МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

**до виконання лабораторних робіт засобами СУБД MySQL
для студентів спеціальності 122 «Комп'ютерні науки»**

Харків
2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт засобами СУБД MySQL для студентів спеціальності 122 «Комп'ютерні науки»

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від 29.06.2021

Харків
2022

Методичні вказівки до виконання лабораторних робіт засобами СУБД MySQL для студентів спеціальності 122 «Комп'ютерні науки»/ Уклад. О.Г. Сімонова, О.В. Охотська, І.Б. Шеліхова. – Харків: НТУ "ХПІ", 2022. – 40 с.

Укладачі: Ольга Сімонова
Олена Охотська
Інеса Шеліхова

Рецензент: Андрій Дашкевич, к.т.н., доц.

Кафедра геометричного моделювання та комп'ютерної графіки

ВСТУП

Теперішній етап розвитку систем управління базами даних (СУБД) характеризується появою нової технології доступу до даних Інтранет. Основна відмінність цього підходу від технології клієнт-сервер у тому, що відпадає необхідність використання спеціалізованого клієнтського програмного забезпечення. Для роботи з віддаленою базою даних використовується стандартний браузер Інтернету. При цьому вбудований у завантажуванні користувачем HTML-сторінки код, написаний зазвичай мовою Java, Java-script, PHP та інших, відстежує всі дії користувача за допомогою SQL-запитів до бази даних. Зручність цього підходу призвела до того, що він став використовуватися не тільки для віддаленого доступу до баз даних, але й для користувачів локальної мережі підприємства. Прості завдання обробки даних, не пов'язані зі складними алгоритмами, що вимагають узгодженої зміни даних у багатьох взаємопов'язаних об'єктах, досить просто та ефективно можуть бути побудовані за цією архітектурою.

У будь-якій сучасній СУБД, що підтримує реляційну модель, для обробки даних використовується стандартизована мова SQL, прообраз якої був створений в IBM в 70-х роках 20-го століття для перевірки можливостей моделі даних Едгара Кодда в рамках експериментальної СУБД System R. Спочатку мова називалася SEQUEL, але через конфлікт торгових марок пізніше була перейменована в SQL.

У своїй основі SQL є декларативною мовою, тобто дозволяє вказати, що хочете зробити – які дані з яких таблиць і в якому вигляді отримати, а як це робити – в якому порядку вихідні дані рахувати і як обробити для виведення бажаного результату – залишається на розсуд СУБД.

СУБД MySQL з'явилася як безкоштовна система, що працює на потужних, але не дуже дорогих персональних комп’ютерах, що надає в розпорядження користувачів значну обчислювальну потужність і широкий вибір операційних систем. Така СУБД дозволяє дістати доступ до баз даних організаціям й індивідуальним користувачам. Спілкуватися з СУБД MySQL можна за допомогою мов Perl і PHP та локального сервера Apache.

ЛАБОРАТОРНА РОБОТА №1
ОСНОВИ РОБОТИ З РЕЛЯЦІЙНИМИ БАЗАМИ ДАНИХ

Мета роботи: Отримати теоретичні знання і практичні навички реалізації баз даних (БД). Здійснити аналіз предметної області. Освоїти концептуальне проектування і навчитися визначати сутності й атрибути БД. Навчитися розробляти інфологічну модель БД у вигляді ER-діаграм. Отримати теоретичні знання і практичні навички при фізичному проектуванні баз даних. Навчитися створювати даталогічну модель БД.

У сучасній технології баз даних передбачається, що створення бази даних, її підтримка і забезпечення доступу користувачів до неї здійснюються централізовано за допомогою спеціального програмного інструментарію – системи управління базами даних. База даних (БД) – це пойменована сукупність даних, що відображає стан об'єктів та їх відносин у розглянутій предметній області. Об'єктом називається елемент предметної області, інформацію про який ми зберігаємо. Об'єкт може бути реальним (наприклад, людина, виріб; або населений пункт) і абстрактним (наприклад, подія, рахунок покупця або досліджуваний студентами курс).

Виконання роботи

1. **Обрати завдання**

Обрати варіант завдання, що відповідає номеру студента в списку навчальної групи. Для всіх наступних практичних робіт варіант залишається незмінним.

2. **Провести аналіз предметної області.**

Проєктування баз даних починається з аналізу предметної області обраного завдання за наведеним планом.

Теоретичні відомості. Предметною областю називається фрагмент реальності, який описується чи моделюється за допомогою БД і її додатків. У предметної області виділяються інформаційні об'єкти – ідентифікуються об'єкти реального світу, процеси, системи, поняття і т.д., відомості про які зберігаються в БД.

Аналіз предметної області доцільно розбити на три фази:

1. Аналіз концептуальних вимог та інформаційних потреб;

2. Виявлення інформаційних об'єктів і зв'язків між ними;

3. Побудова концептуальної моделі предметної області і проєктування концептуальної схеми БД.

