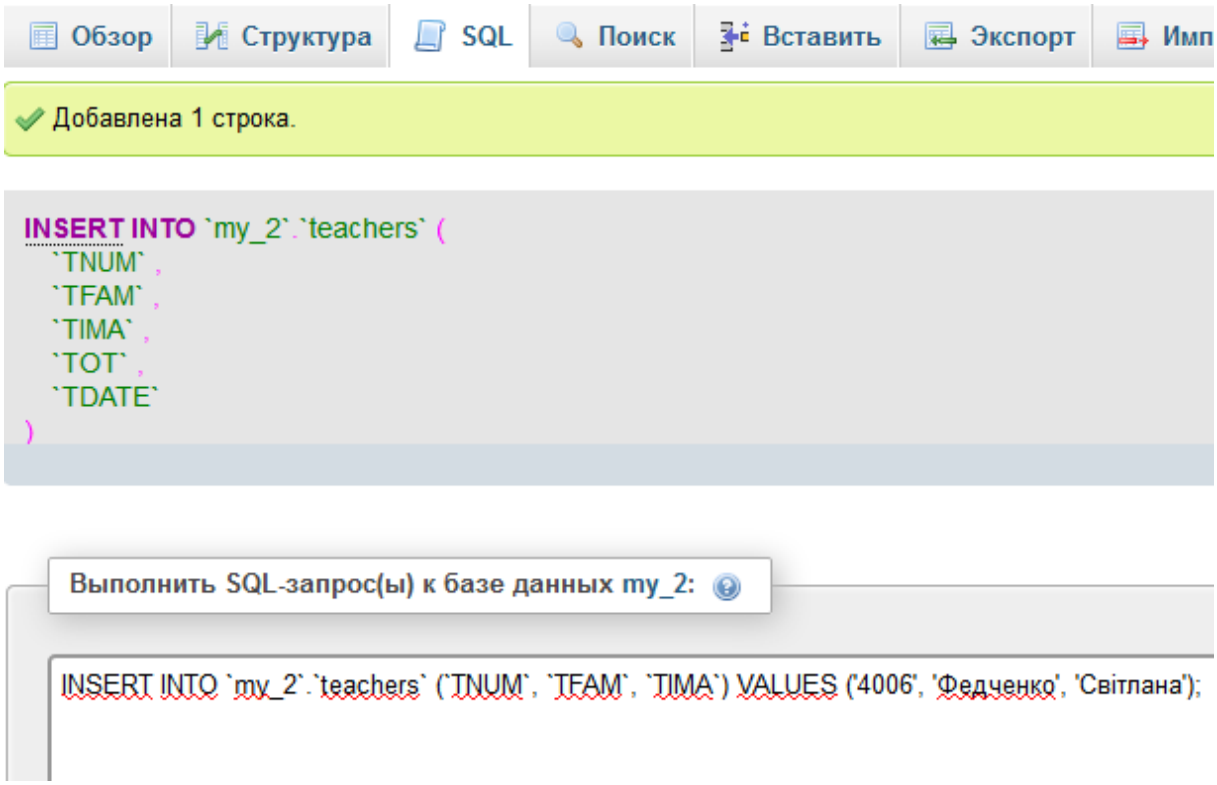


Рисунок 3

На рисунку 3 отримали запит у режимі SQL, що додає інформацію до таблиці TEACHES з 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 враховувати, те, що команда може видаляти тільки цілі записи таблиці, а не індивідуальні значення де-яких полів.

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

DELETE FROM ‘STUDENTS’;

Для визначення рядків, які треба видалити за 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 4 зображена таблиця STUDENTS.

Для видалення таблиці необхідно застосувати SQL-запит, що зображено на рисунку 5. Необхідно підтвердити виконання дії, якщо необхідно. Після виконання запиту в повідомленні буде вказано кількість видалених рядків та час виконання операції.

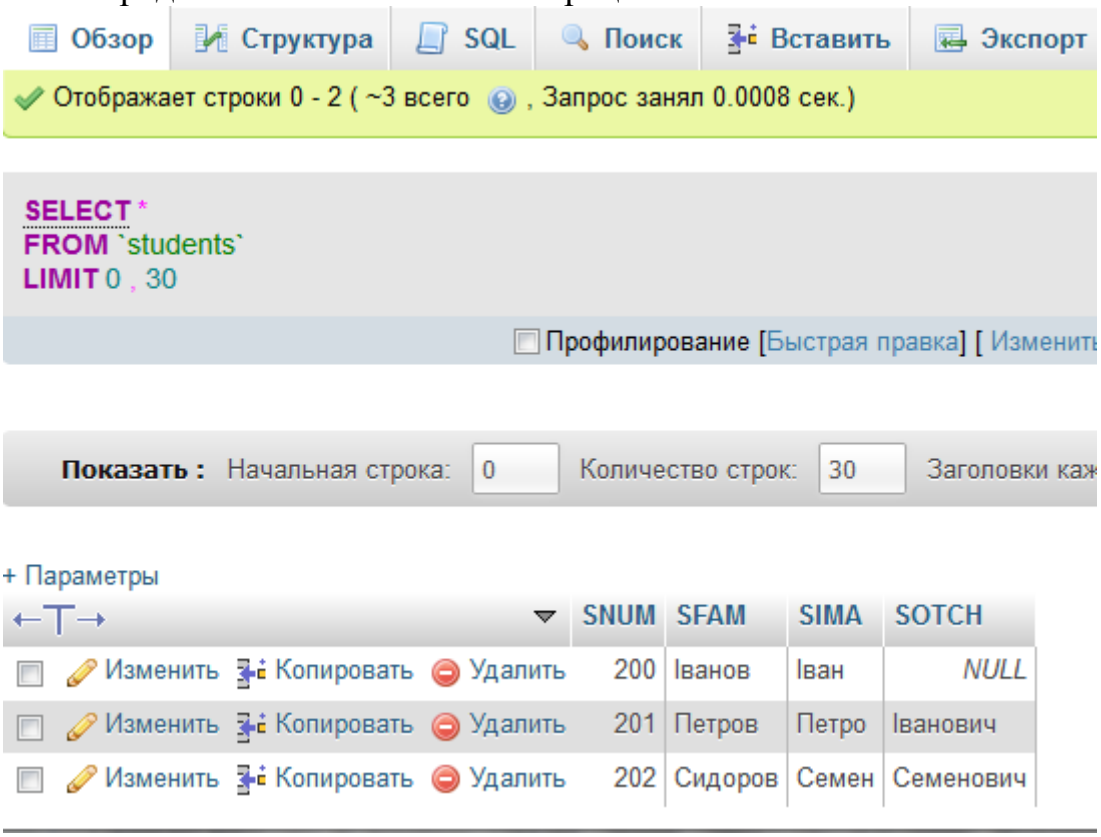


Рисунок 4 - Таблиця STUDENTS

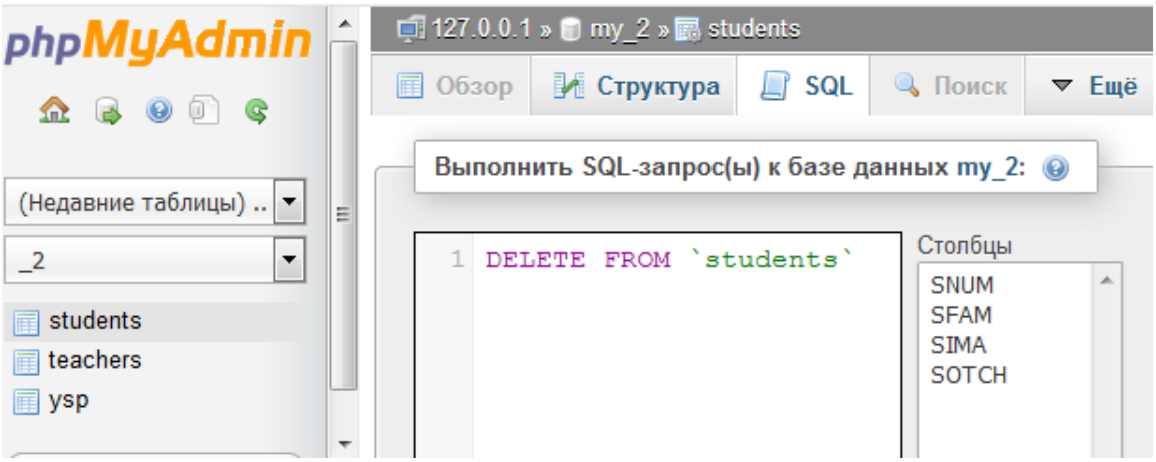


Рисунок 5 – Запит на видалення рядків з таблиці

Зміна існуючих даних

Можливість зміни всіх або деяких значень в таблиці реалізується за допомогою команди UPDATE. В ній вказується ім’я таблиці, яка використовується та слово SET, яке визначає зміну, яка 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 6). Наприклад:

```
UPDATE ‘USP’
SET ‘OCENKA’=5;
WHERE PNUM=2003;
```

