МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«ОРГАНІЗАЦІЯ БАЗ ДАНИХ»

ДЛЯ СТУДЕНТІВ ДЕННОЇ ФОРМИ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ
123 – «КОМП’ЮТЕРНА ІНЖЕНЕРІЯ»
ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»
ЧАСТИНА І

КРЕМЕНЧУК 2021

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Організація баз даних» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія» освітнього ступеня «Бакалавр». Частина І

Укладачі : к. т. н. П. П. Костенко,
асист. Н. Л. Сохін
Рецензент д. т. н., проф. М. І. Гученко

Кафедра комп'ютерних та інформаційних систем

Затверджено методичною радою Кременчуцького національного університету імені Михайла Остроградського
Протокол № _____ від _____
Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

ВСТУП

Методичні вказівки укладені на підставі робочої програми навчального курсу «Організація баз даних» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія» освітнього ступеня «Бакалавр».

Виконання лабораторних робіт дозволить студентам поглибити теоретичні знання, отримані під час лекційних занять, підвищити рівень їх засвоєння та набути практичних навичок у галузі створення баз даних (БД).

У результаті виконання лабораторних робіт студент повинен

знати:

– призначення та основні принципи організації БД;

– основні моделі БД і способи їх реалізації;

– мову запитів SQL;

– принципи оброблення запитів.

уміти:

– створювати та наповнювати бази даних;

– модифікувати структуру таблиць бази даних;

– забезпечувати контроль і відновлення цілісності даних;

– використовувати мову SQL для створення запитів і роботи з даними.

1 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1

Тема. Знайомство з PHPMyAdmin

Мета: отримання практичних навичок роботи з додатком для адміністрування СУБД MySQL PHPMyAdmin.

Короткі теоретичні відомості

PHPMyAdmin – це веб-додаток з відкритим кодом, написаний мовою PHP, що надає графічний веб-інтерфейс для роботи базою даних (БД) MySQL або MariaDB. Тобто PHPMyAdmin дозволяє через браузер адмініструвати сервер MySQL, запускати SQL-запити, переглядати та редагувати вміст таблиць БД. Його дистрибутив можна завантажити за адресою: *https://www.phpmyadmin.net/downloads/*. Для роботи системи потрібний веб-сервер з підтримкою php (наприклад Apache).

Якщо працювати з розгорнутим на локальному ПК веб-сервером у складі пакету DENWER, то для входу в PHPMyAdmin необхідно в адресному рядку браузера набрати посилання *http://localhost/Tools/phpMyAdmin/index.php*. В іншому ж випадку частина «*http://localhost/Tools/phpMyAdmin/*» замінюється на шлях до папки на веб-сервері, де розгорнуто PHPMyAdmin.

У вікні входу, що відкриється (рис. 1), необхідно ввести логін та пароль користувача. При використанні пакета DENWER за замовчуванням користувач БД має ім’я *root*, а пароль відсутній. У іншому випадку логін і пароль користувача необхідно отримати у адміністратора БД.

Після введення правильного логіна і пароля відкриється головна сторінка додатка (рис. 2). На ній відображені такі основні розділи:

1. Список існуючих БД. Клацнувши на будь-яку з них, можна перейти на сторінку для роботи з вибраною БД.

2. Вкладка *Базы данных* (рис. 3) дозволяє створювати нові БД та адмініструвати існуючі.

Рисунок 1 – Форма входу в PHPMyAdmin

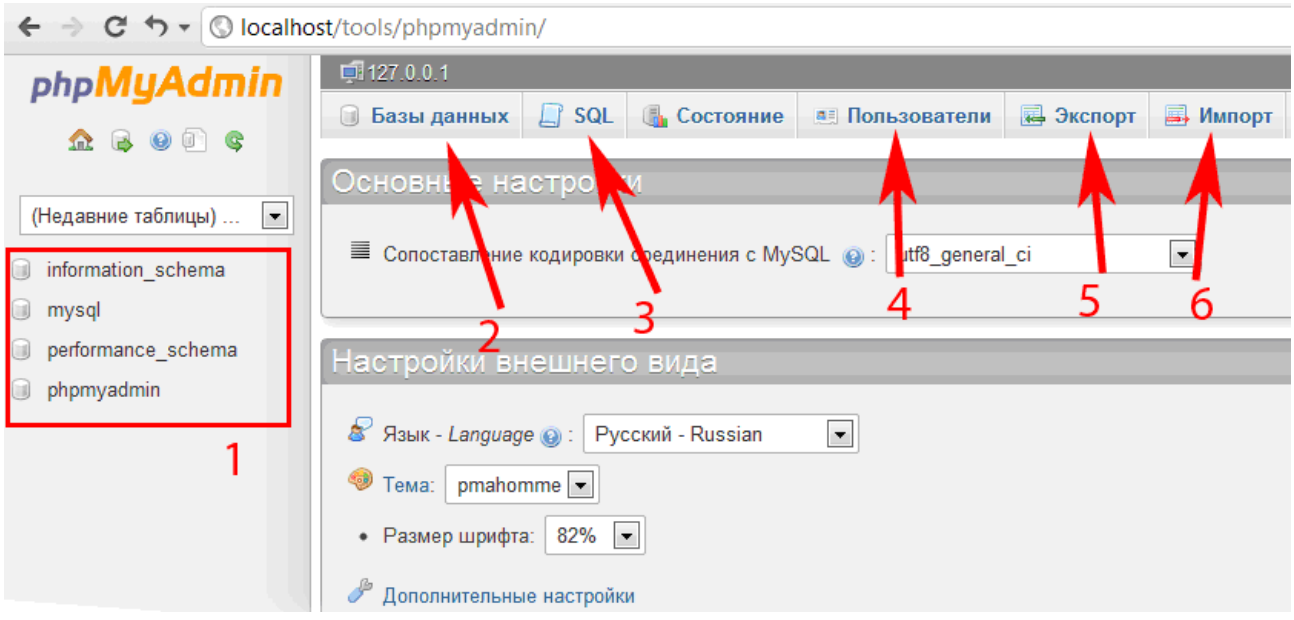

Рисунок 2 – Головна сторінка PHPMyAdmin

3. Вкладка *SQL* дозволяє виконувати будь-які SQL-запити.

4. Вкладка *Пользователи* дозволяє додавати, видаляти і редагувати користувачів MySQL, а також редагувати їх права.

5. Вкладка *Экспорт* дозволяє експортувати вибрану БД в окремий файл для перенесення на інший сервер.

6. Вкладка *Импорт* дозволяє імпортувати дані в існуючу БД.

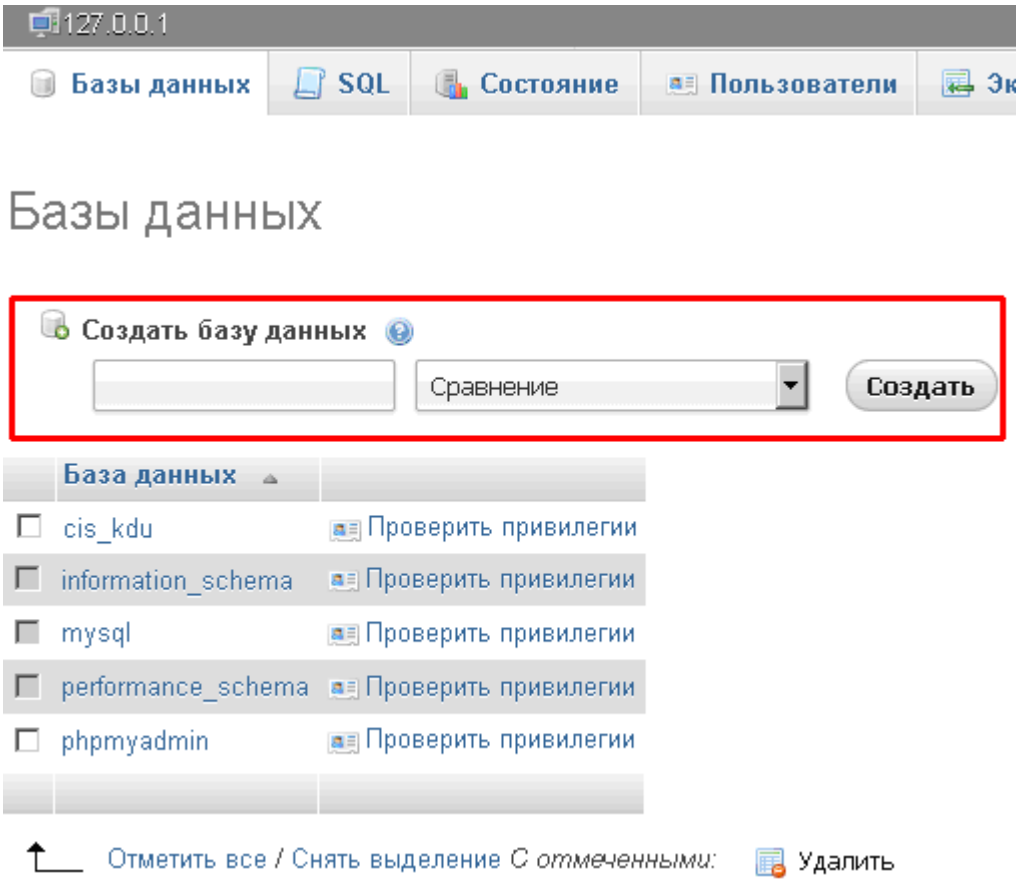

Рисунок 3 – Вікно адміністрування БД у PHPMyAdmin

Щоб створити нову БД, слід задати її ім’я та у полі *Сравнение* вибрати кодування, після чого натиснути *Создать*. Щоб додати таблиці до БД, необхідно спочатку зробити її активною, натиснути на ім’я БД у полі зліва (після чого у цьому полі будуть відображатися існуючі у вибраній БД таблиці). Тепер можна створювати нові таблиці (рис. 4), вказавши їх ім’я та кількість стовпчиків.

Вигляд форми для задання імен та властивостей стовпчиків таблиці наведено на рис. 5.