Варіант 1.	Спроєктувати структуру бази даних про студентів для їх розподілу по місцях практики: прізвище, рік народження, стать, група, факультет, середній бал, місце роботи, місто.
Варіант 2.	Спроєктувати структуру бази даних про автомобілі: номер, рік випуску, марка, колір, стан, прізвище власника, адреса.
Варіант 3.	Спроєктувати структуру бази даних про квартири, призначені для продажу: район, поверх, площа, кількість кімнат, відомості про власника, ціна.
Варіант 4.	Спроєктувати структуру бази даних про книги, куплені бібліотекою: назва, автор, рік видання, адреса автора, адреса видавництва, ціна, книготорговельна фірма.
Варіант 5.	Спроєктувати структуру бази даних про співробітників, що мають комп'ютер: прізвище, номер кімнати, назва відділу, дані про комп'ютери.
Варіант 6.	Спроєктувати структуру бази даних про замовлення, отримані співробітниками фірми: прізвище, сума замовлення, найменування товару, назва фірми-клієнта, прізвище замовника.
Варіант 7.	Спроєктувати структуру бази даних про оцінки, отримані студентами на іспитах: прізвище, група, предмет, номер квитка, оцінка, викладач.
Варіант 8.	Спроєктувати структуру бази даних про викладачів кафедри: прізвище, посада, ступінь, номер кімнати, курси, що викладаються.
Варіант 9.	Спроєктувати структуру бази даних про авторів web-сайту і їх статті: ім'я, адреса, обліковий запис, пароль, тема, заголовок, текст статті, ілюстрації.
Варіант 10.	Спроєктувати структуру бази даних про список розсилки і передплатників: тема і зміст листа, дата відправки, імена та адреси передплатників, їх облікові записи і паролі.

2.1 Аналіз концептуальних вимог та інформаційних потреб

Вимоги користувачів до розроблюваної БД є списком запитів із зазначенням їх інтенсивності та обсягів даних. Ці відомості розробники БД отримують в діалозі з її майбутніми користувачами. Тут же з'ясовуються вимоги до введення, оновлення та коригування інформації. Вимоги користувачів уточняються і доповнюються під час аналізу наявних і перспективних завдань.

2.2 Виявлення інформаційних об'єктів і зв'язків між ними

Друга фаза аналізу предметної області полягає у виборі інформаційних об'єктів, завданні необхідних властивостей для кожного об'єкта, виявленні зв'язків між об'єктами, визначенні обмежень, що накладаються на інформаційні об'єкти, типи зв'язків між ними, характеристики інформаційних об'єктів.

Всі інформаційні об'єкти предметної області пов'язані між собою. Відповідності, відносини, що виникають між об'єктами предметної області, називаються зв'язками. Розрізняють чотири типи зв'язку: «один до одного» (1: 1); «Один до багатьох» (1: М); «Багато до одного» (М: 1); «Багато до багатьох» (М: М).

При відборі зв'язків, які розміщують в ИЛМ (інформаційно-логічну модель), слід керуватися інформаційними потребами користувачів бази даних. Кожен зв'язок характеризується ім'ям, типом, класом приналежності і напрямком. Ім'я зв'язку має бути дієслівним оборотом, наприклад «Належить», «Закріплені за», «Входить в» і т.д.

2.3 Побудова інфологічної структури (концептуальної моделі) предметної області

Заключна фаза аналізу предметної області полягає в проектуванні її інфологічної структури або концептуальної моделі. Концептуальна модель включає в себе описи об'єктів і їх взаємозв'язків, що представляють інтерес в аналізованій предметній області (ПО) і виявляються в результаті аналізу даних. Концептуальна модель застосовується для структурування предметної області з урахуванням інформаційних інтересів користувачів системи. Концептуальна модель є представленням точки зору користувача на предметну область і не залежить ні від програмного забезпечення СУБД, ні від технічних рішень. Можуть змінюватися прикладні програми, що обробляють дані, може змінюватися організація їх фізичного зберігання, концептуальна модель залишається незмінною або збільшується з метою включення додаткових даних.

Однією з поширених моделей концептуальної схеми є модель «сутність–зв'язок» (ER-модель або ER-діаграма). Основними конструкціями даної моделі є сутності та зв'язки. Під сутністю розуміють основний зміст об'єкта ПО, про який збирають інформацію. В якості сутності можуть виступати місце, річ, особа, явище і т.п. Екземпляр сутності – конкретний об'єкт.

Даталогічна модель являє собою опис бази даних, виконаний в термінах використовуваної СУБД. Найбільш часто при розробках баз даних застосовують реляційні СУБД. Для СУБД цього типу даталогічну модель зручно представити у вигляді набору таблиць спеціальної форми. Така таблиця складається для кожної відносини, використовуваної в базі даних. Відносини в базі відповідають класам об'єктів з інфологічної моделі. Крім того, відносини можуть представляти деякі зв'язки предметної області. Кожній таблиці потрібно поставити у відповідність її ключі. Схема ключа являє собою перерахування атрибутів відносин, що становлять ключ. Розрізняють прості і складні ключі. Простий ключ будується на основі одного атрибута. Складний ключ будується на базі використання декількох атрибутів. Ключі прийнято розділяти на первинні, зовнішні і допоміжні. Первинний індекс повинен бути тільки один для кожної таблиці.

Значення атрибутів, використовуваних для формування первинного ключа, повинні бути унікальними для кожного запису в таблиці. Значення первинного ключа унікально ідентифікують кожен запис. Не може бути двох записів в таблиці з однаковим значенням первинного ключа. Зовнішні ключі використовуються для реалізації зв'язків типу 1:М між відносинами. Зовнішній ключ будується для відносини, що знаходиться на стороні «багато» зв'язку 1:М. Для кожного такого відношення на даталогічній моделі повинен бути показаний зовнішній ключ. Зовнішній ключ завжди повинен мати відповідний йому первинний ключ відношення, що знаходиться на стороні «один» зв'язку типу 1:М. Для пов'язаних ключа «зовнішній – первинний» з'єднуються на схемі даталогічної моделі ламаною лінією. Зовнішніх ключів може бути кілька для однієї таблиці. Слід зазначити, що первинні та зовнішні ключі будуються як правило на основі цілочисельних атрибутів, а не атрибутів, що мають строковий або речовинний тип. Крім первинних і зовнішніх ключів часто використовують допоміжні індекси. Вони застосовуються для реалізації зв'язків, отримання потрібного впорядкування при виведенні на екран і створенні звітів і т.д. У кожному разі використання допоміжного індексу, його необхідність повинна бути обґрунтована. Застосування великої кількості індексів

уповільнює роботу СУБД, тому що операції над записами відношення потребують корегування всіх індексів.