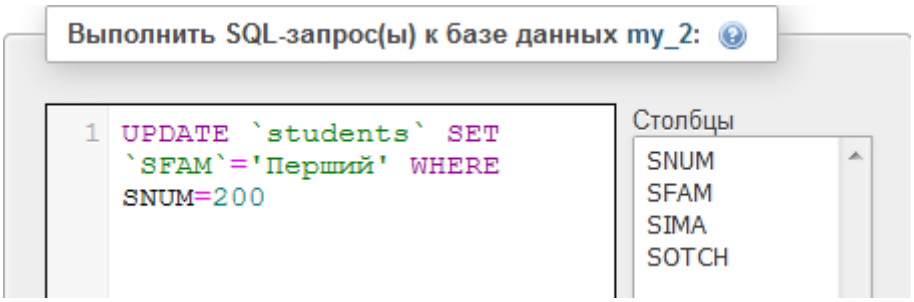


Рисунок 6 – Запит з предікатом

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*2;
```

Команда збільшує значення стипендії в 2 рази.

До недоліків команди UPDATE можна віднести неможливість посилатися на таблицю, яка задіяна 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;
```

Контрольні питання до лабораторної роботи 2:

1. **Що представляє собою SQL?**
2. **На які підгрупи команд поділяється SQL?**
3. **Що представляє собою DML?**

4. **Які команди складають DML?**
5. **Які запити на зміну ви знаєте?**
6. **За допомогою якої інструкції можна створити запит на додавання записів у таблицю?**
7. **Що означає команда UPDATE у запитах?**
8. **Яку інструкцію SQL повинен мати запит на вилучення записів?**
9. **Що означає команда INSERT INTO у запитах?**
10. **Що означає команда DELETE у запитах?**
11. **За допомогою якої команди SQL є можливість зміни всіх або деяких значень в таблиці?**

Лабораторна робота № 3

Тема: Керування базами даних за допомогою SQL. DQL. Вибірка, читання даних, отримання підсумкових значень.

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.

Зміст роботи за варіантом індивідуального завдання:

1. За допомогою SQL- запиту вивести всі дані з певної таблиці. В інструкції застосувати необов'язкове скорочення у вигляді символа «зірочка» (*).
2. Вивести за допомогою SQL- запиту інформацію з двох полів таблиці.
3. Створити SQL- запит для вибірки даних поля вашої таблиці таким чином, щоб результат не мав дублікатів.
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. Отримайте інформацію з таблиці з даними 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 таким чином, 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).

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 7.

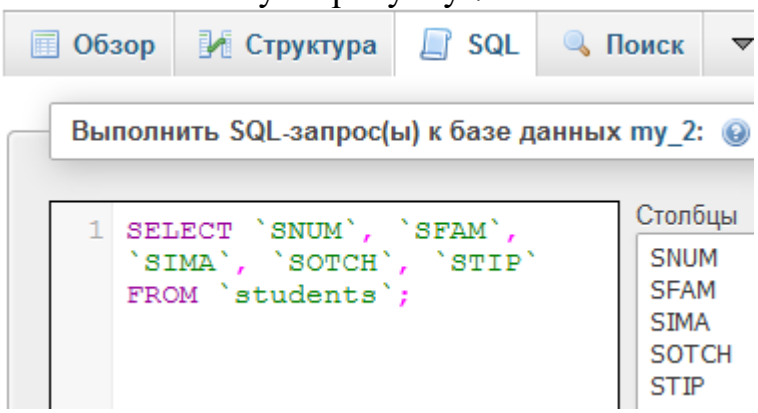


Рисунок 7

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 8

Якщо є необхідність уникнути дублювання застосовують DISTINCT-аргумент.

Наприклад є таблиця (рисунок 8).

Приклад 3:

SELECT DISTINCT STIP FROM STUDENTS;

Цю інструкцію застосовують для отримання списку результатів без дублікатів.

На рисунку 9 отримали результат виконання такого запиту.

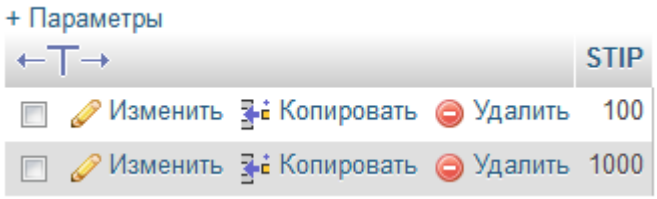


Рисунок 9

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.

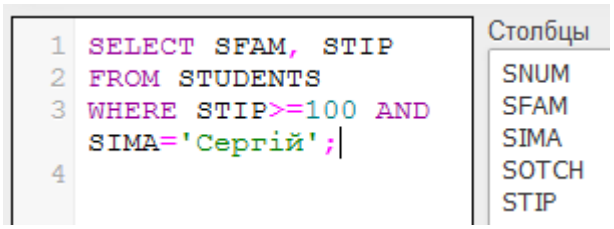
Приклад4:

SELECT SFAM, STIP
FROM STUDENTS
WHERE STIP=25.50;

виведуться тільки ті прізвища та розмір стипендії студентів, для яких у полі STIP буде значення 25.50.

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 10).



Результат:

SFAM	STIP
Струмеленко	100

Рисунок 10

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 і їх результати зображено на рисунку 11.

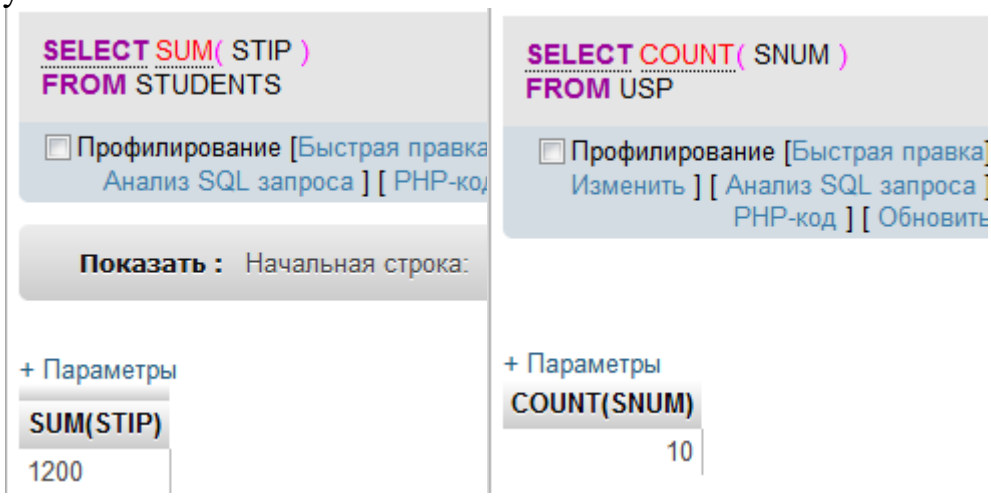


Рисунок 11

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. Що 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.