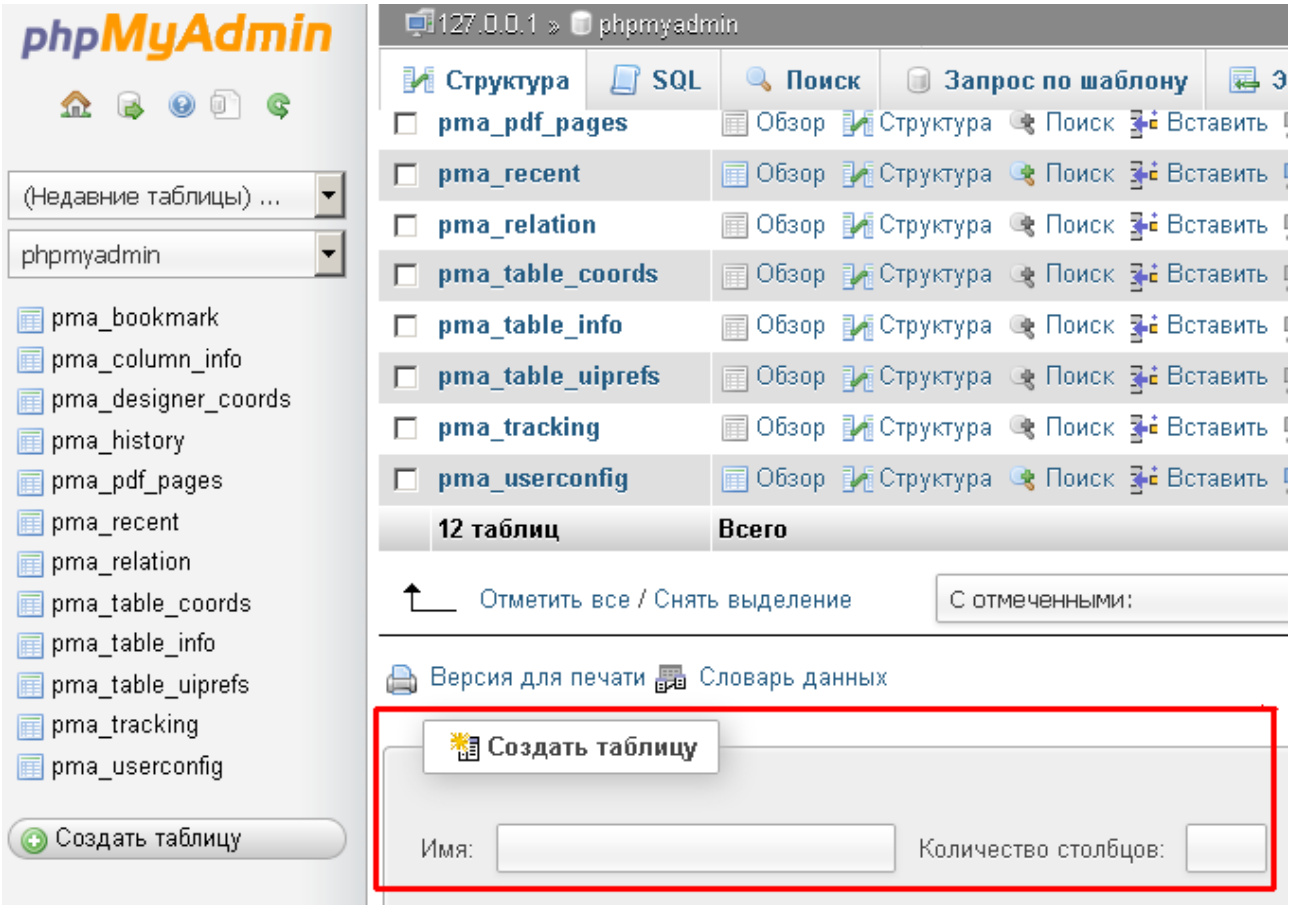

Рисунок 4 – Створення таблиці БД в PHPMyAdmin

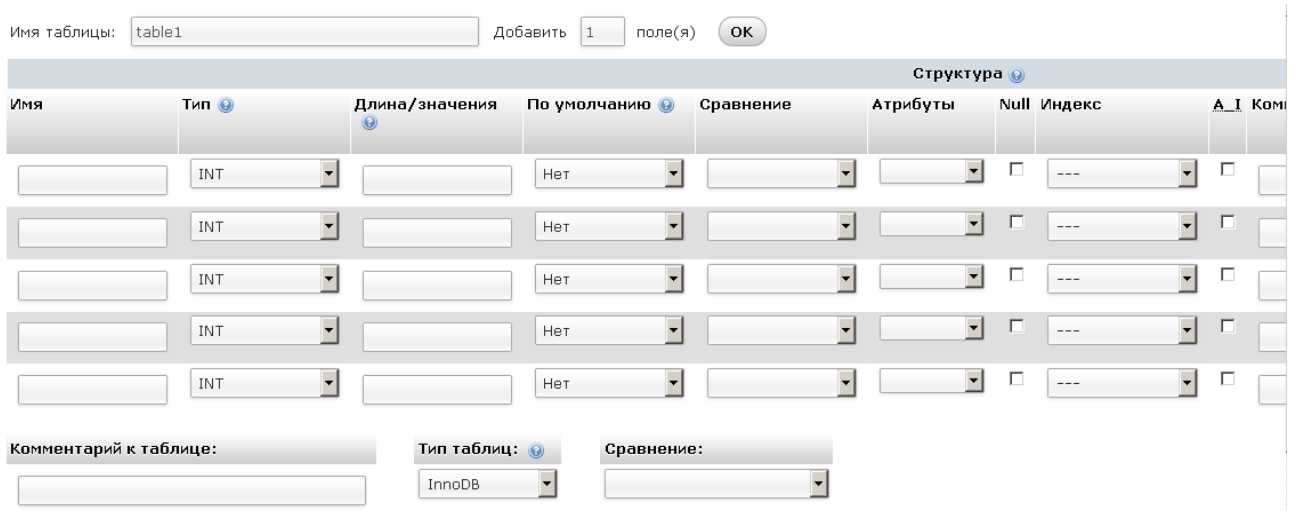

Рисунок 5 – Форма створення таблиці в PHPMyAdmin

Щоб внести зміни до структури таблиці (змінити кількість стовпчиків або їх параметри), необхідно перейти на вкладку *Структура* (рис. 6). У цьому вікні виведено список таблиць поточної БД. Щоб перейти до редагуваня структури

таблиці (рис. 7), слід натиснути на ім’я потрібної таблиці.

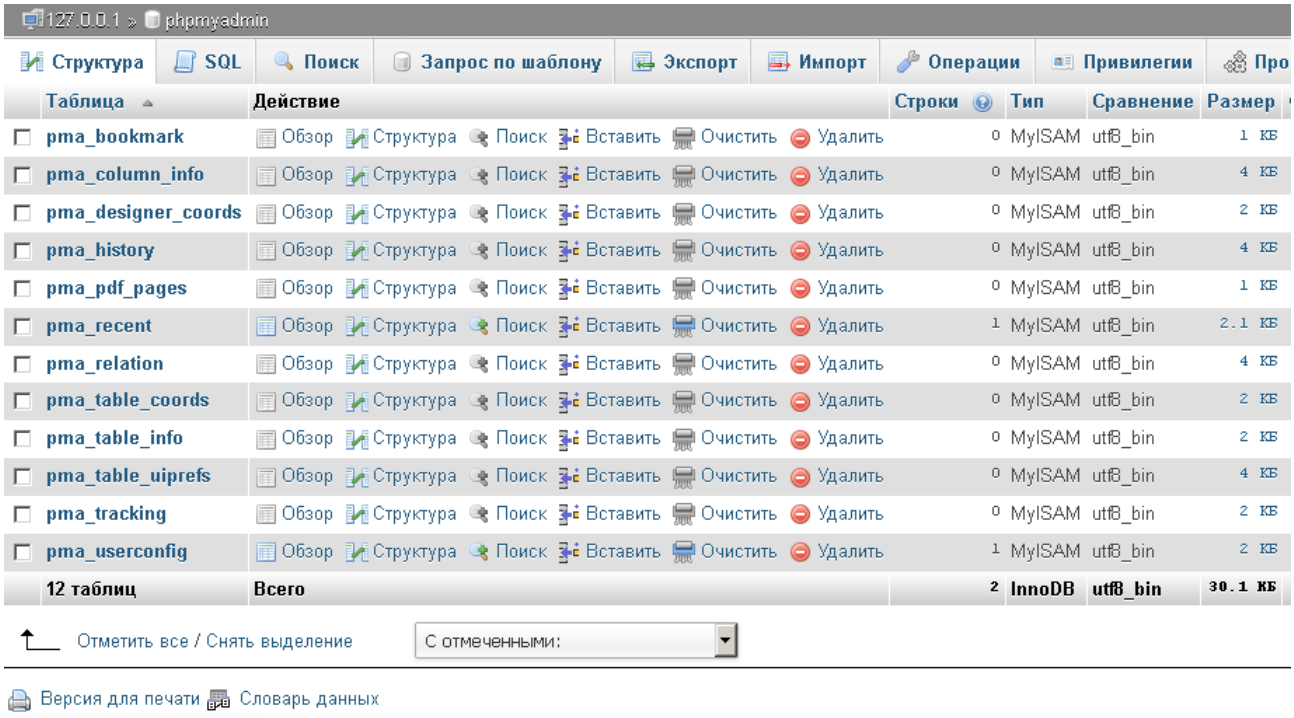

Рисунок 6 – Вікно перегляду структури БД у PHPMyAdmin

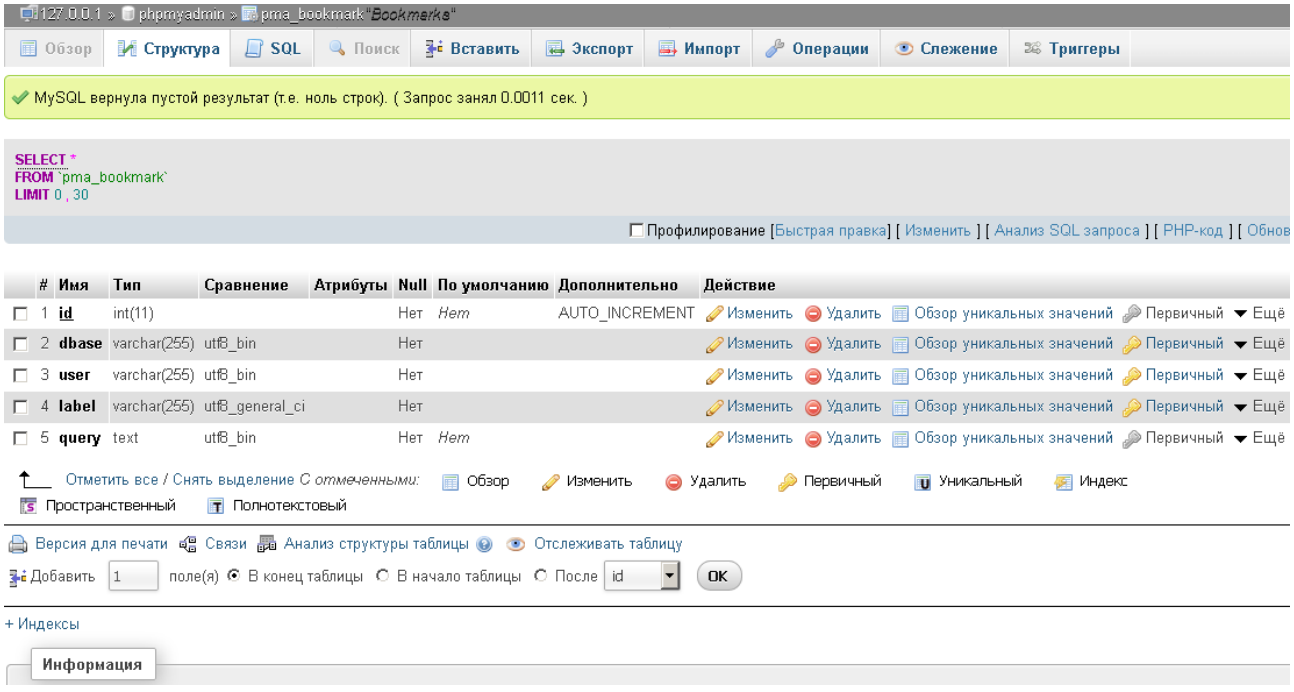

Рисунок 7 – Вікно перегляду структури таблиці в PHPMyAdmin

Щоб переглянути вміст поточної таблиці БД, необхідно перейти на вкладку *Обзор* (рис. 8).