3. Виконати практичну частину роботи

Заповнити наведені таблицях 1-3 з метою систематизації інформаційного змісту предметної області (ПО).

Таблиця 1 – Список сутностей предметної області

N п.п.	Наименование сущности	Краткое описание

Тут слід привести опис основних сутностей (об'єктів) ПО. Відбір сутностей проводиться на основі аналізу інформаційних потреб.

Таблиця 2 – Список атрибутів

N п.п.	Наименование атрибута	Краткое описание

Тут наводиться відбір атрибутів (не менше 5-ти) для кожного екземпляра сутності. Відбираються тільки ті атрибути сутностей, які необхідні для формування відповідей на регламентовані і непередбачені запити. Для кожного об'єкта слід привести таблиці його атрибутів.

Таблиця 3– Список зв’зків

N п.п.	Наименование связи	Сущности, участвующие в связи	Краткое описание

На основі аналізу інформаційних запитів слід виявити зв'язки між сутностями. Для виявлених зв'язків також потрібно заповнити таблицю 1.3.

3.2 Побудувати інфологічну модель.

Інфологічне (концептуальне) проектування – побудова семантичної моделі предметної області, тобто інформаційної моделі найвищого рівня абстракції. Така модель створюється без орієнтації на будь-яку конкретну СУБД та модель даних.

Конкретний вигляд та зміст інфологічної моделі бази даних визначається обраним для цього формальним апаратом. Зазвичай використовуються графічні нотації, подібні до ER-діаграм.

Найчастіше інфологічна модель бази даних включає:

- опис інформаційних об'єктів або понять предметної галузі та зв'язків між ними.
- опис обмежень цілісності, тобто вимог до допустимих значень даних та зв’язків між ними.

На підставі раніше обраного варіанту і таблиць 1-3 описати класи об'єктів (сутностей) і їх властивості; розставити існуючі зв'язки між ними; на підставі табл. 3 в письмовій формі обгрунтувати типи зв'язків (1:1, 1:M і т.д.).

Інфологічну модель краще репрезентувати графічно, із зображенням всіх таблиць і зв'язків між ними. На рисунку 1 представлений приклад графічного представлення схеми зв'язків.

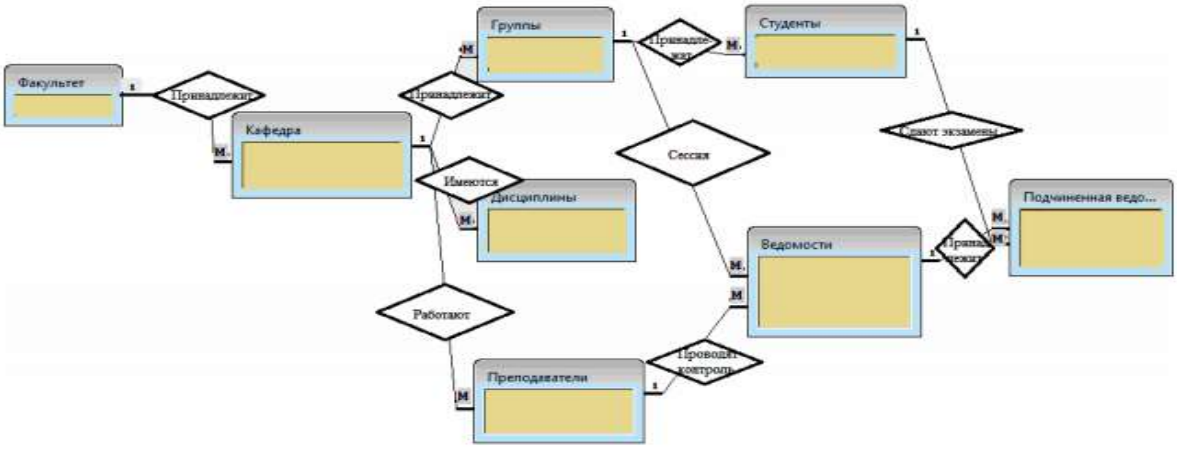


Рисунок 1 - Графічне представлення схеми зв'язків

При графічній побудові інфологічної моделі слід дотримуватися єдиного масштабу для всієї схеми. Всі прямокутники, що позначають класи об'єктів, повинні бути одного розміру. Аналогічно, всі ромби з іменами зв'язків також повинні мати однаковий розмір.

3.3 Описати даталогічну модель.

Даталогічне проектування - створення схеми бази даних на основі конкретної моделі даних, наприклад, реляційної моделі даних. Для реляційної моделі даних даталогічна модель – набір схем відносин, зазвичай із зазначенням первинних ключів, і навіть «зв'язків» між відносинами, які становлять зовнішні ключі.

Перетворення інфологічної (концептуальної) моделі на даталогічну (логічну) модель, як правило, здійснюється за формальними правилами. Цей етап може бути значною мірою автоматизований.

На етапі логічного проектування враховується специфіка конкретної моделі даних, але може не враховуватися специфіка конкретної СУБД.

На підставі раніше обраного варіанту і таблиць 1-3, інфологічної моделі необхідно: провести відповідність ключів для кожної таблиці 1-3; заповнити для кожної таблиці БД форму, згідно табл. 4; побудувати даталогічну модель БД.

Таблиця 4 – Структура таблиці для даталогічної моделі

N п.п.	Наименование реквизита	Идентификатор	Тип	Длина	Формат изображения	Ограничения и комментарий

На рисунку 2.2 наведений приклад побудови даталогічної моделі.