Приклад 8 - щоб отримати інформацію про учасників конференції ВНЗ 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 запит уз прикладу 8 буде таблиця про учасників конференції, що містить інформацію з двох таблиць, це видно на рисунку 12.

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

Рисунок 12

Приклад 9:

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

Результат на рисунку. 13.

+ Параметры

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

Рисунок 13

Важливою особливістю запитів SQL є їх спроможність визначати зв’язки між таблицями і 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 14):

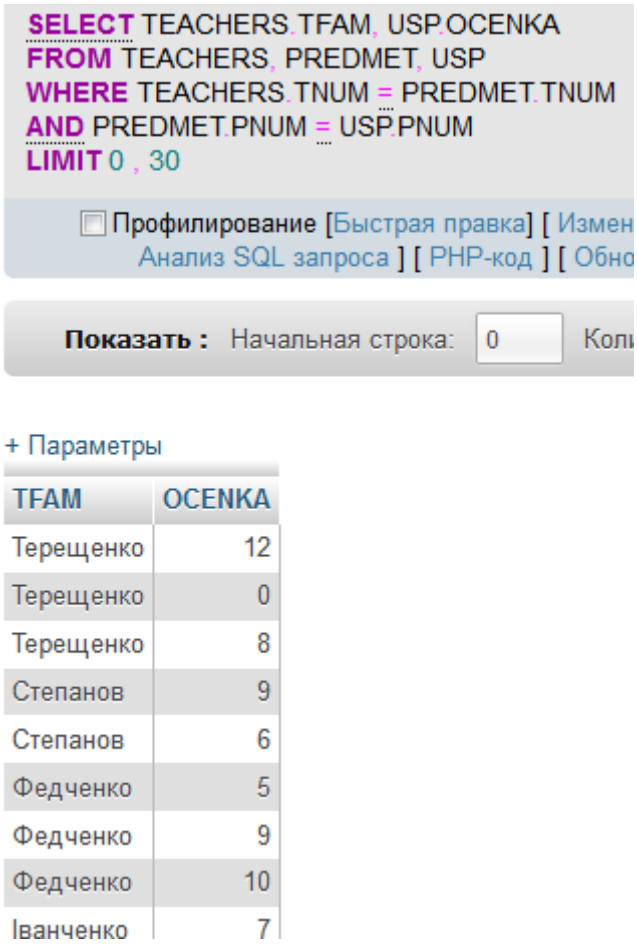


Рисунок 14

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 предікаті.

Результат виконання запиту з підзапитом з прикладу 11 зображено на рисунку 15.

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

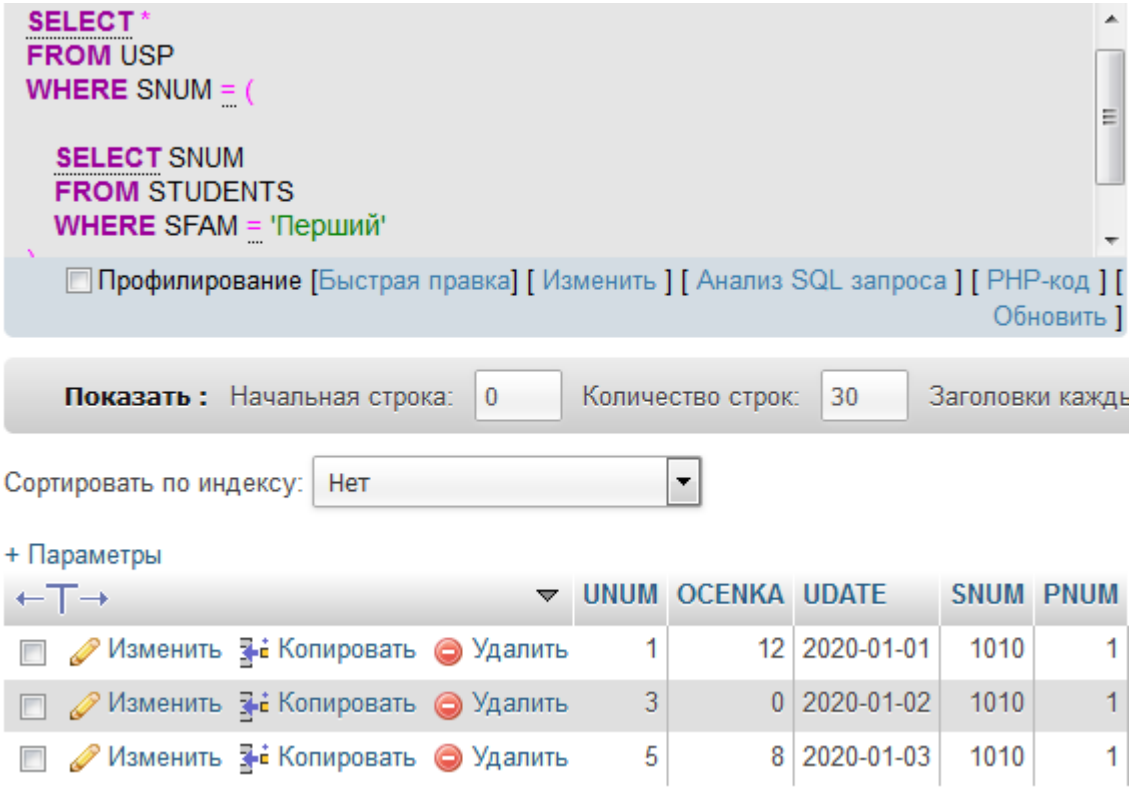


Рисунок 15

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=2020-06-10);

В цьому випадку підзапит вибере всі оцінки за 10.06.2009. 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=2020-01-03);

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

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.

Оператор HAVING

Оператор HAVING має призначення, подібне до оператора WHERE, але використовується з агрегатними даними. Наприклад:

SELECT Customers.Customeri, COUNT (Orders.Orderi) FROM Customers INNER JOIN Orders ON Customers.Customeri = Orders.Customeri GROUP BY Customers.Customeri HAVING COUNT(Orders.Orderi) >= 10

Цей запит аналогічний попередньому, але в результуючий набір даних включені тільки ті замовники, які розмістили десять або більше замовлень.

Оператор JOIN буде розглянуто в лабораторній роботі №7.

Контрольні питання до лабораторної роботи 3:

1. **Що означає DQL?**
2. **За допомогою якої інструкції в запитах здійснюється вибірка даних?**
3. **За допомогою якого символу в інструкції SQL-запитів можна вивести всі дані 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 означає інструкція SELECT...FROM...?**
7. **Для чого 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 вбудованими?**
12. **Які запити називають багатотабличними?**
13. **Для чого використовують оператор GROUP BY?**
14. **Чи відрізняється оператор HAVING від WHERE?**

Лабораторна робота № 4

Тема: Керування базами даних за допомогою SQL. DCL.

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

Зміст роботи за варіантом індивідуального завдання:

1. Створити нових користувачів вашої БД (з іменами user1, user2, user3).
2. User1 надати доступ до всієї інформації вашої БД.
3. User2 надати доступ тільки перегляд інформації БД.
4. User3 надати доступ 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 ...]

Для випадків, коли можуть знадобитися більш жорсткі обмеження, є 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 (Пользователи), як показано на рисунку 16.