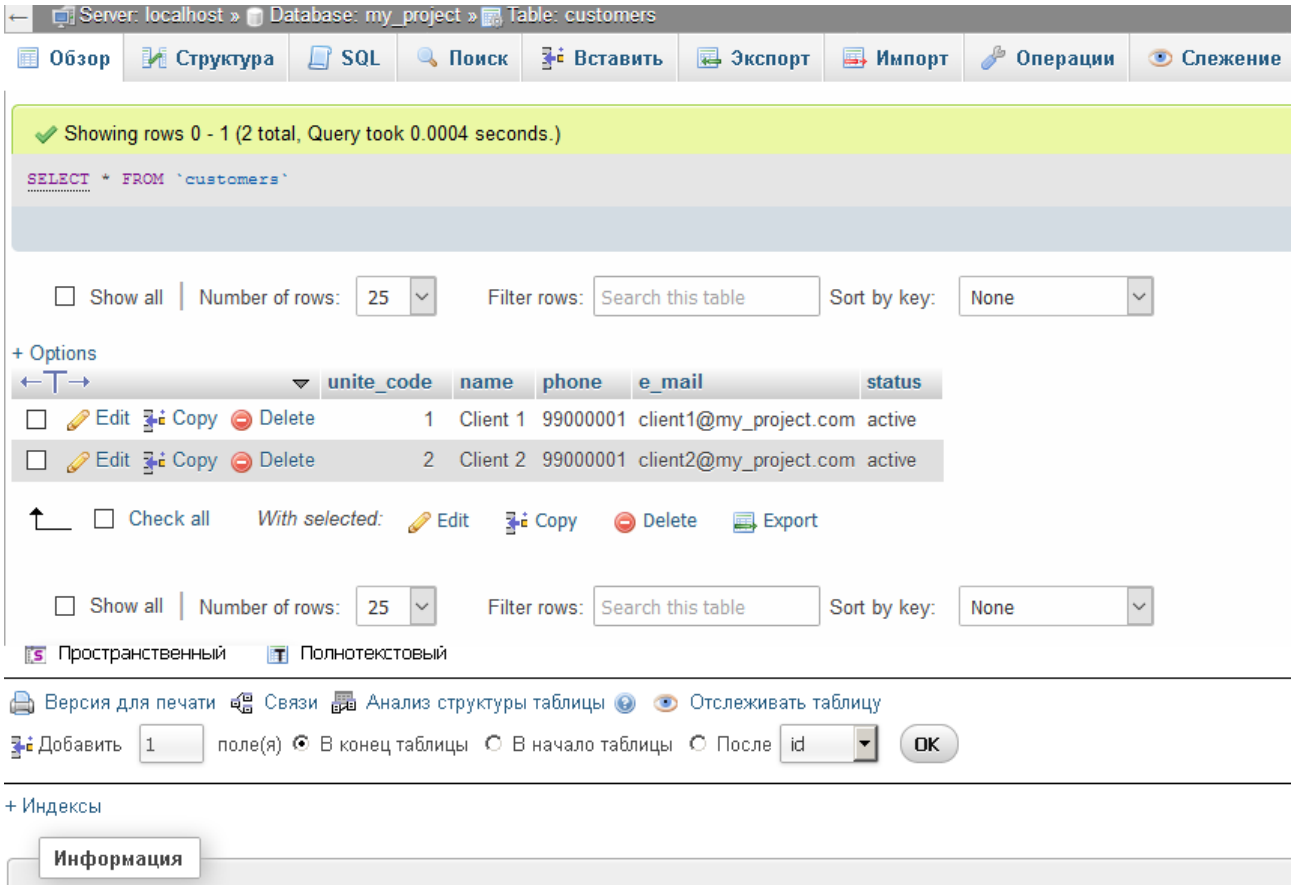

Рисунок 8 – Вікно перегляду вмісту таблиці БД у PHPMyAdmin

Вміст рядків таблиці можна змінювати, копіювати чи видаляти, натиснувши на відповідне посилання зліва. Щоб додати дані в таблицю (рис. 9), необхідно перейти на вкладку *Вставить*.

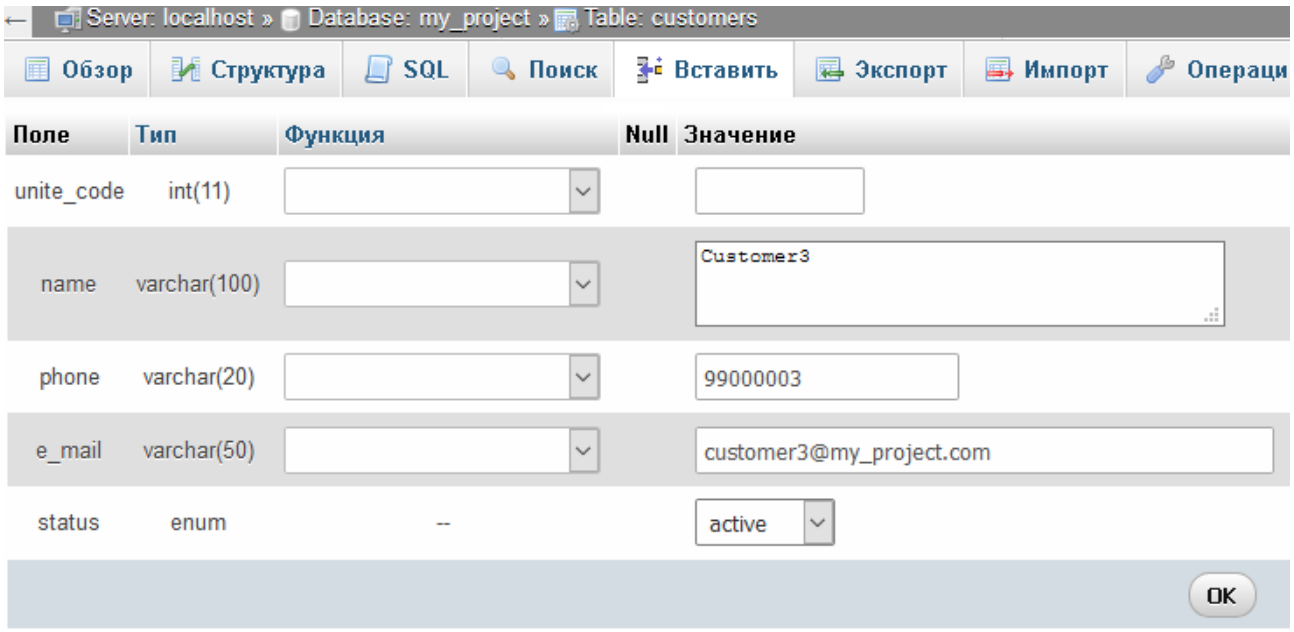

Рисунок 9 – Форма додавання даних у таблицю в PHPMyAdmin

Окрім маніпулювання з таблицями БД, PHPMyAdmin дозволяє створювати SQL-запити. Щоб створити запит, необхідно перейти на вкладку *SQL* (рис. 10).

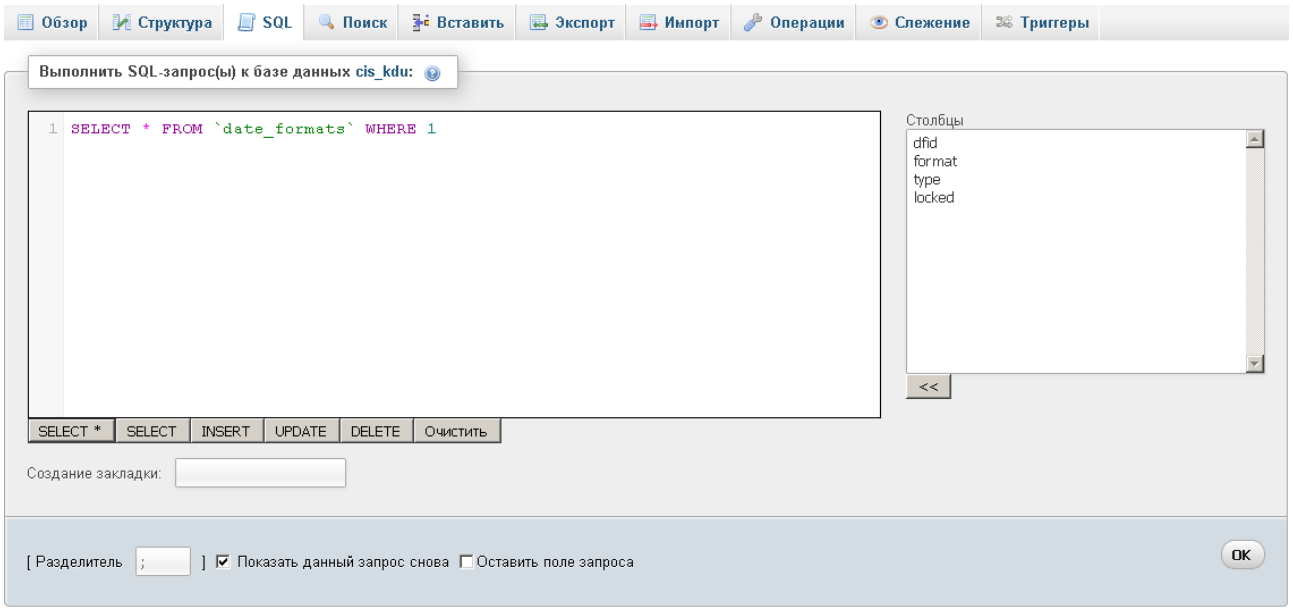

Рисунок 10 – Форма для створення запиту в PHPMyAdmin

Якщо базу даних необхідно перенести на інший сервер, то на вихідному сервері виконують експорт БД, а на новому – її імпорт. Щоб виконати експорт поточної БД, необхідно перейти на вкладку *Экспорт* (рис. 11). Попередньо слід вибрати базу даних, яку необхідно експортувати.

У цьому вікні вибирають спосіб експорту (*Быстрый* – якщо немає необхідності вказувати спеціальні налаштування експорту) та місце, куди буде записано експортний файл (пункт *Вывод*).

Щоб імпортувати таблиці в БД, необхідно перейти на вкладку *Импорт* (рис. 12). Попередньо слід вибрати базу даних, в яку необхідно виконати імпорт.

Тут вказують місцезнаходження файла для імпорту, його кодування та формат імпорту (окрім *SQL* часто використовують формати *XML* та *CSV*).

Вибравши всі необхідні параметри, слід натиснути кнопку *ОК* для початку процесу імпорта.