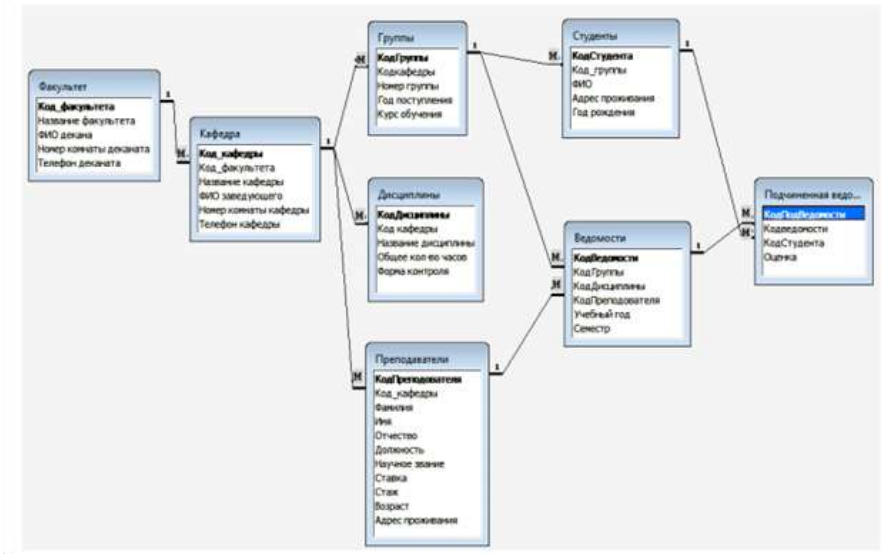


Рисунок 2.2 - Приклад побудови даталогічної моделі

3.4. Описання фізичної моделі даних.

Фізичне проектування - створення схеми бази даних для конкретної СУБД. Специфіка конкретної СУБД може включати обмеження на ім'я об'єктів бази даних, обмеження на підтримувані типи даних та інше. Крім того, специфіка конкретної СУБД при фізичному проектуванні включає вибір рішень, пов'язаних з фізичним середовищем зберігання даних (вибір методів управління дисковою пам'яттю, поділ БД за файлами та пристроями, методів доступу до даних), створення індексів тощо.

Етап фізичного проєктування тісно пов'язаний з етапом реалізації БД. На цьому етапі вирішуються питання, пов'язані з продуктивністю системи, визначаються структури зберігання даних та методи доступу.

Процес проєктування БД не може бути автоматичним, так як для вирішення багатьох проблем участь людини є обов'язковою.

Завдання розробника БД полягає у структуризації даних таким чином, щоб усунути непотрібне дублювання та забезпечити швидкий шлях пошуку необхідної інформації

При проектуванні БД можуть виникнути небажані властивості, такі як надмірність, аномалії оновлення, аномалії включення, аномалії видалення та інших. Для зменшення небажаних показників БД до схем відносин застосовують процедури нормалізації.

Нормалізація - це розбиття таблиці на дві або більше таблиць, які мають кращі властивості при включенні, змінах та видаленні даних. Остаточна мета нормалізації зводиться до отримання такого проєкту БД, у якому виключено надмірність інформації. Це робиться не тільки з метою економії пам'яті, а більше для виключення можливої суперечливості даних, що зберігаються.

Теорія нормалізації полягає в концепції нормальних форм. Кажуть, що таблиця знаходиться у цій нормальній формі, якщо вона задовольняє певний набір вимог. Теоретично існує п'ять нормальних форм, але практично використовуються лише перші три. Перші дві нормальні форми є проміжними кроками для приведення БД до третьої нормальної форми.

4. Питання для самоперевірки

1. Які етапи проектування бази даних?
2. Що таке інфологічне моделювання?
3. Що є результатом інфологічного моделювання?
4. Що таке даталогічне моделювання?
5. Що є результатом даталогічного моделювання?
6. Що таке фізичне моделювання?
7. Що є результатом фізичного моделювання?
8. Для чого робиться нормалізація?

ЛАБОРАТОРНА РОБОТА №2
СТВОРЕННЯ ТА ЗМІНЕННЯ БАЗИ ДАНИХ ТА ТАБЛИЦІ «РОЗКЛАД РУХУ ЛІТАКІВ»

Мета практичної роботи: Ознайомитися з можливостями клієнтської програми MySQL, яка представляє собою утиліту командного рядка. Створити з її допомогою базу даних, набір таблиць в ній, заповнити таблиці даними для подальшої роботи, провести модифікацію таблиць.

Виконання роботи.

1. ***Порядок запуску СУБД MySQL*****.**

Запустити локальний сервер Denwer:

Робочий стіл | ярлик Start Denwer ►вікно запуску Denwer; дочекатися закриття вікна

Відкрити середу PHPMyAdmin:

Запустити браузер

► адресний рядок ← http://localhost/denwer

► знайти http://localhost/Tools /phpMyAdmin

► вікно phpMyAdmin; Server: localhost

Після того як сервер MySQL запущений, до нього можна підключитися.

Відкрити вікно командного рядка. Вивести список наявних баз даних. Команда **SHOW DATABASES**.