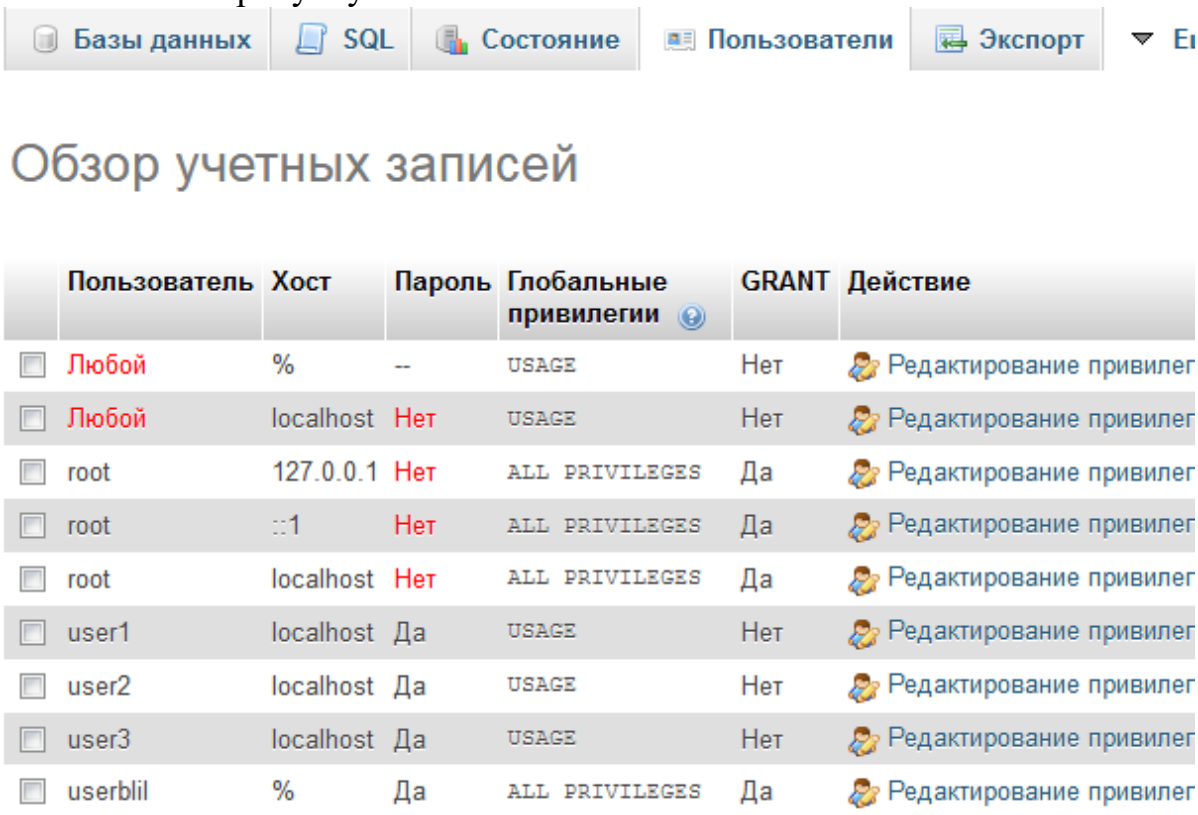


Рисунок 16

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

GRANT ALL PRIVILEGES ON *.* TO 'newuser'@'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).

Команди GRANT і REVOKE дозволяють системним адміністраторам створювати користувачів MySQL, а також надавати права користувачам або позбавляти їх прав на чотирьох рівнях привілеїв:

Глобальний рівень

Глобальні привілеї застосовуються до всіх баз даних на зазначеному сервері. Ці привілеї зберігаються в таблиці mysql.user.

Рівень бази даних

Привілеї бази даних застосовуються до всіх таблиць зазначеної бази даних. Ці привілеї зберігаються в таблицях mysql.db і mysql.host.

Рівень таблиці

Привілеї таблиці застосовуються до всіх стовпців зазначеної таблиці. Ці привілеї зберігаються в таблиці mysql.tables_priv.

Рівень стовпчика

Привілеї стовпчика застосовуються до окремих стовпців зазначеної таблиці. Ці привілеї зберігаються в таблиці mysql.columns_priv.

Короткий список деяких можливих варіантів прав доступу, які можуть отримати користувачі:

ALL PRIVILEGES - це дасть користувачеві MySQL повний доступ до заданої бази даних (якщо база даних не вказана, то до всіх);
CREATE - дозволяє створювати нові таблиці або бази даних;
DROP - дозволяє видаляти таблиці або бази даних;
DELETE - дозволяє видаляти рядки з таблиць;
INSERT - дозволяє додавати 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';

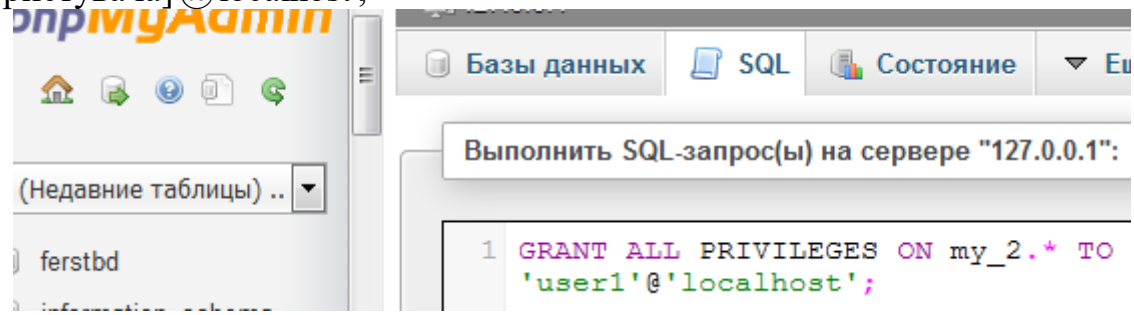


Рисунок 17

На рисунку 17 зображено запит, за допомогою якого, користувачу user1 надаються привілеї повного доступу до всіх таблиць БД my_2.

Перевірити результат запиту можливо на вкладці користувачів (Пользователи), знайшовши користувача необхідно вибрать Редагування привілеїв і вдосконалитись у виконаних діях (рисунок 18), також можливо редагувати потрібне.

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

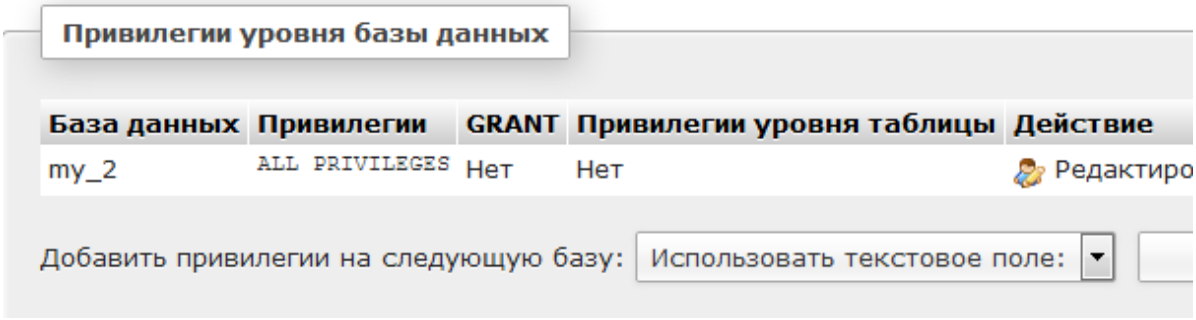


Рисунок 18

Не слід призначати привілеї ALTER звичайним користувачам. Це дає користувачеві 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 питання до лабораторної роботи 4:

1. **Що представляє собою DCL?**
2. **За допомогою якої команди можна створити нового користувача БД?**
3. **Яким чином надаються права користувачам БД?**
4. **Які рівні привілеїв існують?**
5. **Перелічіти основні привілеї користувачів.**
6. **Яка з привілей надає користувачу повний доступ до БД?**
7. **Яким чином відбувається позбавлення прав користувачів?**

8. **Яким чином видаляють користувачів?**
9. **Які дії виконуються командою FLUSH PRIVILEGES?**

Лабораторна робота №5

Тема: Керування базами даних за допомогою SQL. TCL.

Мета: Навчитися контролювати транзакціями при роботі з БД.

Зміст роботи за варіантом індивідуального завдання:

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

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.

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 застосувати 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.