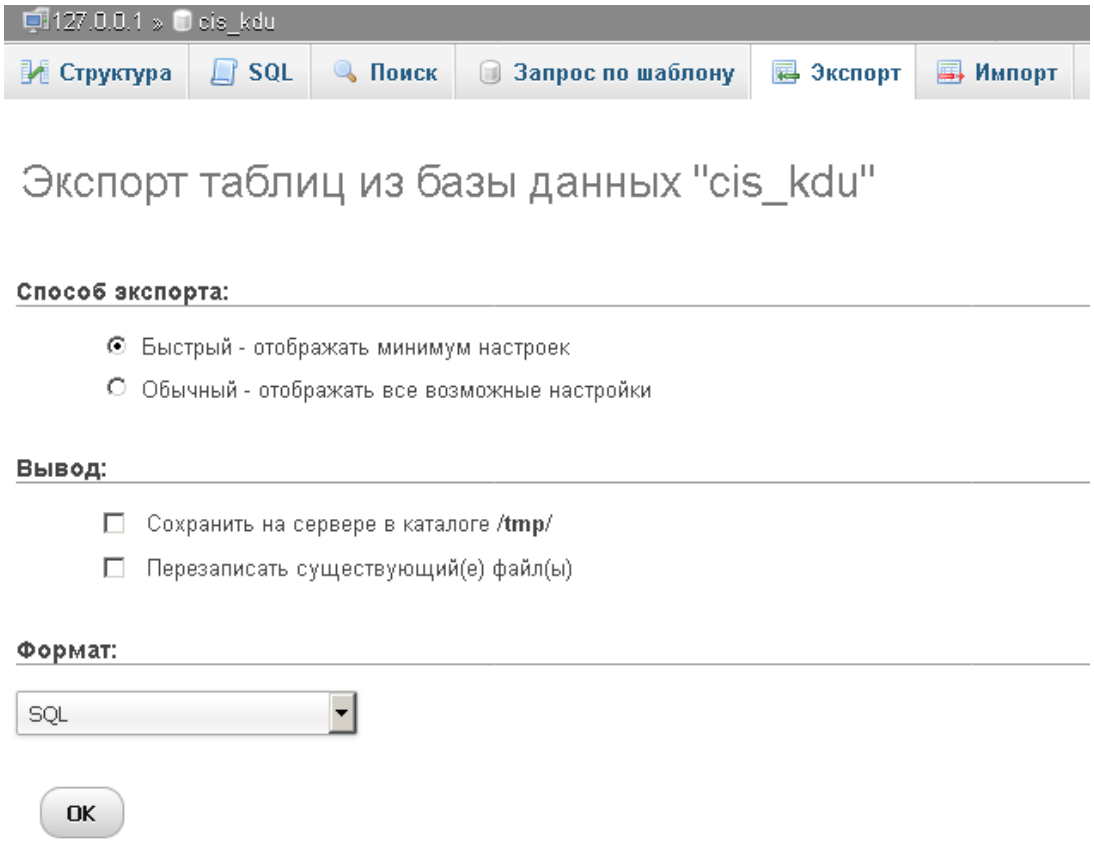

Рисунок 11 – Форма для експорту БД у PHPMyAdmin

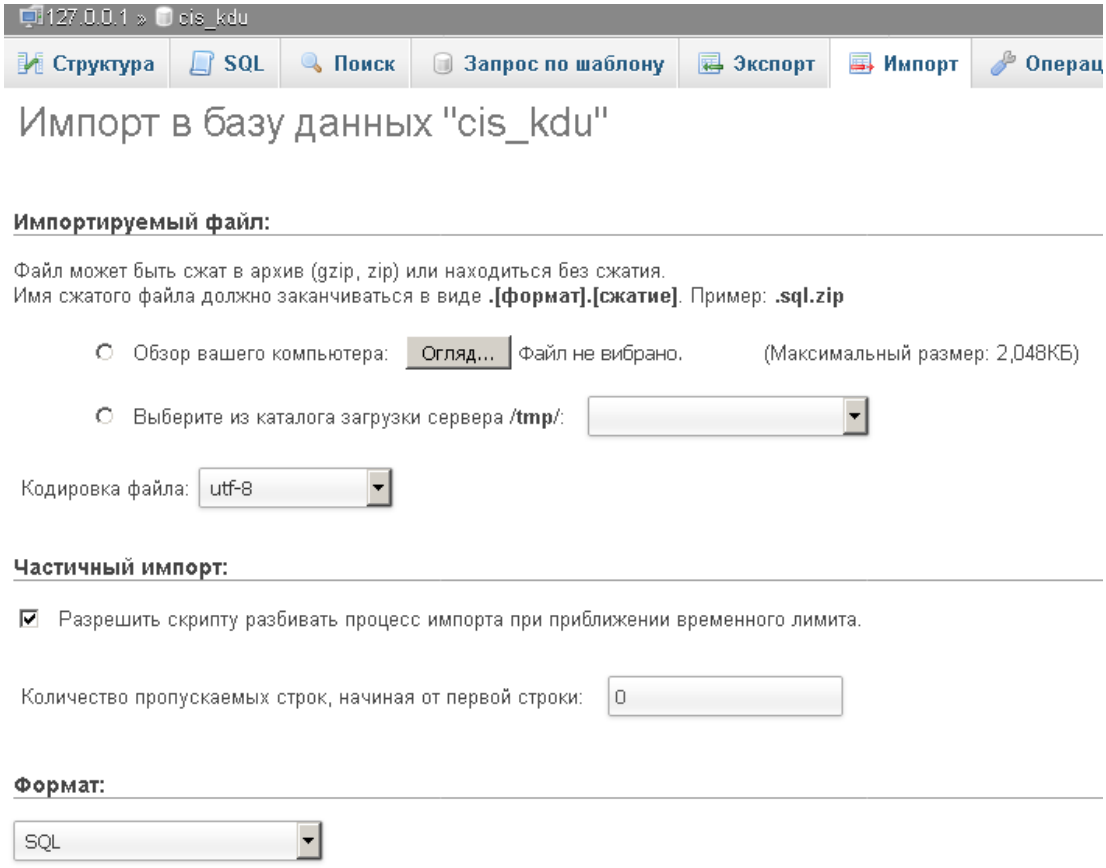

Рисунок 12 – Форма для імпорту БД у PHPMyAdmin

Порядок виконання роботи

1. Встановити пакет DENWER.

2. Запустити пакет (ярлик Start DENWER).

3. Запустити бразер та в адресному рядку набрати *localhost*.

4. На стартовій сторінці DENWER перейти за посиланням http://localhost/Tools/phpMyAdmin («Проверка MySQL и phpMyAdmin»).

5. У web-інтерфейсі PHPMyAdmin перейти на вкладку «Базы данных» та створити нову базу даних (БД) з ім’ям MyDB_<N>, де <N> – порядковий номер студента в журналі групи.

6. Зробити створену БД поточною, натиснувши на посилання з її ім’ям в меню зліва, та створити в БД 3 таблиці: t1 (3 поля), t2 (3 поля) та t3 (2 поля). Правила задання імен полів аналогічні правилам задання імен змінних, типи даних для полів вибирати довільні.

7. Вивчити можливості редагування структури таблиць, для чого перейти на вкладку «Структура» та ознайомитися з доступними діями. Змінити тип поля для будь-якої зі створених таблиць, видалити поле, додати нове поле.

8. В меню зліва натиснути на посилання з ім’ям першої таблиці, зробивши її поточною. Далі перейти на вкладку «Вставить» та заповнити даними 4 рядки першої таблиці. Тип даних, що вводяться до стовпчика «Значение», має відповідати типу даних, заданих для цього поля при створенні.

9. Аналогічно заповнити по 4 рядки в другій та третій таблицях.

10. Вивчити можливості редагування даних таблиці, для чого перейти на вкладку «Обзор» та ознайомитися з доступними діями. Виконати редагування даних для другої таблиці.

11. Вивчити процедуру експорту БД. Експортувати можна дані окремої таблиці (зробивши її поточною) або експортувати всю БД (щоб зробити поточною БД, необхідно натиснути на посилання з ім’ям, що знаходиться в рядку над вкладками меню). Виконати експорт створеної БД в файл.

12. Вивчити процедуру імпорту. Для цього відкрити створений в результаті виконання попереднього пункту завдання файл та скопіювати

оператор CREATE TABLE, що відповідає третій таблиці. Перейти на вкладку SQL та вставити скопійований текст у вікно SQL-запитів. Далі необхідно змінити ім’я таблиці та виконати запит, натиснувши кнопку «ОК». Таблиця з новим ім’ям та структурою, що повторює структуру третої таблиці, з’явиться в БД.

13. Коректно завершити роботу з пакетом DENWER (ярлик Stop DENWER).

Зміст звіту

1. Назва та мета роботи.
2. Опис виконання роботи.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Призначення та основні можливості PHPMyAdmin.
2. Правила задання імен БД та таблиць у СУБД MySQL.
3. Основні типи даних у СУБД MySQL.
4. Призначення полів «Null» та «A_I» при створенні таблиці.
5. Призначення поля «Сравнение» при створенні таблиці

Література: [2, 3].

Лабораторна робота № 2

Тема. Створення та наповнення бази даних

Мета: отримання практичних навичок створення структури бази даних та наповнення її таблиць.

Короткі теоретичні відомості

Для виконання лабораторних робіт вибрано предметну область «Деканат ВНЗ». ER-діаграма предметної області (без вказання назв і типів зв’язків між сутностями) наведена на рис. 1.