2. Ознайомитися з набором команд мови SQL, пов'язаних зі створенням бази даних, створенням, модифікацією структури таблиць і їх видаленням, вставкою, модифікацією і видаленням записів таблиць:

Створення бази даних MySQL (CREATE DATABASE)
Видалення бази даних MySQL (DROP DATABASE)
Створення таблиці в базі даних MySQL (CREATE TABLE)
Видалення таблиці з бази даних MySQL (DROP TABLE)
Змінення властивостей таблиці:

- перейменування таблиці (ALTER TABLE RENAME)
- вставка стовпців (ALTER TABLE ADD)
- змінення властивостей стовпця (ALTER TABLE CHANGE)
- Видалення стовпців (ALTER TABLE DROP)

Вставка рядка в таблицю (INSERT INTO)
Видалення рядка із таблиці (DELETE FROM)

Оновлення записів в таблиці (UPDATE)

2.1 Створити БД «Аеропорт»

Створення бази даних виконується за допомогою команди

```
CREATE DATABASE [IF NOT EXISTS] імя_бази_данних
```

Команда **CREATE DATABASE** створює базу даних з вказаним ім'ям. Для використання команди необхідно мати привілей CREATE для бази даних. Якщо база даних з таким ім'ям існує, генерується помилка.

Специфікація_create:

```
[DEFAULT] CHARACTER SET – ім’я_набора_символів
[DEFAULT] COLLATE – ім’я_порядка_зіставлення
```

Опція специфікація_create може вказуватися для визначення характеристик бази даних. Характеристики бази даних зберігаються в файлі db.opt, розташованому в каталозі даних. Конструкція **CHARACTER SET** визначає набір символів для бази даних за замовчуванням. Конструкція **COLLATION** задає порядок зіставлення за умовчанням.

Оскільки спочатку в базі немає ніяких таблиць, оператор **CREATE DATABASE** тільки створює підкаталог в каталозі даних **MySQL**.

Примітка: Команда закінчується символом крапки з комою.

Створити базу даних з ім'ям AIRPORT.

```
CREATE DATABASE AIRPORT;
```

Сервер відповість, що запит оброблений, База даних була успішно створена.

2.2 Визначити поточну базу даних.

База даних AIRPORT вже створена. Для роботи з нею, необхідно її «активувати» або «вибрати». Всякий раз при роботі з клієнтом **MySQL** необхідно визначати, яка база даних буде використовуватися.

Визначити поточну базу даних можна кількома способами:

- якщо працюєте з командного рядка без phpMyAdmin:
  - при запуску: `mysql> MySQL AIRPORT;`
  - за допомогою оператора USE: `mysql> USE AIRPORT;`
  - за допомогою \ u: `mysql> \ u AIRPORT;`
  - Після визначення бази даних вводимо команду `SELECT DATABASE ();` і бачимо нашу БД.
- якщо працюєте в phpMyAdmin:
  - Переконайтеся, що база даних створена (**SHOW DATABASES**).

Зробіть базу даних поточною.

2.3 Створити таблицю «Розклад»

Створення таблиці проводиться командою **CREATE TABLE**.

Бази даних зберігають дані в таблицях. Найпростіше таблиці можна уявляти собі, як складаються з рядків і стовпців. Кожен стовпець визначає дані певного типу. Рядки містять окремі записи.

Синтаксис команди **CREATE TABLE** такий:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] ім'я
[(Специфікація, ...)] [опція, ...]
[[IGNORE | REPLACE] запит]
```

Прапор TEMPORARY задає створення тимчасової таблиці, яка існує протягом поточного сеансу. По завершенні сеансу таблиця віддаляється. Тимчасовим таблицями можна присвоювати імена інших таблиць, роблячи останні тимчасово недоступними. Специфікатор **IF NOT EXIST** пригнічує виведення повідомлень про помилки в разі, якщо таблиця з вказаним ім'ям вже існує. Імені таблиці може передувати ім'я бази даних, відокремлене крапкою. Якщо це не зроблено, таблиця буде створена в базі даних, яка встановлена за замовченням.

Дозволяється створювати таблиці без стовпців, проте в більшості випадків специфікація хоча б одного стовпчика все ж таки присутня. Специфікації стовпців і індексів наводяться в круглих дужках і розділяються комами.

Формат специфікації наступний:

```
ім'я тип [NOT NULL | NULL] [DEFAULT значення]
[AUTO_INCREMENT] [KEY] [посилання]
```

Специфікація типу включає назву типу та його розмірність. За замовчуванням стовпці приймають значення NULL. Специфікатор NOT NULL забороняє подібну поведінку.

У будь-якого стовпця є значення за замовчуванням. Якщо воно не вказано, програма **MySQL** вибере його самостійно. Для стовпців, які приймають значення NULL, значенням за замовчуванням буде NULL, для строкових стовпців – порожній рядок, для численних стовпців – нуль. Змінити цю установку дозволяє оператор **DEFAULT**.

Поля-лічильники, що створюються за допомогою прапора AUTO_INCREMENT, ігнорують значення за замовчуванням, так як в них

записуються порядкові номери. Тип лічильника повинен бути беззнаковим цілим. У таблиці може бути присутнім лише одне поле-лічильник. Їм не обов'язково є первинний ключ. Коли **MySQL** зустрічається зі стовпцем з атрибутом AUTO_INCREMENT, то генерується нове значення, яке на одиницю більше, ніж найбільше значення в стовпці. Тому ми не повинні ставити для цього стовпця значення, MySQL генерує їх самостійно. З цього також випливає, що кожне значення в цьому стовпці буде унікальним.

Примітка: в MySQL команди і імена стовпців розрізняють регістр символів, однак імена таблиць і баз даних можуть залежати від регістру в зв'язку з використовуваної платформою (як в Linux).

Ключовий стовпець (primary key) необхідний для того, щоб виключити можливість збігу даних. Якщо є стовпець з унікальними значеннями, то можна легко розрізнити два записи. Краще доручити присвоювання унікальних значень самій системі MySQL.

Після створення таблиці переконаємося, що вона існує командою

SHOW TABLES;

3. Виконання практичної частини роботи:

3.1 Створити таблицю «Розклад» зі структурою:

Номер рейсу	Ціле
Дні тижня	Текст (8)
Пункт відправлення	Текст (24)
Час відльоту	Час
Пункт призначення	Текст (24)
Час прибуття	Час
Тип літака	Текст (8)
Вартість квитка	Валюта

та ввести її разом з даними в БД «Аеропорт».

3.2 Внесення даних в таблицю «Розклад». Оператор INSERT

Оператор **INSERT** використовується для заповнення таблиці даними.