Необхоімо 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.

Оператор ROLLBACK TO SAVEPOINT відкочується транзакцію до іменованої точки збереження. Модифікації рядків, які виконувалися поточної транзакцією після цієї точки, скасовуються відкотом, однак InnoDB не знімає блокування рядків, які були встановлені в пам'яті після точки збереження. (Відзначимо, що для знову вставлених рядків

інформація про блокування спирається на ідентифікатор транзакції, збережений в рядку, блокування не зберігається в пам'яті окремо. У цьому випадку блокування рядка знімається при скасуванні.) Точки збереження, встановлені в більш пізні моменти, ніж іменована точка, видаляються.

Всі точки збереження транзакцій віддаляються, якщо виконується оператор COMMIT або ROLLBACK без вказівки імені точки збереження.
LOCK TABLES блокує таблиці для поточного потоку сервера. UNLOCK TABLES знімає будь-які блокування, утримувані поточним потоком. Всі таблиці, заблоковані в поточному потоці, неявно розблокуються, коли потік виконує інший оператор LOCK TABLES або коли закривається з'єднання з сервером.
LOCK TABLES не є оператором, безпечним щодо транзакцій, і неявно завершує транзакцію перед спробою блокувати таблиці.

Контрольні питання 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 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: Первинні та зовнішні ключі, обмеження, керування зв'язками між таблицями за допомогою SQL.

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.

Зміст роботи за варіантом індивідуального завдання:

1. Підготувати всі таблиці для БД, визначити необхідні обмеження (5 таблиць).
2. Визначити зовнішні ключі у підготовлених для зв’язку таблиць.
3. Графічне представлення
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).

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

Зовнішні ключі дозволяють встановити зв'язки між таблицями. Зовнішній ключ встановлюється для стовпців з залежної, підлеглої таблиці, і вказує на один із стовпців з головної таблиці. Як правило, зовнішній ключ вказує на первинний ключ з пов'язаної головної таблиці.

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

Було розроблено багато версій мови SQL, перш ніж він став настільки повноцінним і потужним. 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 1.1).

Таблиця 1.1 – 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 1.2 – 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

Як правило БД складаються із декількох таблиць, для яких ключі служать сполучними ланками. Так у таблиці 1.2 Orders (Закази) розміщено дані про закази, створені клієнтами.

Щоб реляційна база даних працювала належним чином, необхідно упевнитися в тому, що дані в таблиці введені правильно. Наприклад, якщо в таблиці Orders (таблиця 1.2) зберігається інформація про замовлення, а в Order Items - його детальний опис, ви повинні бути впевнені, що всі ідентифікатори замовлень, згадані в таблиці OrderItems, існують і в таблиці Orders. Аналогічно, кожен клієнт, згаданий в таблиці Orders, не повинен бути забутий і в таблиці Customers (таблиця 1.1).

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

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

Доведеться примусово ввести правила для виконання операцій UPDATE і DELETE. Виконання перевірок на клієнтській стороні - процес,

який забирає багато часу. Змусити СУБД виконувати ці перевірки - метод набагато ефективніший.

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

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

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

Первинні ключі

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 призначений на роль первинного ключа, але тільки якщо він задовольняє наступним умовам.

- Ніякі два рядки не можуть мати одне і те ж значення первинного ключа.
- Кожен рядок повинен мати якесь значення первинного ключа. (В таких шпальтах не повинно бути дозволено використання значень NULL.)
- Стовпець, що містить значення первинного ключа, не може бути модифікований або оновлений.
- Значення первинного ключа ні за яких обставин не можуть бути використані повторно. Якщо якийсь рядок видалено з таблиці, його первинний ключ не може бути призначений іншому рядку.

Зовнішні ключі

Зовнішній ключ - це стовпець однієї таблиці, значення якого збігаються зі значеннями стовпчика, що є первинним ключем іншої таблиці. Зовнішні ключі - дуже важлива частина механізму забезпечення посилальної цілісності даних. Щоб розібратися в тому, що собою представляють зовнішні ключі, розглянемо наступний приклад.

Таблиця Orders (таблиця 1.2) містить єдиний рядок для кожного замовлення, зафіксованого в базі даних. Інформація про клієнта зберігається в таблиці Customers (таблиця 1.1). 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 і цілеспрямоване вилучення, зовнішні ключі можуть надати допомогу 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 і 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 19.

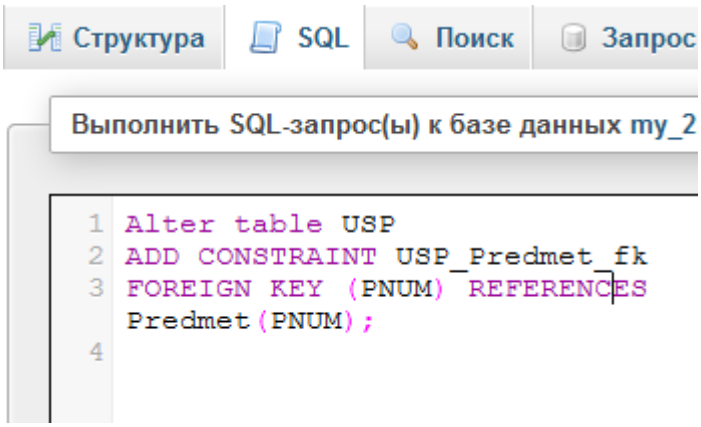


Рисунок 19

Щоб візуально побачити зв'язок між таблицями можна вибрать Дизайнер з вкладки меню Еще (рисунок 20)

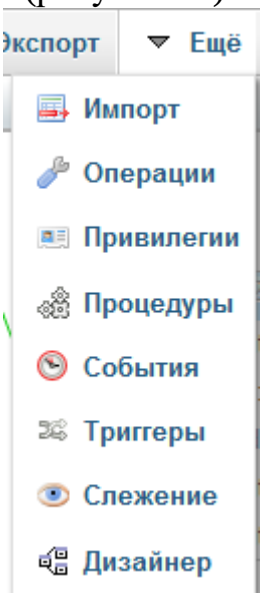


Рисунок 20

Результат буде, наприклад, такий, як на рисунку 21.

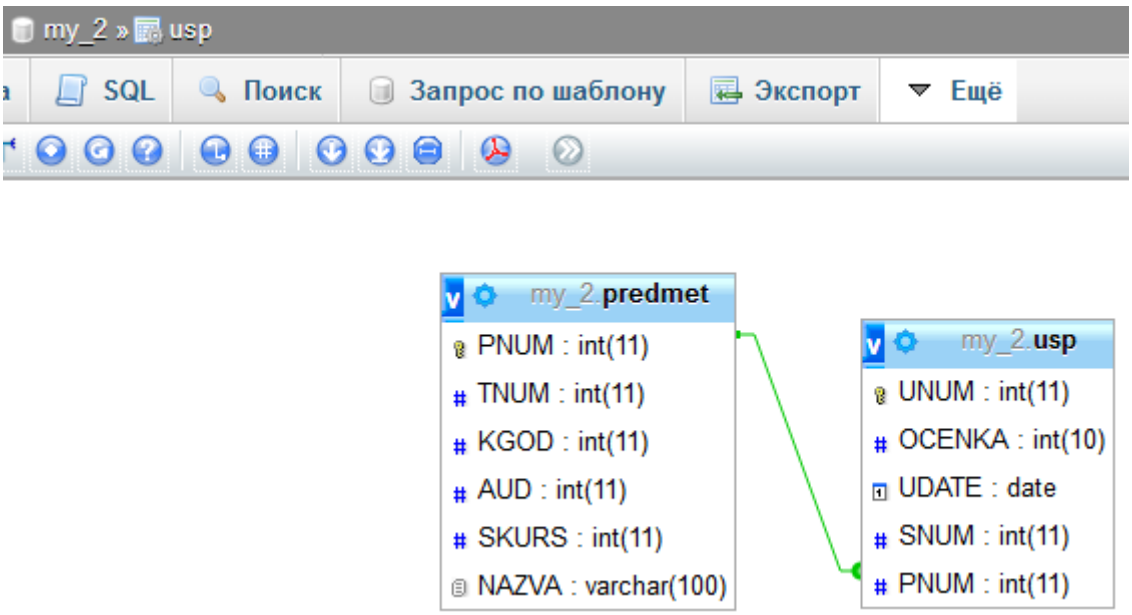


Рисунок 21

Загальний синтаксис установки зовнішнього ключа на рівні таблиці:
[CONSTRAINT имя_ограничения]
FOREIGN KEY (столбец1, столбец2, ... столбецN)
REFERENCES главная_таблица (столбец_главной_таблицы1,
столбец_главной_таблицы2, ... столбец_главной_таблицыN)
[ON DELETE действие]
[ON UPDATE действие]

Для створення обмеження зовнішнього ключа після FOREIGN KEY вказується стовпець таблиці, який буде представляти 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 і 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 і 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 А.