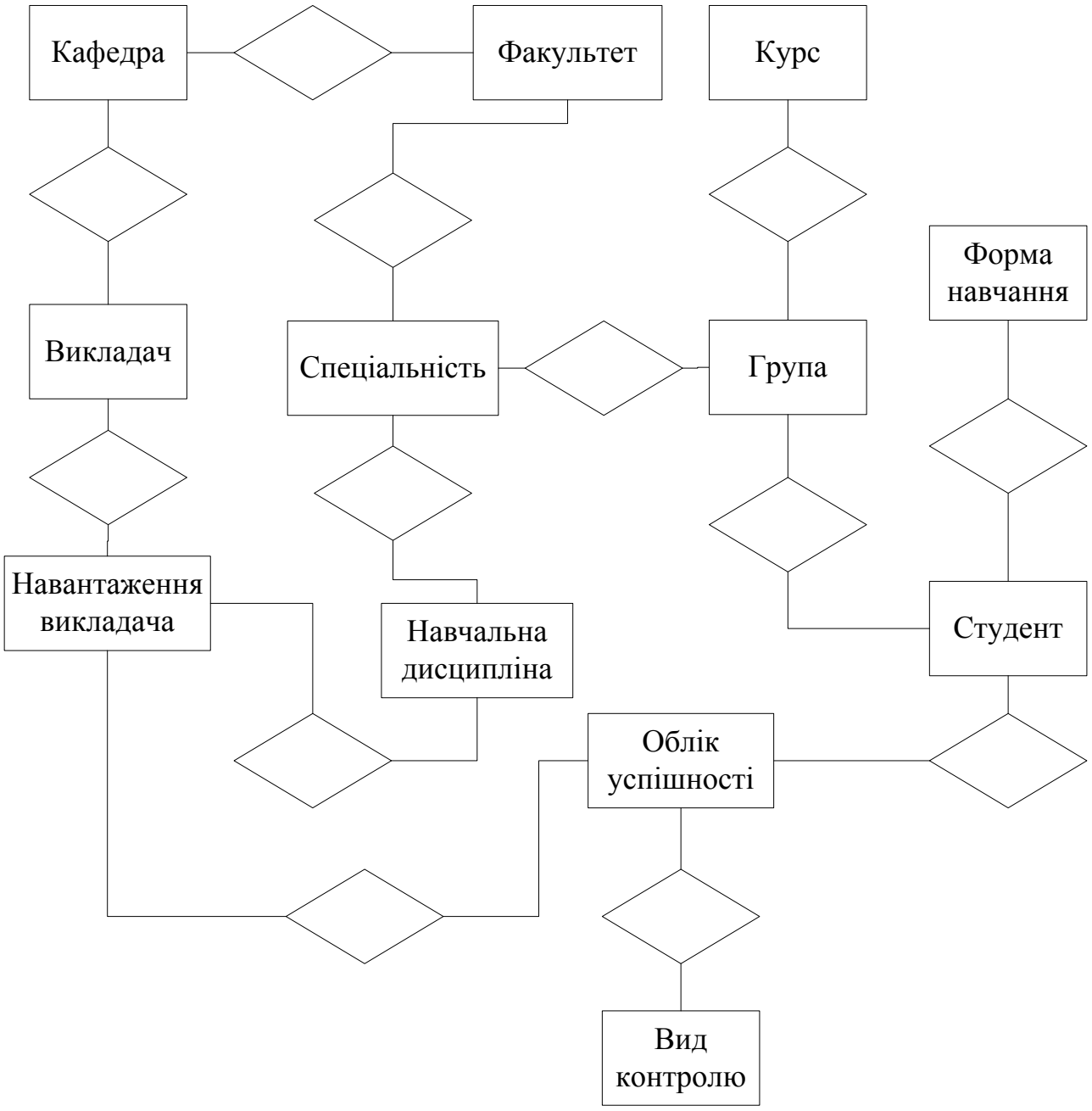

Рисунок 1 – ER-діаграма предметної області

Виділено такі сутності та їх атрибути:

– студент (номер залікової книжки, прізвище, ім’я, по батькові, стать, дата народження, рік вступу, рік закінчення, група, форма навчання, адреса, номер телефону);

– група (назва групи, спеціальність);

– курс (номер курсу);

– форма навчання (назва форми навчання);

– спеціальність (назва спеціальності, абревіатура, факультет);

– факультет (назва факультету, абревіатура, декан, номер кабінету, телефон деканату);

– кафедра (назва кафедри, абревіатура, факультет, завідувач кафедри, номер кабінету, телефон кафедри);

– викладач (прізвище, ім’я, по батькові, кафедра, номер телефону);

– навчальна дисципліна (назва навчальної дисципліни, абревіатура, спеціальність, для якої викладається, кількість годин);

– навантаження викладача (навчальна дисципліна, викладач, вид контролю, група, семестр, рік викладання);

– вид контролю (назва виду контролю);

– облік успішності (навчальна дисципліна, студент, дата складання, оцінка за національною шкалою, оцінка за 100-бальною шкалою, оцінка за шкалою ECTS).

Для більш повного опису атрибутів сутностей складають їх специфікації, де конкретизуть назви, типи даних та обмеження, що накладаються на значення атрибута.

Як приклад, розглянемо специфікацію атрибутів сутності «Студент», наведену в табл. 1.

До специфікації додано ще один атрибут – «Код студента», що необхідний для однозначної ідентифікації екземпляра сутності. Атрибут «Назва форми навчання» замінено на «Код форми навчання», оскільки це значення має братися з таблиці-довідника «Форма навчання» (що відображено відповідним зв’язком між сутностями «Студент» і «Форма навчання» на ER-діаграмі). На підставі специфікації складається структура таблиці для реалізації в середовищі конкретної СУБД (табл. 2). Значення зі стовпчика «Атрибут» стають назвами полів таблиці, а з огляду на значеня поля «Тип даних і формат» вибираються типи даних, підтримувані СУБД.

Таблиця 1 – Специфікація атрибутів сутності «Студент»

Атрибут	Тип даних і формат	Обмеження
Код студента	Цілий	Унікальний, не пустий
Номер залікової книжки	10 символів	Не пустий
Прізвище	50 символів	Не пустий
Ім’я	50 символів	Не пустий
По батькові	20 символів	
Стать	1 символ	Не пустий
Дата народження	Дата	Не пустий
Рік вступу	Дата	Не пустий
Рік закінчення	Дата	
Код групи	Цілий	Не пустий
Код форми навчання	Цілий	Не пустий
Адреса	100 символів	
Телефон	20 символів	

Таблиця 2 – Структура таблиці Student

№ поля	Найменування поля	Тип даних	Призначення поля
1	id_stud	INT	Унікальний код студента
2	zalikova_kn	VARCHAR(10)	Номер залікової книжки
3	prizv	VARCHAR(50)	Прізвище
4	imya	VARCHAR(50)	Ім’я
5	po_batkovi	VARCHAR(20)	По батькові
6	stat	VARCHAR(1)	Стать студента
7	data_nar	DATE	Дата народження
8	data_vst	DATE	Рік вступу
9	data_zak	DATE	Рік закінчення

Продовження табл. 2

№ поля	Найменування поля	Тип даних	Призначення поля
10	id_grupi	INT	Код групи
11	id_form_navch	INT	Код виду навчання
12	adresa	VARCHAR(100)	Адреса студента
13	tel	VARCHAR(20)	Телефон студента

Порядок виконання роботи

1. Доповнити ER-діаграму предметної області, наведену на рис. 1, вказавши назви та типи зв'язків між сутностями відповідно до опису, поданого в теоретичних відомостях. За необхідності внести зміни до структури ER-діаграми.

2. Скласти описи сутностей за зразком, поданим у табл. 1.

3. Скласти описи структури таблиць за зразком, поданим у табл. 2.

4. За допомогою додатка PHPMyAdmin створити БД *Dekanat* та таблиці, що відповідають описаним у теоретичних відомостях сутностям. Навести скріншот вмісту БД.

5. В таблицях з первинним ключем заповнити не менше 5 рядків, у підлеглих таблицях – не менше 10 рядків. Внесені дані повинні бути змістовними, несуперечливими та відповідати предметній області БД. Навести скріншоти заповнених таблиць.

Зміст звіту

1. Назва та мета роботи.

2. Опис виконання роботи.

3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Обмеження *PRIMARY KEY* та *FOREIGN KEY*.

2. Типи таблиць, що підтримуються в СУБД MySQL.

Література: [3, 5–7, 9, 10].

Лабораторна робота № 3

Тема. Вивчення операторів модифікації таблиць і даних

Мета: отримання практичних навичок використання команд модифікації таблиць і даних.

Короткі теоретичні відомості

За логічним призначенням оператори мови SQL поділяють на такі групи:

– мова визначення даних *DDL* (Data Definition Language);

– мова маніпулювання даними *DML* (Data Manipulation Language);

– мова контролю даних *DCL* (Data Control Language);

– мова роботи з транзакціями *TCL* (Transaction Control Language).

Мова визначення даних включає оператори, що керують об’єктами БД, до яких належать таблиці, індекси, представлення (команди *CREATE TABLE, DROP TABLE, ALTER TABLE*).

Мова маніпулювання даними включає оператори, що керують вмістом таблиць БД і дозволяють отримувати інформацію з цих таблиць SQL (команди *SELECT*, *UPDATE*, *DELETE* та *INSERT*).

Мова контролю даних забезпечує роботу з правами (команди *GRANT* та *REVOKE*).

Мова роботи з транзакціями забезпечує роботу з транзакціями (команди *START TRANSACTION, COMMIT* та *ROLLBACK*).

Отже, для модифікації таблиць з множини операторів мови SQL використовують команди *ALTER TABLE* та *CREATE TABLE*, а для модифікації даних – *UPDATE, DELETE* та *INSERT*. Розглянемо синтаксис цих команд.

Оператор *ALTER TABLE* забезпечує можливість змінювати структуру існуючої таблиці.

Наприклад, можна додавати або видаляти стовпці, створювати або видаляти індекси, перейменовувати стовпці або саму таблицю. Можна також змінювати коментар для таблиці та її тип.

Синтаксис оператора виглядає так:

ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]

alter_specification:

ADD [COLUMN] create_definition [FIRST | AFTER column_name]

або ADD [COLUMN] (create_definition, create_definition,...)

або ADD INDEX [index_name] (index_col_name,...)

або ADD PRIMARY KEY (index_col_name,...)

або ADD UNIQUE [index_name] (index_col_name,...)

або ADD FULLTEXT [index_name] (index_col_name,...)

або ADD [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)

[reference_definition]

або ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}

або CHANGE [COLUMN] old_col_name create_definition

[FIRST | AFTER column_name]

або MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]

або DROP [COLUMN] col_name

або DROP PRIMARY KEY

або DROP INDEX index_name

або DISABLE KEYS

або ENABLE KEYS

або RENAME [TO] new_tbl_name

або ORDER BY col

або table_options

Розглянемо приклади використання команди *ALTER TABLE*.

Приклад 1. Створення первинного індекса

Додамо в таблицю *Users* первинний ключ за стовпцем *Id*:

ALTER TABLE Users ADD PRIMARY KEY (Id)

Приклад 2. Створення зовнішнього індекса

Додамо в таблицю *Athing* зовнішній ключ з ім'ям *Athing_Fk1* за стовпцем *Created_by_user_id*, що посилається як на батьківський на стовпець *Id* таблиці *Users*:

ALTER TABLE Athing ADD CONSTRAINT Athing_Fk1 FOREIGN KEY (Created_by_user_id) REFERENCES Users(Id)

Приклад 3. Додавання нового стовпця

Додамо в таблицю *Customers* новий стовпець *Address*:

ALTER TABLE Customers

ADD Address VARCHAR (50) NULL;

В даному випадку стовпець *Address* має тип *VARCHAR* і для нього визначений атрибут *NULL*.

Приклад 4. Видалення стовпця

Видалимо стовпець *Address* з таблиці *Customers*:

ALTER TABLE Customers

DROP COLUMN Address;

Приклад 5. Зміна значення за замовчуванням

Встановимо в таблиці *Customers* для стовпця *Age* значення за замовчуванням 22:

ALTER TABLE Customers

ALTER COLUMN Age SET DEFAULT 22;

Приклад 6. Зміна типу стовпця

Змінимо в таблиці *Customers* тип даних у стовпці *FirstName* на *CHAR (100)* і встановимо для нього атрибут *NULL*:

ALTER TABLE Customers

MODIFY COLUMN FirstName CHAR (100) NULL;

При створенні зовнішнього індекса можна задати обмеження цілісності, що спрацюють при видаленні запису в батьківській таблиці:

– *ON DELETE RESTRICT* (запис у батьківській таблиці неможливо видалити, поки в підлеглій таблиці є хоча б один рядок, що посилається на цей запис);

– *ON DELETE SET NULL* (при видаленні запису в батьківській таблиці в полях відповідних записів підлеглої таблиці буде встановлено *NULL*-значення).