```
INSERT into table_name (column1, column2, ...) values (value1, value2 ...);
```

де:

table_name - ім'я таблиці, в яку треба внести дані;

column1, column2 ... - імена стовпців,

value1, value2 ... - значення для відповідних стовпців.

Важливо:

- значення стовпців, які є текстовими рядками, записуються в лапках;
- значення для стовпця з властивістю auto_increment задає система **MySQL**, яка знаходить в стовпці найбільше значення, збільшує його на одиницю, і вставляє нове значення.

Якщо наведена вище команда правильно введена в запрошенні клієнта **MySQL**, то програма виведе повідомлення про успішне виконання. Створення додаткових записів вимагає використання окремих операторів INSERT. Щоб полегшити цю роботу можна помістити всі оператори INSERT в файл. Це повинен бути звичайний текстовий файл з оператором INSERT в кожному рядку.

Введення даних з файлу – команда LOAD DATA

Команда **LOAD DATA INFILE** читає рядки з текстового файлу і вставляє їх в таблицю з дуже високою швидкістю.

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
[REPLACE | IGNORE] INTO TABLE tbl_name
[FIELDS [TERMINATED BY '\ t']
[[OPTIONALLY] ENCLOSED BY ''] [ESCAPED BY '\\']]
[LINES TERMINATED BY '\ n'] [IGNORE number LINES]
[(Col_name, ...)]
```

Якщо задано ключове слово **LOCAL**, то файл читається з клієнтського хоста. Якщо ж LOCAL не вказується, то файл повинен знаходитися на сервері. Якщо текстові файли, які потрібно прочитати, знаходяться на сервері, то з міркувань безпеки ці файли повинні або розміщуватися в директорії бази даних, або бути доступними для читання всім користувачам. Крім того, для застосування команди LOAD DATA INFILE до серверних файлів необхідно володіти привілеями FILE для серверного хоста.

Внести в таблицю «Розклад» наступні дані:

Номер рейсу	**Дні тижня**	**Пункт відправлення**	**Час вильоту**	**Пункт призначення**	**Час прибуття**	**Тип літака**	**Вартість квитка**
Дані							
138	2_4_7	Баку	21.12	Москва	0.52	ІЛ-86	115.00
57	3_6	Єреван	7.20	Київ	9.25	ТУ-154	92.00
1234	2_6	Казань	22.40	Баку	23.50	ТУ-134	73.50
242	1 по 7	Київ	14.10	Москва	16.15	ТУ-154	57.00
86	2_3_5	Мінськ	10.50	Сочі	13.06	ІЛ-86	78.50
137	1_3_6	Москва	15.17	Баку	18.44	ІЛ-86	115.00
241	1 по 7	Москва	9.05	Київ	11.05	ТУ-154	57.00
577	1_3_5	Рига	21.53	Таллінн	22.57	АН-24	21.50
78	3_6	Сочі	18.25	Баку	20.12	ТУ-134	44.00
578	2_4_6	Таллінн	6.30	Рига	7.37	АН-24	21.50

Після створення таблиці переконатися, що вона існує командою:
SHOW TABLES;

4. Завдання для самостійної роботи

Виконати завдання, використав команди ALTER TABLE, INSERT, DELETE, UPDATE.

Змінити властивості таблиці:

- перейменувати таблицю
- вставити стовпець (наприклад, «Тривалість_польоту», тип даних - ціле);
- змінити властивості стовпця (перейменувати, змінити тип даних);
- видалити стовпець.

Вставити рядок в таблицю.

Видалити рядок з таблиці.

Оновити дані в таблиці.

5. Питання для самоперевірки

1. Яким чином здійснюється видалення таблиці з бази даних **MySQL**?
2. Як змінити властивості таблиці?
3. Яким оператором здійснюється перейменування таблиці?
4. Яку дію виконує оператор **ALTER TABLE ADD**?
5. Для чого використовується оператор **ALTER TABLE CHANGE**?
6. Який оператор здійснює видалення стовпців?
7. Що робить оператор **INSERT INTO**?
8. Як видалити рядки з таблиці?
9. За допомогою якого оператора оновлюються записи в таблиці?

ЛАБОРАТОРНА РОБОТА №3
БАЗОВІ ОПЕРАЦІЇ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

Мета лабораторної роботи: Створення бази даних ***«Magazin»***, яка містить чотири взаємопов'язаних таблиці у відповідності до реляційної моделі даних. Підготовка і реалізація серії запитів, пов'язаних з вибіркою інформації і модифікацією даних таблиць.

Виконання роботи.

1. Створення бази даних

1.1. Створити базу даних «Magazin»

1.2. Створити таблицю «Producty»

	Поле	Тип	Сравнение	Атрибуты	Null	По умолчанию	Д
☐	id_Prod	int(11)			Нет	None	
☐	id_grProd	int(11)			Нет	None	
☐	Prod	varchar(20)	cp1251_general_ci		Нет	None	
☐	Cena	decimal(10,2)			Нет	None	

cp1251_general_ci :
1251– шифр принятой кодовой таблицы,
general – сравнение без учета регистров.

1.3. Заповнити таблицю «Producty».