Контрольні питання до лабораторної роботи 6:

1. **Що означає поняття первинного ключа?**
2. **Дайте визначення зовнішнього ключа.**
3. **Які обмеження ви знаєте?**
4. **Яким чином можна створити первинний ключ?**
5. **Якими способами за допомогою SQL визначають зовнішні ключі?**

6. **За допомогою якої команди можна видалити зв’язок між таблицями?**
7. **Для чого використовують опцію CASCADE?**
8. **Що означає вираз ON UPDATE?**
9. **Що означає вираз ON DELETE?**

Лабораторна робота №7

Тема: Типи з’єднань в MySQL.

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

Зміст роботи за варіантом індивідуального завдання:

Виконати різні запити на пошук 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.

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*).

Внутрішнє з'єднання з'єднує записи двох таблиць (A та B) на основі заданого предикату з'єднання. При цьому обчислюється декартів добуток усіх записів таблиць. Таким чином, усі записи таблиці A буде з'єднано з кожним із записів таблиці B, після чого в результатній таблиці залишаться лише ті записи, які задовільняють предикат з'єднання.

Результат лівого зовнішнього з'єднання для таблиць А і Б містить всі кортежі з лівої таблиці (А), навіть якщо умова об'єднання не містить збігів з кортежами правої таблиці (Б). Це означає те, що, якщо умова порівняння не знайде записів у таблиці Б, то з'єднання в результаті все ж поверне рядки,

але значення з колонок таблиці Б будуть порожніми. Іншими словами, ліве зовнішнє з'єднання повертає всі значення з лівої таблиці і додає значення колонок з правої таблиці або NULL, якщо немає збігу за предикатом з'єднання.

Результат правого зовнішнього з'єднання для таблиць А і Б містить всі кортежі з правої таблиці (Б), навіть якщо умова з'єднання не містить збігів з кортежами лівої таблиці (А). Це означає те, що, якщо умова порівняння не знайде записів в таблиці А, то з'єднання в результаті все ж поверне рядки, але значення з колонок таблиці А будуть нульовими. Іншими словами, праве зовнішнє з'єднання повертає всі значення з правої таблиці і додає значення колонок з лівої таблиці або NULL, якщо немає збігу за предикатом з'єднання.

Повне зовнішнє з'єднання сполучає результати лівого та правого зовнішніх з'єднань. Результатна таблиця містить усі записи з обох таблиць, позначаючи NULL-значеннями відсутність збігів з кожного боку.

Оператор декартовогоз'єднання `CROSS JOIN` з'єднує дві таблиці. Порядок таблиць для оператора неважливий, оскільки оператор є симетричним. Заголовок таблиці-результату є об'єднанням (конкатенацією) заголовків таблиць, що з'єднуються. Тіло результату формується таким чином: кожен рядок однієї таблиці з'єднується з кожним рядком іншої таблиці, даючи тим самим в результаті всі можливі поєднання рядків двох таблиць.

Приклад виконання роботи:

Використовуючи раніше створенні таблиці виконаємо відповідні запити:

orders

←T→	order_id	customer_id	amount	date
	1	3	30.00	2020-01-01
	2	1	12.00	2020-01-07
	3	3	45.00	2020-01-16
	4	1	78.00	2020-01-02
	6	2	25.25	2020-01-07

Рисунок 22

customers

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

Рисунок 23

Пошук заказів з двох таблиць які зробив «Юрій Михайлович»

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

Результат

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

Рисунок 24

Дозаповнимо нашу БД ще 3 таблицями для виконання необхідних запитів

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

books

←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 25

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

Order_items

←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 26

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

book_reviews

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

Рисунок 27

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)
);

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

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

Виконаємо запит, хто робив хоча б один заказ по курсу JAVA:

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

SELECTcustomers.name

2) Об’єднати усі 4 таблиці тому що заказані книги у нас в таблиці order а ім’я замовника пов’язане з таблицею custmers.

Зроблені замовлення в таблиці Order_itemsa опис книг для запиту пошуку по опису відповідно в таблиці books.

Тому виконаємо запит INNERJOIN замінивши записом через кому

FROM customers, orders, order_items, books

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 28:

Параметри

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

Рисунок 28

Як бачимо в усіх книгах в назві присутня назва «Java».

Далі виконаємо розповсюджений в MySQL тип з'єднання «LEFTJOIN».

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 заказу, або книги, які ніхто не заказував. Саме простіше тут виконати лівостороннє з'єднання, при якому виконується пошук рядків за вказаними умовами з'єднання двох таблиць. Якщо у вказаній справа таблиці немає підходящого кортежу, то до результату добавляється рядок з нульовими значеннями в відповідних стовпчиках.

Приклад:

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

Далі виконаємо виведення тільки тих клієнтів, які нічого не заказали, це перевірка на значення NULL в полі первинного ключа правої таблиці (order_id), так як 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 використовуєтьсяконструкція AS(alias). Їх можна створити в самому початку запиту, а потім використовувати по необхідності. Часто псевдоніми використовують в якості коротких імен.

Розглянемо запит:

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

Такий запит більш короткий крім цього псевдоніми можна присвоювати стовпчикам. Псевдоніми таблиць необхідні коли потрібно з'єднати таблицю з самою собою (наприклад для пошуку в цій же таблиці строчок які мають однакові значення). Так якщо нам буде необхідно знайти клієнтів які проживають в одному городі, можна одній і тій же таблиці присвоїти різні псевдоніми і 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 взнати, скільки строк відноситься до певного набору або середнє значення якого-небудь стовпця таблиці, скажімо середня сума заказу в грошовому еквіваленті. Для цього в MySQL використовуються функції агрегування. Дані функції можна застосовувати як до таблиці в цілому так і до груп даних всередині таблиці.
Найбільш часто які використовуються на практиці наведено в таблиці 2.

Таблиця 2 – Функції агрегування в MySQL

AVG(стовпчик)	Середня величина значень у вказаному стовпці.
COUNT(елементи)	При вказуванні стовпця видається кількість не нульових значень. Якщо пере назвою стовпця помістити оператор DISTINCT, видається тільки кількість різних значень в стовпцю. Якщо вказати COUNT(*) то підрахунок строчок буде проведено незалежно від нульових значень.

MIN(стовпець)	Мінімальне значення у вказаному стовпцю.
MAX(стовпець)	Максимальне значення у вказаному стовпцю.
STD(стовпець)	Стандартне відхилення у вказаному стовпці.
STDDEV(стовпець)	Аналогічно попередньому.
SUM(стовпець)	Сума значень у вказаному стовпці.