Створити первинний і зовнішній індекси можна також за допомогою команди *CREATE TABLE*. Синтаксис команди такий:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]
[table_options] [select_statement]

create_definition:
col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
[PRIMARY KEY] [reference_definition]
або PRIMARY KEY (index_col_name,...)
або KEY [index_name] (index_col_name,...)
або INDEX [index_name] (index_col_name,...)
або UNIQUE [INDEX] [index_name] (index_col_name,...)
або FULLTEXT [INDEX] [index_name] (index_col_name,...)
або [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)
[reference_definition]
або CHECK (expr)
type:
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
або SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
або MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
або INT[(length)] [UNSIGNED] [ZEROFILL]
або INTEGER[(length)] [UNSIGNED] [ZEROFILL]
або BIGINT[(length)] [UNSIGNED] [ZEROFILL]
або REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]

або DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]

або FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]

або DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]

або NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]

або CHAR(length) [BINARY]

або VARCHAR(length) [BINARY]

або DATE

або TIME

або TIMESTAMP

або DATETIME

або TINYBLOB

або BLOB

або MEDIUMBLOB

або LONGBLOB

або TINYTEXT

або TEXT

або MEDIUMTEXT

або LONGTEXT

або ENUM(value1,value2,value3,...)

або SET(value1,value2,value3,...)

index_col_name:

col_name [(length)]

reference_definition:

REFERENCES tbl_name [(index_col_name,...)]

[MATCH FULL | MATCH PARTIAL]

[ON DELETE reference_option]

[ON UPDATE reference_option]

reference_option:

RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:

TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }

або AUTO_INCREMENT = #

або AVG_ROW_LENGTH = #

або CHECKSUM = {0 | 1}

або COMMENT = "string"

або MAX_ROWS = #

або MIN_ROWS = #

або PACK_KEYS = {0 | 1 | DEFAULT}

або PASSWORD = "string"

або DELAY_KEY_WRITE = {0 | 1}

або ROW_FORMAT= { default | dynamic | fixed | compressed }

або RAID_TYPE= {1 | STRIPED | RAID0 } RAID_CHUNKS=# RAID_CHUNKSIZE=#

або UNION = (table_name,[table_name...])

або INSERT_METHOD= {NO | FIRST | LAST }

або DATA DIRECTORY="абсолютний шлях до каталога"

або INDEX DIRECTORY=" абсолютний шлях до каталога"

select_statement:

[IGNORE | REPLACE] SELECT ...

Розглянемо приклад створення індексів за допомогою *CREATE TABLE*.

1) необхідно створити таблицю з первинним ключем:

CREATE TABLE users (id int(11) not null auto_increment

username varchar(255) NOT NULL

password varchar(255) NOT NULL

active int NOT NULL

PRIMARY KEY (id))

ENGINE=InnoDB COLLATE=utf8_unicode_ci;

2) після цього можна створити підлеглу таблицю із зовнішнім ключем:

CREATE TABLE athing (id int(11) not null auto_increment

name varchar(255) not null

status varchar(255) not null

created_by_user_id int(11) not null

PRIMARY KEY (id)

CONSTRAINT athing_fk1 FOREIGN KEY (created_by_user_id)
REFERENCES users (id)

) ENGINE=InnoDB COLLATE=utf8_unicode_ci;

Щоб додати дані в таблицю БД використовують команду *INSERT*. Її синтаксис наступний:

INSERT [INTO] ім'я_таблиці [(список_стовпців)] VALUES (значення_1, значення_2, ... значення_N)

Після *INSERT INTO* в дужках вказують через кому список стовпців, в які необхідно додати дані, а після *VALUES* в дужках перераховують у тому самому порядку слідування стовпців значення, які в ці стовпці додаються.

Приклад використання команди *INSERT*, що додає 1 рядок в таблицю *Products*:

INSERT Products (ProductName, Manufacturer, ProductCount, Price)

VALUES ('iPhone X', 'Apple', 5, 76000);

Значення в стовпці будуть записані в такому порядку: в стовпчик *ProductName* буде записано рядок «iPhone X», у стовпчик *Manufacturer* – записано рядок «Apple» і т. д.

Важливо, щоб значення і типи даних відповідних стовпців співпадали. Тому, оскільки стовпець *ProductName* має тип varchar, значення, що буде записано в нього, береться в одинарні лапки, а стовпець *ProductCount* має тип *int*, то ж відповідне йому значення має бути цілим числом.

Команда UPDATE застосовується для зміни значення у вже наявних рядках.

Її синтаксис такий:

UPDATE ім'я_таблиці

SET стовпець_1=значення_1, стовпець_2=значення_2,... стовпець_N=значення_N

[WHERE умови_оновлення]

Приклади використання команди *UPDATE*:

– збільшити у всіх товарів ціну на 3000:

UPDATE Products

SET Price = Price +3000;

– змінити назву виробника з «Samsung» на «Samsung Inc.»:

UPDATE Products

SET Manufacturer = 'Samsung Inc.'

WHERE Manufacturer = 'Samsung';

– оновти відразу декілька стовпців:

UPDATE Products

SET Manufacturer = 'Samsung',

ProductCount = ProductCount + 3

WHERE Manufacturer = 'Samsung Inc.';

– установити значення за замовчуванням або *NULL*-значення, для цього при оновленні замість конкретних значень можна використовувати *DEFAULT* і *NULL*:

UPDATE Products

SET ProductCount = DEFAULT

WHERE Manufacturer = 'Huawei';

Якщо необхідно видалити дані з БД, використовують команду DELETE. Її синтксис такий:

DELETE FROM ім'я_таблиці

[WHERE умова_видалення]

Приклади використання команди *DELETE*:

– видалити рядки, для яких значення поля *Manufacturer* – «Huawei»:

DELETE FROM Products

WHERE Manufacturer = 'Huawei';

– видалити всі товари, виробником яких є Apple і які мають ціну менше 60000:

DELETE FROM Products

WHERE Manufacturer = 'Apple' AND Price <60000;

– видалити всі рядки незалежно від умови:

DELETE FROM Products;

Порядок виконання роботи

1. Створити в базі даних *Dekanat* за допомогою команди *CREATE TABLE* тимчасову таблицю *Temp_table_<N>* (<N> – порядковий номер студента в журналі групи), що має 4 поля: одне цілочисельне, два текстових і поле з датою. Текст команди занотувати.

2. За допомогою команди *ALTER TABLE* додати п’ятий стовпчик до таблиці *Temp_table_<N>*, перейменувати його, змінити тип даних і видалити третій стовпчик. Текст команд занотувати.

3. Створити в базі даних *Dekanat* за допомогою команди *ALTER TABLE* первинні та зовнішні ключі для таблиць відповідно до схеми, наведеної в лабораторній роботі № 2. Текст команд занотувати.

4. Перевірити працездатність створених зв’язків, спробувавши внести неприпустимі значення у відповідні поля підлеглих таблиць. Занотувати результати.

5. За допомогою команди *INSERT* додати в таблицю *Temp_table* чотири рядки даних. Текст команди занотувати.

6. За допомогою команди *UPDATE* змінити всі значення в текстовому полі таблиці *Temp_table* на значення *test*. Текст команди занотувати.

7. За допомогою команди *DELETE* видалити рядки таблиці *Temp_table*, для яких значення цілочисельного поля дорівнює 5. Текст команди занотувати.

8. За допомогою команди *DROP TABLE* видалити таблицю *Temp_table*. Текст команди занотувати.

Зміст звіту

1. Назва та мета роботи.
2. Опис виконання роботи.
3. Письмові відповіді на контрольні запитання.

Контрольні питання

1. Призначення та синтаксис оператора *CREATE TABLE*.
2. Призначення та синтаксис оператора *ALTER TABLE*.
3. Призначення та синтаксис оператора *DROP TABLE*.
4. Призначення та синтаксис оператора *INSERT*.
5. Призначення та синтаксис оператора *UPDATE*.
6. Призначення та синтаксис оператора *DELETE*.

Література: [1, 4, 8, 11–13].

Лабораторна робота № 4

Тема. Створення простих запитів

Мета: ознайомлення з базовим синтаксисом оператора *SELECT* та отримання практичних навичок складання простих (однотабличних) запитів.

Короткі теоретичні відомості

Запити – це фундамент мови SQL, і найчастіше її використовують саме як інструмент для створення запитів. Запит до БД будується за допомогою інструкції *SELECT*.

У процесі розвитку стандартів баз даних і стандарту мови SQL сформувалася певна концепція звертання користувача до БД з метою отримання з неї інформації. Користувач ніби просить надати ті чи інші значення стовпців з однієї або декількох таблиць, визначених оператором *FROM*, що відповідають умовам, визначеним оператором *WHERE*. Для зручності читання таких запитів людиною вони формуються як звичайне речення англійською мовою, а для зручності аналізу з боку СУБД ці речення підпорядковані певним правилам побудови.

Інструкція *SELECT* вибирає необхідну інформацію з БД і повертає її у

вигляді таблиці результатів запиту. Важливо зазначити, що оскільки SQL-запит повертає таблицю, то результати запиту можна записати назад в базу даних у вигляді таблиці. Це означає також, що результати двох запитів, що мають схожу структуру, можна об'єднати в одну таблицю. І, нарешті, це говорить про те, що результати запиту самі можуть стати предметом подальших запитів.

Повний синтаксис оператора *SELECT* є складним, однак його можна описати в скороченій формі:

SELECT список_стовпців_для_вибірки
[INTO нова_таблиця]
FROM таблиця
[WHERE умови_пошуку]
[GROUP BY умова_групування]
[HAVING умови_пошуку]
[ORDER BY умова_сортування [ASC | DESC]]

(Примітка: символами [ ] позначена необов'язкова частина, символом | – вибір (або).

Загалом інструкція складається з 6 операторів. Оператори *SELECT* і *FROM* є обов'язковими, 4 інших включаються лише за необхідності. Розглянемо призначення кожного з операторів.

1) Після оператора *SELECT* вказують список стовпців, які мають бути повернуті інструкцією *SELECT* (елементи списка розділяються комами). Ці стовпці можуть містити значення, вибрані із стовпців таблиць БД, або значення, що обчислюються під час виконання запиту (звідки вибирати дані).

2) Після оператора *FROM* через кому вказують список таблиць або інших запитів, які містять елементи даних, які добуваються цим запитом (які стовпчики вибирати).

3) Після оператора *WHERE* вказують умову відбору рядків, які слід вносити до результатів запиту (які рядки у вибраних стовпчиках вибирати).

4) Оператор *GROUP BY* дозволяє створити запит, що поверне підсумкові дані (т. зв. підсумковий запит). Звичайний запит видає в результатах по одному запису для кожного рядка з таблиці. Підсумковий запит спочатку групує рядки за певною ознакою, а потім видає в результатах запиту один підсумковий рядок для кожної групи.