Id_Prod	Id_grProd	Prod	Cena
100	1	Яблука	6,5
200	2	Картопля	5,2
101	1	Груші	9,5
300	3	Гречка	11,5
202	2	Морква	4,0
301	3	Рис	8,7
201	2	Цибуля	4,5

Id_Prod – первичный ключ.

1.4. Створити таблицю «grProd»

	Поле	Тип	Сравнение	Атрибуты	Null	По умолчанию	Доп
☐	id_grProd	int(11)			Нет	*None*	
☐	gr_name	varchar(20)	cp1251_general_ci		Нет	*None*	

Id_grProd – первинний ключ

1.5. Заповнити таблицю «grProd».

Id_grProd	gr_name
3	Крупи
1	Фрукти
2	Овочі

1.6. Створити таблицю «Zakazy»

Поле	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительн
Id_Zak	int(11)			Нет	*None*	auto_increment
Id_Pokup	int(11)			Нет	*None*	
Id_Prod	int(11)			Нет	*None*	
Kol	decimal(10,2)			Нет	*None*	

Id_Zak – унікальний ключ.

1.7. Заповнити таблицю «Zakazy»

Id_Zak	Id_Pokup	Id_Prod	Kol
1	1000	100	2
2	1003	300	3
3	1003	201	1
4	1002	201	4
5	1000	202	2
6	1000	301	1
7	1003	301	2

1.8. Створити таблицю «Pokupateli»

	Поле	Тип	Сравнение	Атрибуты	Null	По умолчанию
☐	Id_Pokup	int(11)			Нет	*None*
☐	Pokup_name	varchar(20)	cp1251_general_ci		Нет	*None*

Id_Pokup – первинний ключ

1.9. Заповнити таблицю «Pokupateli»

Id_Pokup	Pokup_name
1000	Петров
1001	Іванов
1002	Сидоров
1003	Артемов
1004	Івашов

2. Створення запитів.

В середовищі MySQL отримання необхідної інформації виконується за допомогою команди **SELECT**.

У загальному вигляді синтаксис оператора SELECT має наступний вигляд:

```
SELECT [ALL / DISTINCT] <список атрибутів> / *
FROM <список таблиць> [WHERE <умова вибірки>]
 [GROUP BY <список атрибутів>] [HAVING <умова>]
[ORDER BY <список атрибутів>]
[UNION <вираз з оператором SELECT>]
```

У квадратних дужках вказуються необов'язкові елементи.

Ключове слово **ALL** означає, що результатом будуть всі рядки, що задовольняють умові запиту, в тому числі і однакові рядки. **DISTINCT** означає, що в результуючий набір не включаються однакові рядки. Далі йде список атрибутів вихідної таблиці, які будуть включені в таблицю-результат. Символ * означає, що в таблицю-результат включаються всі атрибути вихідної таблиці.

Обов'язковою ключовим словом є слово F**ROM**, за ним слідують імена таблиць, до яких здійснюється запит.

В пропозиції з ключовим словом **WHERE** задаються умови вибірки рядків таблиці. У таблицю-результат включаються тільки ті рядки, для яких умова, вказане в пропозиції **WHERE**, приймає значення істина.

В пропозиції з ключовим словом **GROUP BY** задається список атрибутів угруповання (роз'яснення цього і наступного ключового слова буде представлено трохи пізніше).

У пропозиції **HAVING** задаються умови, що накладаються на кожну групу.

Ключове слово **ORDER BY** задає операцію упорядкування рядків таблиці-результату за вказаною списку атрибутів.

Коротше можна записати:

```
SELECT імена_столбцов FROM ім'я_таблиці [WHERE ... умови];
```

Частина оператора з умовами є необов'язковою (ми розглянемо її пізніше). Оператор **SELECT** без умов виводить всі дані із зазначених стовпців.

Виведемо Id покупця (стовпець Id_Pokup) і прізвища покупця (стовпець Pokup_name) всіх покупців з таблиці ***Pokupateli***.

```
SELECT `Id_Pokup`, `Pokup_name` from `Pokupateli`;
```

Дані представлені в тому порядку, в якому вони були введені. Більш того, останній рядок вказує число рядків в таблиці. Щоб вивести всю таблицю, можна скористатися спрощеною формою оператора **SELECT**:

```
SELECT * from `Pokupateli`;
```

Символ * в цьому виразі означає «ВСІ стовпці».

2.1. Знайти в таблиці «Producty» всю інформацію, що стосується продукту 'Яблука'

```
SELECT * FROM `Producty` WHERE` Prod` = 'Яблука'
```

2.2. Впорядкувати продукти за ціною. Сортування за будь-якою колонкою здійснюється за допомогою конструкції **ORDER BY**. У нашому випадку, наприклад, впорядкувати за спаданням ціни, а потім по зростанню:

```
SELECT * FROM `Producty` ORDER BY `cena` DESC
```

У цьому прикладі дані були розсортовані по спадаючій. Якщо нам треба впорядкувати дані по зростанню, треба замість ключового слова **DESC** застосувати **ASC**:

```
SELECT * FROM `Producty` ORDER BY` cena` ASC
```

Дані також можна впорядкувати за кількома стовпцями. Для цього треба назви стовпців вказати через кому. Це може стати в нагоді, наприклад, якщо в стовпці, по якому проводиться сортування, є кілька однакових значень.

```
SELECT * FROM `Zakazy` ORDER BY` Id_Pokup`, `Id_Prod` ASC
```

Спочатку дані упорядковано відповідно до стовпця Id_Pokup. Потім, якщо в першому стовпці є кілька однакових значень, виконується додаткове сортування за другим стовпцем (всередині групи з однаковими значеннями в першому стовпці).