Розглянемо декілька прикладів:

Приклад1:

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

Результат на рисунку 29:

AVG(amount)
36.750000

Рисунок 29

Для більш детально інформації виконаємо наступний запит:

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

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

Результат на рисунку 30:

customer_id	AVG(amount)
1	45.000000
2	25.250000
3	37.500000

customer_id	AVG(amount)
4	30.250000

Рисунок 30

Контрольні питання до лабораторної роботи 7:

1. **Які типи з’єднань існують в MySQL?**
2. **Що означає операція Join у реляційній базі даних?**
3. **Яке з’єднання називають внутрішнім?**
4. **Яке з’єднання називають зовнішнім?**
5. **Яке з’єднання називають перехресним?**
6. **Що таке псевдоніми (alias)?**
7. **Агрегування даних. Які функції агрегування розглядаються?**
8. **На які з'єднання поділяється зовнішнє з'єднання?**

Додаток А

Приклад створення бази даних

БД повинна містити дві таблиці відповідно до рисунка 1.1. За допомогою операторів SQL вносяться відповідні дані в кожну таблицю по 5 кортежів в кожну. Виконується зв'язок таблиць. Створюються відповідні запити.

Customers(Клієнти)

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

Orders(Закази)

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

Рисунок 1.1 – В кожному замовленні з таблиці Orders є вказівник на клієнта із таблиці Customers.

Як правило, БД складаються із декількох таблиць, для яких ключі служать сполучними ланками. Так, на рисунку 1.1 показана база даних, в яку додається ще одна таблиця. В ній розміщено дані про закази створені клієнтами.

Виконання:

Завантажити OpenServer. Перейти в розділ Додатково. PhpMyAdmin.

Після завантаження перейти в розділ виконання SQL запитів та створити базу даних «books».

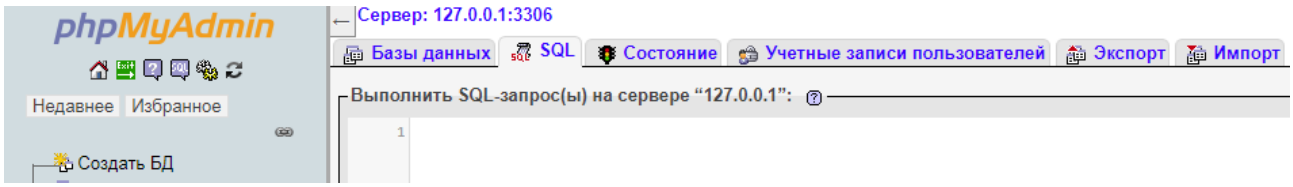


Рисунок 1.2 – PhpMyAdmin

Далі виконаємо відповідні SQL-запити.

Для видалення можливих існуючих таблиць виконаємо запити:

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

Та виконаємо запит для створення відповідної бази даних:

```
CREATE DATABASE books
```

Далі переходимо в відповідну базу даних написавши запит:

```
USE books;
```

Після створення бази даних можна приступити до створення відповідних таблиць «Customers»та «Orders».

Виконаємо запити для створення таблиці «Customers»з відповідними полями як показано на рисунку 1.1:

```
CREATE TABLE customers(
customer_id INT unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(40) NOT NULL,
address CHAR(50) NOT NULL,
city CHAR(30) NOT NULL
);
```

Після виконання запиту система повинна вернути поле запиту, як показано на рисунку 1.3.

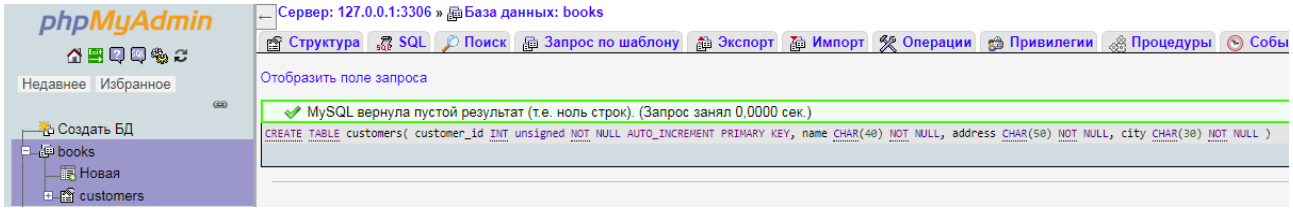


Рисунок 1.3 – результат виконання запиту

Також наступним запитом створюємо таблицю «Orders»:

```
CREATE TABLE orders(
order_id INT unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
customer_id INT unsigned NOT NULL,
amountfloat(6,2),
datedate NOT NULL
);
```

Після створення двох таблиць (між ними буде зв'язок) одна з них у нас буде «дочірня» (зберігає дані із батьківської таблиці) а перша буде

називатись «батьківська». Це будуть відношення між даними створенні за допомогою зовнішніх ключів. Так ми в нашій базі даних створюємо відношення між рядком в таблиці «Orders» та рядком в таблиці «Customers».

Зв'язок між таблицями ми виконаємо по ідентифікатору клієнта (Customer_id) за допомогою зовнішніх ключів (Foreigh key).

Синтаксис представлено нижче відповідно до документації MySQL:

[CONSTRAINT[*symbol*]]FOREIGNKEY
[*index_name*](*col_name*,...)
REFERENCES*tbl_name*(*col_name*,...)
[ONDELETE*reference_option*]
[ONUPDATE*reference_option*]

reference_option:
RESTRICT|CASCADE|SETNULL|NOACTION

Виконаємо запит на доповнення таблиці:

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

Після виконання відповідного запиту перейшовши в розділ Структура – Зв’язки, ми можемо побачити встановлення зв’язків між таблицями (рисунок 1.4).

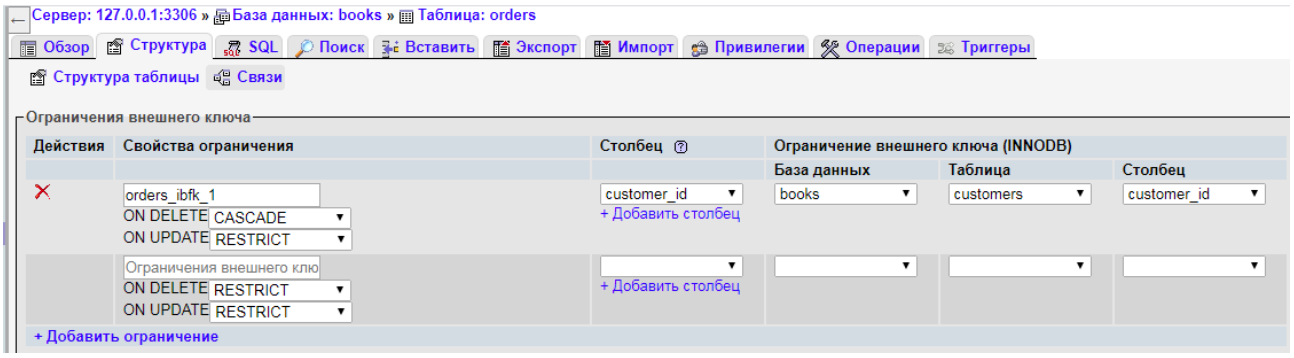


Рисунок 1.4 – зв’язки в таблицях

Тепер ми отримали дві пов’язані таблиці, тепер їх можна заповнювати даними.

Відповідно при внесенні даних в таблиці, спочатку повинна заповнюватися батьківська таблиця (клієнти), з причини, що дані в полі Customers_idтаблиці Orders будуть залежати від даних батьківської таблиці.Так в нашій таблиці Order в полі запису Customer_Id ми не повинні мати змогу внести даних не існуючого ID клієнта.