5) Оператор *HAVING* вказує, що до результатів запиту слід вносити тільки деякі з груп, створених за допомогою оператора *GROUP BY*. Цей оператор, як і *WHERE*, використовує умову відбору.

6) Оператор *ORDER BY* використовують, щоб відсортувати результати запиту за одним або декількома стовпцями. Якщо цей оператор не вказано, результати запиту відсортовані не будуть.

Для отримання даних з усіх стовпців використовують знак «*», але варто зазначити, що його застосування вважається не дуже хорошою практикою. Більш оптимальний підхід полягає у перерахуванні через кому всіх необхідних стовпців після оператора *SELECT*. Виняток становить лише випадок, коли треба отримати дані про абсолютно всі стовпці таблиці або коли назви стовпців не відомі.

Після оператора *SELECT* можна вказати не лише назву стовпчика, але й будь-який вираз, наприклад, результат арифметичної операції. Наприклад можна вивести дані про назву продукта та загальну вартість наявних продуктів:

```
SELECT ProductName, Price*ProductCount
FROM Products;
```

Як результат виконання такого запиту буде створено два стовпці, причому другий стовпець міститиме значення стовпця *Price*, помножене на значення стовпця *ProductCount*, тобто сукупну вартість товару.

За допомогою оператора *AS* можна змінити оригінальну назву стовпця або визначити його псевдонім:

```
SELECT ProductName AS Title, Price * ProductCount AS TotalSum
FROM Products;
```

У цьому запиті для першого стовпця визначається псевдонім *Title*, хоча в

реальності ці дані будуть взяті зі стовпця *ProductName*. У другому стовпці, що має псевдонім *TotalSum*, буде виведено результат добутку значень зі стовпців *ProductCount* та *Price*.

Розглянемо приклади простих запитів до таблиці *Student*, що має поля *zalikova_kn* (номер залікової книжки) та prizv (прізвище, ім'я та по батькові студента).

1) Запит, що поверне всі елементи всіх рядків таблиці.

```
SELECT *
    FROM Student;
```

2) Запит, що поверне всі номери залікових книжок. Термінами реляційної алгебри можна сказати, що буде виконано проекцію.

```
SELECT zalikova_kn
    FROM Student;
```

3) Запит, що поверне всі елементи всіх рядків, де стовпчик prizv має значення 'Мазур'. Термінами реляційної алгебри можна сказати, що буде виконано вибірку, оскільки присутній оператор *WHERE*.

```
SELECT *
    FROM Student
    WHERE prizv = 'Мазур';
```

4) Запит, що поверне ті самі рядки, що й запит № 1, але результат буде відсортовано за прізвищем студента в зворотному порядку (Z-A), оскільки використано оператор *ORDER BY* із заначенням для сортування поля prizv (*ASC* – сортування за зростанням, *DESC* – сортування за зменшенням).

```
SELECT *
    FROM Student
    ORDER BY prizv DESC;
```

За допомогою оператора *DISTINCT* можна вибрати унікальні дані за певними стовпцями. Наприклад, необхідно вивести назви всіх виробників:

```
SELECT Manufacturer FROM Products;
```

Але в результаті цього запиту назви виробників будуть повторюватися,

якщо в таблиці зберігаються дані хоча б про 2 товари одного виробника. Щоб отримати лише унікальні значення, слід модифікувати запит так:

SELECT DISTINCT Manufacturer FROM Products;

Також можна задати умову для вибірки унікальних значень за кількома стовпцями:

SELECT DISTINCT Manufacturer, ProductCount FROM Products;

Цей запит поверне лише ті рядки, для яких пари значень *Manufacturer* та *ProductCount* будуть унікальними.

У запитах можна використовувати агрегатні функції, які обчислюють певні значення в наборі рядків. В MySQL є такі агрегатні функції:

– *AVG*: обчислює середнє значення;

– *SUM*: обчислює суму значень;

– *MIN*: обчислює найменше значення;

– *MAX*: обчислює найбільше значення;

– *COUNT*: обчислює кількість рядків в запиті.

Усі агрегатні функції як параметр використовують вираз, що являє собою критерій для визначення значень. Найчастіше як вираз виступає назва стовпчика, зі значеннями якого необхідно виконати обчислення.

Вирази в функціях *AVG* і *SUM* мають бути числовими значеннями (наприклад, стовпець числового типу). Вираз у функціях *MIN*, *MAX* і *COUNT* може бути числовим або рядковим значенням, а також датою.

Усі агрегатні функції за винятком *COUNT(*)* ігнорують значення *NULL*.

Розглянемо агрегатні функції докладніше.

1. *AVG*

Повертає середнє значення на діапазоні значень стовпця таблиці.

Наприклад, необхідно знайти середню ціну товарів з бази даних:

SELECT AVG (Price) AS Average_Price FROM Products;

Для пошуку середнього значення як вираз в функцію передається стовпець *Price*. Для отриманого в результаті обчислень значення встановлюється псевдонім *Average_Price*, хоча встановлювати псевдонім не

обов’язково.

На етапі вибірки можна застосовувати фільтрацію. Наприклад, необхідно знайти середню ціну для товарів певного виробника:

SELECT AVG (Price) FROM Products

WHERE Manufacturer = ‘Apple’;

Також можна знаходити середнє значення для складніших виразів. Наприклад, щоб знайти середню суму всіх товарів, враховуючи їх кількість, необхідно виконати запит:

*SELECT AVG (Price * ProductCount) FROM Products;*

2. *COUNT*

Обчислює кількість рядків у вибірці. Є дві форми цієї функції. Перша форма *COUNT(*)* підраховує кількість рядків у вибірці. Наприклад:

SELECT COUNT(*) FROM Products;

Друга форма функції обчислює кількість рядків за певним стовпцем (рядки зі значеннями *NULL* ігноруються). Наприклад:

SELECT COUNT (Manufacturer) FROM Products;

3. *MIN* і *MAX*

Функції обчислюють мінімальне та максимальне значення за стовпцем відповідно. Наприклад, якщо необхідно знайти мінімальну ціну серед товарів, необхідно виконати запит:

SELECT MIN (Price), MAX (Price) FROM Products;

MIN і MAX в MySQL

Ці функції також ігнорують значення *NULL* і не враховують їх при підрахунку.

4. *SUM*

Обчислює суму значень стовпця. Наприклад, щоб підрахувати загальну кількість товарів, слід виконати запит:

SELECT SUM (ProductCount) FROM Products;

Замість імені стовпця може передаватися вираз з обчислюваним значенням. Наприклад, щоб знайти загальну вартість всіх наявних товарів, слід

виконати запит:

*SELECT SUM (ProductCount * Price) FROM Products;*

5. *ALL* і *DISTINCT*

За замовчуванням усі розглянуті вище агрегатні функції для обчислення результату враховують усі рядки вибірки. Але вибірка може містити повторювані значення. Якщо необхідно виконати обчислення лише з використанням унікальних значень, виключивши повторювані дані, то для цього застосовується оператор *DISTINCT*. Наприклад:

SELECT COUNT (DISTINCT Manufacturer) FROM Products;

За замовчуванням замість *DISTINCT* застосовується оператор *ALL*, який вибирає всі рядки. Наприклад:

SELECT COUNT (ALL Manufacturer) FROM Products;

Оскільки цей оператор неявно мається на увазі, якщо оператор *DISTINCT* відсутній, то його можна не вказувати.

6. Комбінування функцій

Агрегатні функції в запиті можна комбінувати. Наприклад:

```
SELECT COUNT (*) AS ProdCount,
    SUM (ProductCount) AS TotalCount,
    MIN (Price) AS MinPrice,
    MAX (Price) AS MaxPrice,
    AVG (Price) AS AvgPrice
FROM Products
```

Щоб повернути не весь результат виконання запиту, а лише потрібну кількість рядків, використовують оператор *LIMIT*. Його синтаксис:

LIMIT [зміщення,] кількість_рядків

Якщо оператору LIMIT передається один параметр, то він задає кількість рядків, які будуть повернуті в результаті виконання запиту. Якщо вказано два параметри, то перший параметр встановлює зміщення відносно початку, тобто скільки рядків потрібно пропустити, а другий параметр вказує на кількість рядків, які будуть повернуті в результаті виконання запиту.

Наприклад, щоб отримати дані перших трьох рядків з таблиці *Products*, слід використати запит:

*SELECT * FROM Products*

LIMIT 3;

Запит, що поверне з таблиці *Products* рядки з 3-го до 5-го, має вигляд:

*SELECT * FROM Products*

LIMIT 2, 3;

В даному випадку будуть пропущені два перші рядки та виведено дані з наступних 3 рядків таблиці.

Для формування умови в операторі *WHERE* можна використовувати т.зв. оператори фільтрації:

1) оператор *IN*.

Визначає набір значень, які повинні мати стовпці. Його синтаксис:

WHERE вираз [NOT] IN (вираз)

Вираз у дужках після IN визначає набір значень. Цей набір може обчислюватися динамічно на підставі, наприклад, ще одного запиту, або це можуть бути конкретні значення (константи).

Приклад запиту, що виводить дані про товари, у яких виробник або Samsung, або Xiaomi, або Huawei:

*SELECT * FROM Products*

WHERE Manufacturer IN ('Samsung', 'HTC', 'Huawei');

Оператор *NOT*, навпаки, дозволяє вибрати всі рядки, стовпці яких не містять певних значень:

*SELECT * FROM Products*

WHERE Manufacturer NOT IN ('Samsung', 'HTC', 'Huawei');

2) оператор *BETWEEN*.

Дозволяє задати діапазон значень за допомогою початкового та кінцевого значення. Його синтаксис:

WHERE вираз [NOT] BETWEEN початкове_значення AND кінцеве_значення

Приклад запиту, що виводить дані про товари, у яких ціна від 2000 до 5000 (початкове і кінцеве значення також включаються в діапазон):

*SELECT * FROM Products*

WHERE Price BETWEEN 2000 AND 5000;

Якщо треба, навпаки, вибрати ті рядки, які не потрапляють в заданий діапазон, то додається оператор *NOT*. Наприклад:

*SELECT * FROM Products*

WHERE Price NOT BETWEEN 2000 AND 5000;

Також можна використовувати складніші вирази. Наприклад, отримаємо дані про товари з сукупною вартістю (ціна * кількість) в діапазоні від 9000 до 15000:

*SELECT * FROM Products*

*WHERE Price * ProductCount BETWEEN 9000 AND 15000;*

3) оператор *LIKE*.

Дозволяє задати шаблон рядка, якому має відповідати вираз. Його синтаксис:

WHERE вираз [NOT] LIKE шаблон_рядка

Для визначення шаблону застосовують спеціальні символи підстановки:

– символ %. Відповідає будь-якому підрядку, що може містити будь-яку кількість будь-яких символів, в тому числі і пустий підрядок.

Наприклад, вираз *WHERE ProductName LIKE 'Galaxy%'* відповідає таким значенням як «Galaxy Note» або «Galaxy S7»

– символ _. Відповідає будь-якому, але лише одному символу.

Наприклад, вираз *WHERE ProductName LIKE 'Galaxy S_'* відповідає таким значенням як "Galaxy S7" або "Galaxy S8".

Приклад використання оператора LIKE:

*SELECT * FROM Products*

WHERE ProductName LIKE 'iPhone%';

4) оператор *IS NULL*.

Дозволяє вибрати всі рядки, стовпці яких мають значення *NULL*.

Наприклад:

*SELECT * FROM Products*

WHERE ProductCount IS NULL;

Якщо додати оператор *NOT*, то можна вибрати рядки, стовпчики яких не містять значення *NULL*. Наприклад:

*SELECT * FROM Products*

WHERE ProductCount IS NOT NULL;

Результати виконання запиту можна відсортувати за одним чи декількоа полями. Для цього умову сортування задають після оператора *ORDER BY*. Наприклад, щоб відсортувати вибірку з таблиці Products за стовпцем Price, необхідно виконати такий запит:

*SELECT * FROM Products*

ORDER BY Price;

Також можна сортувати дані, використовуючи псевдонім стовпчика, який визначається за допомогою оператора *AS*. Наприклад:

*SELECT ProductName, ProductCount * Price AS TotalSum*

FROM Products

ORDER BY TotalSum;

Як критерій сортування можна використовувати вираз, утворений на основі значень стовпців. Наприклад:

SELECT ProductName, Price, ProductCount

FROM Products

*ORDER BY ProductCount * Price;*

За замовчуванням дані сортуються за зростанням, проте за допомогою оператора *DESC* можна задати сортування за зменшенням. Наприклад:

SELECT ProductName, ProductCount

FROM Products

ORDER BY ProductCount DESC;

За замовчуванням замість *DESC* використовується оператор *ASC*, який сортує за збільшенням.

Наприклад:

SELECT ProductName, ProductCount

FROM Products

ORDER BY ProductCount ASC;

Якщо необхідно виконати сортування відразу за кількома стовпцями, то ці стовпці слід вказати через кому після оператора *ORDER BY*. Наприклад:

SELECT ProductName, Price, Manufacturer

FROM Products

ORDER BY Manufacturer, ProductName;

Під час виконання такого запиту рядки спочатку будуть упорядковані за стовпцем *Manufacturer* за збільшенням.

Потім, якщо є два рядки, в яких у стовпці *Manufacturer* знаходиться однакове значення, вони сортуються за стовпцем *ProductName* також за збільшенням.

Але за необхідності за допомогою *ASC* і *DESC* можна окремо для різних стовпців визначити сортування за збільшенням і за зменшенням. Наприклад:

SELECT ProductName, Price, Manufacturer

FROM Products

ORDER BY Manufacturer ASC, ProductName DESC;

Для групування даних у запитах призначені оператори *GROUP BY* та *HAVING*.

1) Оператор *GROUP BY*. Він визначає, як рядки будуть групуватися.

Наприклад, щоб згрупувати товари за виробниками, слід виконати запит:

SELECT Manufacturer, COUNT(*) AS ModelsCount

FROM Products

GROUP BY Manufacturer

Стовпець *Manufacturer* представляє назву групи, а другий стовпець – *ModelsCount* представляє результат функції *COUNT*, яка обчислює кількість рядків у групі.

Якщо в операторі *SELECT* проводиться вибірка по одному або кількох

стовпцях і разом з цим використовуються агрегатні функції, то необхідно використовувати *GROUP BY*. Наприклад, наступний запит працювати не буде, оскільки він не містить виразу групування:

SELECT Manufacturer, COUNT (*) AS ModelsCount

FROM Products

Оператор *GROUP BY* може виконувати групування за декількома стовпцями. Наприклад, додамо до попереднього запиту групування за кількістю товарів:

SELECT Manufacturer, ProductCount, COUNT (*) AS ModelsCount

FROM Products

GROUP BY Manufacturer, ProductCount

Слід пам'ятати, що оператор *GROUP BY* має бути розташований після оператора *WHERE*, але перед оператором *ORDER BY*. Наприклад:

SELECT Manufacturer, COUNT (*) AS ModelsCount

FROM Products

WHERE Price> 30000

GROUP BY Manufacturer

ORDER BY ModelsCount DESC

GROUP BY в MySQL

2) Фільтрація груп. Оператор *HAVING*.

Оператор *HAVING* дозволяє виконати фільтрацію груп, тобто визначає, які групи будуть включені до результату виконання запиту.

Використання *HAVING* багато в чому подібне до використання *WHERE*. Але оператор *WHERE* застосовується для фільтрації рядків, а *HAVING* – для фільтрації груп.

Наприклад, щоб знайти всі групи товарів за виробниками, для яких визначено більше 1 моделі, необхідно виконати запит:

SELECT Manufacturer, COUNT (*) AS ModelsCount

FROM Products

GROUP BY Manufacturer

HAVING COUNT (*)> 1;

В одному запиті можна поєднувати оператори *WHERE* і *HAVING*. Наприклад:

SELECT Manufacturer, COUNT (*) AS ModelsCount

FROM Products

*WHERE Price * ProductCount>8000*

GROUP BY Manufacturer

HAVING COUNT (*)>1;

У цьому запиті спочатку фільтруються рядки: вибираються ті товари, загальна вартість яких більше 8000. Потім вибрані товари групуються за виробниками, а вже далі фільтруються самі групи – вибираються ті групи, які містять більше 1 моделі.

Якщо при цьому необхідно провести сортування, то оператор *ORDER BY* йде після оператора *HAVING*. Наприклад:

SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units

FROM Products

*WHERE Price * ProductCount>80000*

GROUP BY Manufacturer

HAVING SUM (ProductCount)>2

ORDER BY Units DESC;

У цьому запиті групування йде за виробниками, і також вибирається кількість моделей для кожного виробника (*Models*) і загальна кількість всіх товарів за усіма цими моделями (*Units*). У кінці групи відсортовано відповідно до кількості товарів за зменшенням.

Порядок виконання роботи

1. Виконайте наведені в теоретичних відомостях запити до таблиці Student. Результати занотуйте до звіту.

2. За прикладами, наведеними в теоретичних відомостях, складіть по 7 простих (однотабличних) запитів до кожної з трьох таблиць БД Dekanat відповідно до варіанта (вибирається за номером студента в журналі):

– №№ 1–5 запити до таблиць Спеціальність, Навчальна дисципліна, Викладач;

– №№ 6–10 запити до таблиць Викладач, Факультет, Облік успішності;

– №№ 11–15 запити до таблиць Облік успішності, Навантаження викладача, Кафедра.

– №№ 16–20 запити до таблиць Спеціальність, Облік успішності, Кафедра.

У запитах обов’язково мають бути використані оператори WHERE, GROUP BY, HAVING та ORDER BY. Запити та результати їх виконання занести до звіту.

3. За прикладами, наведеними в теоретичних відомостях, складіть вісім запитів до таблиць БД згідно з варіантом, у яких використовуються умови відбору з перевіркою на належність до діапазону та до множини, перевіркою на відповідність шаблону та з використанням агрегуючих функцій – підрахунком суми, кількості, середнього, максимального та мінімального значення. Запити та результати їх виконання занести до звіту.

Зміст звіту

1. Назва та мета роботи.

2. Опис виконання роботи.

3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Базовий синтаксис оператора SELECT.

2. Порівняння, перевірка на належність до діапазону та належність до множини в умові відбору.

3. Перевірка на відповідність шаблону та перевірка на рівність значенню NULL в умові відбору.

4. Агрегуючі функції.

Література: [1, 4, 8, 11–13].

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ

У 5-му семестрі студенти виконують 8 лабораторних робіт. Загальна кількість балів, яку отримують студенти за виконання та захист лабораторних робіт, становить 16 балів (максимально по 2 бали на кожну лабораторну роботу).

Шкала оцінювання знань студентів: національна та ECTS

<table>
<tr><td rowspan="2">Сума балів за всі види навчальної діяльності</td><td rowspan="2">Оцінка ECTS</td><td colspan="2">Оцінка за національною шкалою</td></tr>
<tr><td>Для іспиту, курсового проєкту (роботи), практики</td><td>Для заліку</td></tr>
<tr><td>90–100</td><td>A</td><td>Відмінно</td><td rowspan="5">Зараховано</td></tr>
<tr><td>82–89</td><td>B</td><td rowspan="2">Добре</td></tr>
<tr><td>74–81</td><td>C</td></tr>
<tr><td>64–73</td><td>D</td><td rowspan="2">Задовільно</td></tr>
<tr><td>60–63</td><td>E</td></tr>
<tr><td>35–59</td><td>FX</td><td>Незадовільно з можливістю повторного складання</td><td>Не зараховано з можливістю повторного складання</td></tr>
<tr><td>0–34</td><td>F</td><td>Незадовільно з обов’язковим повторним вивченням навчальної дисципліни</td><td>Не зараховано з обов’язковим повторним вивченням навчальної дисципліни</td></tr>
</table>

СПИСОК ЛІТЕРАТУРИ

1. Андон Ф., Резниченко В. Язык запросов SQL. Учебный курс. Санкт-Петербург : BHV, 2006. 416 с.
2. Артеменко Ю. Н. MySQL. Справочник по языку. Москва : Вильямс, 2005. 432 с.
3. Балик Н. Р., Мандзюк В. І. MySQL: лабораторний практикум: Посібник для студентів. Тернопіль : Навчальна книга – Богдан, 2008. 88 с.
4. Грофф Дж. Р., Вайнберг П. Н. SQL. Полное руководство : пер. с англ. Киев : BHV; Ирина, 2001. 813 с.
5. Дейт К. Дж. Введение в системы баз данных : пер. с англ. Москва : Вильямс, 2005. 1328 с.
6. Завадський І.О. Основи баз даних. Київ: Видавець І.О. Завадський, 2011. 192 с.
7. Избачков Ю. С., Петров В. Н., Васильев А. А., Телина И. С. Информационные системы : учебник для вузов. Санкт-Петербург : Питер, 2011. 544 с.
8. Кляйн К., Кляйн Д., Хант Б. SQL. Справочник : пер. с англ. Москва : КУДИЦ-ОБРАЗ, 2006. 832 с.
9. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. Санкт-Петербург : БХВ-Петербург, 2003. 1440 с.
10. Крёнке Д. Теория и практика построения баз данных : учеб. пособие. Санкт-Петербург : Питер, 2003. 800 с.
11. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс : пер. с англ. Москва : Вильямс, 2003. 1088 с.
12. Руководство по MySQL. [Електронний ресурс]. Режим доступу: https://metanit.com/sql/mysql.
13. SQL Tutorial – Tutorialspoint. [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/sql/index.htm

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Організація баз даних» для студентів денної форми навчання зі спеціальності 123 – «Комп’ютерна інженерія» освітнього ступеня «Бакалавр». Частина І

Укладачі : к. т. н. П. П. Костенко,
асист. Н. Л. Сохін

Відповідальний за випуск в. о. зав. кафедри «Комп’ютерні та інформаційні системи» доц. В. М. Сидоренко

Підп. до др. _____. Формат 60×84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ____. Наклад _____ прим. Зам. №_____. Безкоштовно.

Редакційно-видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600