Наступним етапом є заповнення таблиць нашої бази відповідними даними використовуючи оператор INSERT, що має наступний вигляд:

```
INSERT [INTO] таблиця [(стовпець1, стовпець2,….)]
VALUES (значення1, значення2,значення3,…).
```

Заповнюємо таблицю клієнти за прикладом, що наведено нижче:
INSERT INTO `customers` (`customer_id`, `name`, `address`, `city`) VALUES (NULL, 'МихайлоЗгама', 'вул.Квіткова.6', 'м.Харків');
INSERT INTO `customers` (`customer_id`, `name`, `address`, `city`) VALUES (NULL, 'ВалентинШевченко', 'вул.Вишнева 7', 'м.Кропивницький');

Після заповнення таблиці «клієнти» приступаємо заповнювати таблицю «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');

Тепер спробуємо вставити в таблицю «закази» дані з неіснуючим ідентифікатором клієнта (order_id = 5)
INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`) VALUES (NULL, '5', '45', '2020-01-16').
Звичайно, система видасть помилку, тому що клієнта з таким ідентифікатором, поки що, не існує, і відповідний запит ми зможемо виконати тільки при появі клієнта з Custmer_id = 5.

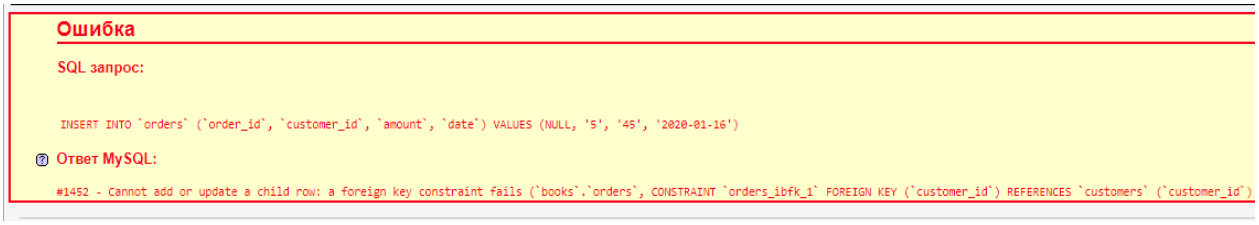


Рисунок 1.5 – Помилковий SQL-запит для зв’язаних таблиць

Такий підхід дозволяє зберегти цілісність даних в таблицях.
Відповідно це добре видно в графічному інтерфейсі, коли ми робимо вставку даних система автоматично запропонує нам можливі варіанти для заповнення таблиці «Закази» поле «customer_id» рисунок 1.6. Як бачимо варіантів крім тих що є в таблиці «Клієнти» система нам не пропонує.

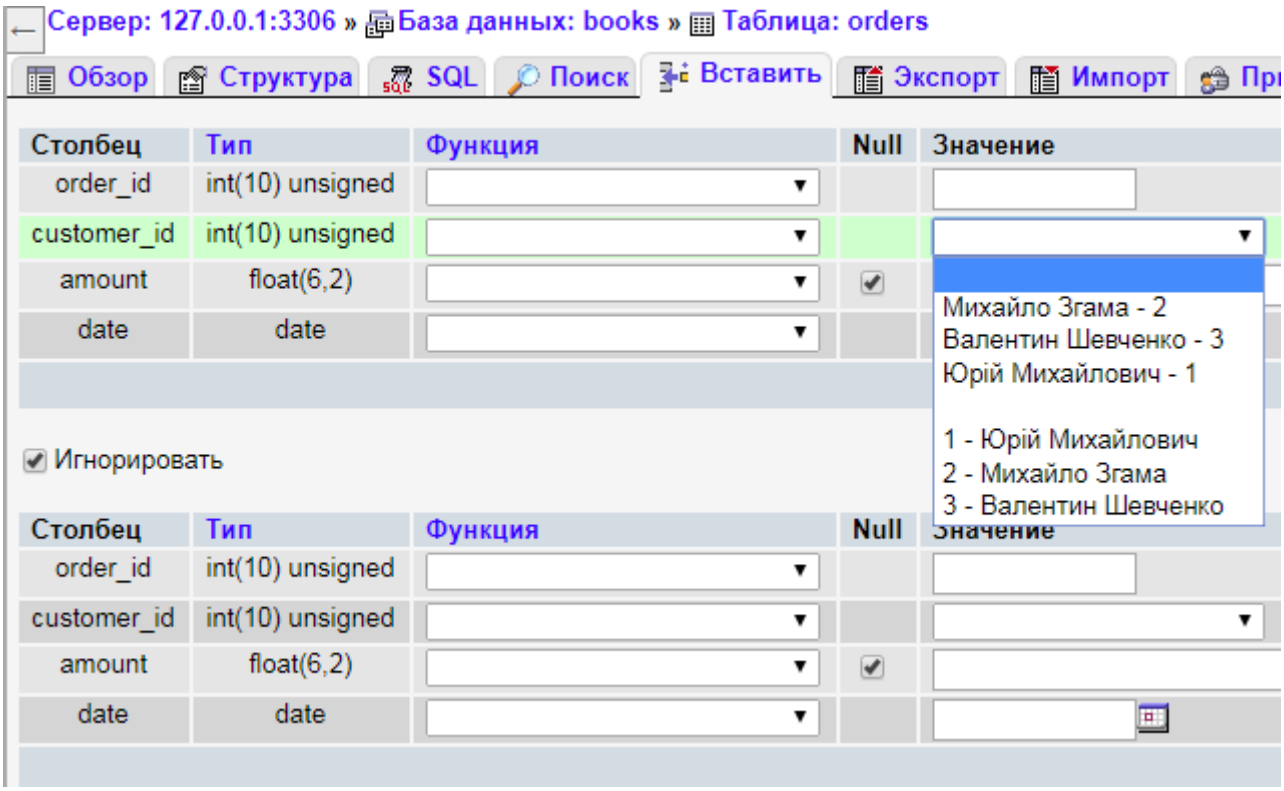


Рисунок 1.6 – Введення даних зв’язаних таблиць за допомогою графічного інтерфейсу

Таким чином, було створено базу даних з виконанням всіх необхідних умов. Бажаємо успіхів!

Використана література

1. Васвани В. MySQL: использование и администрирование. – СПб.: Питер, 2011. – 368 с.: ил.
2. А.В. Маркин Построение запросов и програмирование на SQL: учеб. пособие, - Рязань: РГРТУ, 2008.-312с.
3. Ицик Бен-Ган Microsoft SQL Server 2008. Основы T-SQL:Пер. с англ.-СПб.: БХВ-Петербург, 2009.-432 с.:ил
4. Бьюли А. Изучаем SQL. – Пер. с англ.-СПб:Символ-Плюс, 2007.-312с., ил.
5. С.В. Глушаков Д.В. Ломотько Базы данных Учебный курс Харьков «Фолио» Москва «АСТ» 2001.
6. Конноллн, Томас, Бегг, Карелии. К64 Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильяме", 2003. — 1440 с. : ил.
7. А.Ю. Берко О.Н. Верес Організація баз даних Практичний курс. Навчальний посібник. Львів, 2003.
8. Л. М. Дибкова Інформатика і комп’ютерна техніка Навчальний посібник 3-тє видання, доповнене Київ «Академвидав» 2011.
9. Гайна Г.А. Г12 Основи проектування баз даних: Навчальний посібник. − К.: КНУБА, 2005. – 204 с. ISBN 966-627-117-6
10. Справочное руководство по MySQL [Електронний ресурс]. – Режим доступу: http://www.mysql.ru/docs/man/SET_TRANSACTION
11. Денвер - локальный сервер. [Електронний ресурс]. – Режим доступу: http://www.denwer.ru
12. SQL Fiddle [Електронний ресурс]. – Режим доступу: http://sqlfiddle.com/about.html
13. Open Server [Електронний ресурс]. – Режим доступу: https://ospanel.io/docs/
14. Установка web сервера (OpenServer) 19 ноября 2015 PHP Start [Електронний ресурс]. – Режим доступу: https://php-start.com/blog/open-server-intallation
15. Форта Бен Освой самостоятельно SQL. 10 минут на урок, 3-е издание. : Пер. с англ. — М. : Издательский дом “Вильямс”, 2006. — 288 с. : ил.
16. FOREIGN KEY Constraints [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html

17. Руководство по MySQL [Електронний ресурс]. – Режим доступу: https://metanit.com/sql/mysql/2.5.php