2.3. Вибрати певну групу записів. Якщо потрібно, щоб під час пошуку видавалися не всі знайдені записи, а певна група, то потрібно використовувати параметр **LIMIT**. У цьому параметрі задається два значення:

```
LIMIT start, length
```

де:
start – вказує, з якої позиції потрібно видавати знайдені записи; length – кількість записів.

Наприклад, нам потрібно вибрати з таблиці «Producty» записи, починаючи з 0 по 5 (тобто щоб вивести 5 записів):

```
SELECT * FROM `Producty` LIMIT 0,5
```

2.4.Перевірити під час пошуку, чи відповідає дане символьне значення рядку з вказаною маскою, то з цією метою використовується предикат **LIKE**. Як маски використовуються всі дозволені символи (з урахуванням верхнього і нижнього регістрів), а також спеціальні символи:

% – заміщає будь-яку кількість символів (в тому числі і 0);

_ – заміщає тільки один символ.

Дозволено також використовувати конструкцію **NOT LIKE**.

Наприклад, нам потрібно вибрати з таблиці «Pokupateli» покупців, прізвище яких на букву «І»:

```
SELECT * FROM `Pokupateli` WHERE `Pokup_name` LIKE "I%"
```

Другий спосіб

```
SELECT * FROM `Pokupateli` WHERE (locate("Іва",`Pokup_name`)>0)
```

2.5. Поєднати поля та агрегатні функції. Речення **GROUP BY** дозволяє поєднувати поля и агрегатні функції в єдиному реченні **SELECT**. Наприклад, необхідно визначити вагу покупки кожного покупця.

```
SELECT `Id_Pokup`,SUM(`kol`) FROM `Zakazy` GROUP BY `Id_Pokup`
```

3. Самостійно виконати запит:

1. Вивести всі замовлення, зроблені покупцем, id якого дорівнює 1000.
2. Вивести кількість купленого товару з id = 201 («Цибуля»).
3. Вивести всі продукти, відсортовані за ціною.
4. Який продукт має мінімальну ціну.
5. Вивести список продуктів, ціна яких більше 4 і менше 9 грн.
6. Вивести список покупців, покупка яких має найбільшу вагу.
7. Вивести список покупців, в прізвищах яких не зрозуміла одна буква.

4. Питання для самоперевірки:

1. Як дізнатися число рядків в таблиці за допомогою оператора **SELECT**?
2. Що робить оператор **SELECT** без умов?
3. Які службові слова обов'язково присутні в операторі **SELECT**?
4. Який оператор задає імена для виведених стовпців?
5. Для чого використовується оператор **BETWEEN**?
6. Який оператор групує аналогічні дані?
7. Який оператор дозволяє накласти обмеження на групу?
8. Що робить оператор **UPDATE** без умов?
9. Навіщо потрібно ключове слово **DISTINCT**?

ЛАБОРАТОРНА РОБОТА №4
ВИБІР ДАНИХ З ТАБЛИЦЬ. ВКЛАДЕНІ ЗАПИТИ. ОБ'ЄДНАННЯ ТАБЛИЦЬ

Мета практичної роботи: підготувати та реалізувати серію запитів, пов'язаних з вибіркою інформації зі створених таблиць. Вивчити функції для роботи з датою і часом.

Виконання роботи.

Використовувати базу даних «Magazin» з лабораторної роботи 3.

1. Познайомитися з операціями: Join, Union, вкладеними запитами.

1.1 Поєднання (Join) призначені для забезпечення вибірки даних з декількох таблиць і включення цих даних в один результуючий набір. Існує чотири види поєднань: внутрішнє, зовнішнє, повне, перехресне.

Для об'єднання трьох і більше таблиць можна застосовувати послідовність поєднань.

Для з'єднання таблиць необхідно розділ **FROM** доповнити ключовими словами **JOIN**, яке визначає таблиці, які поєднуються, і метод поєднання, і **ON**, яке вказує загальні для таблиць поля.

```
SELECT <select list> FROM   <first table> [<alias>]
<JOIN TYPE>   <second table> [<alias>] [ON <join condition>][;]
```

*Внутрішнє поєднання **INNER JOIN***

При внутрішньому поєднанні порівнюються значення загальних полів двох таблиць. Конструкція **INNER JOIN** повертає тільки рядки, узгоджені по всіх полях, які позначені як використовувані для з'єднання. Записи, для яких немає пари в пов'язаної таблиці, в результат не включаються.

За замовчуванням застосовуються внутрішні з'єднання, ключове слово INNER є необов'язковим.

*Зовнішні з'єднання **LEFT OUTER JOIN, RIGHT OUTER JOIN***

При використанні конструкції **OUTER** існує можливість включити в результуючий набір рядки, які відповідають хоча б одному із заданих критеріїв. Такі сполуки застосовуються для отримання повного набору записів однієї з таблиць.

У з'єднаннях розрізняються боки – лівий і правий. Лівою вважається таблиця, зазначена в першу чергу, а правою – таблиця, зазначена після неї. Ключове слово OUTER необов'язкове.

*Повні з'єднання **FULL JOIN***

Якщо застосовується з'єднання з ключовим словом **FULL**, то в результати повинні бути включені всі рядки з таблиць, що знаходяться по обидва боки від ключового слова **JOIN**, без будь-яких винятків.

Перехресні з'єднання

При такому з'єднанні виводяться всі комбінації записів таблиць, при цьому не потрібно вказувати співпадаючих значень полів, тому умова **ON** опускається. Відбувається з'єднання кожного рядка таблиць, які перебувають з одного боку від ключового слова **JOIN**, з кожним рядком таблиць, що знаходяться з іншого боку від ключового слова **JOIN**. В результаті формується декартів добуток всіх рядків, заданих по обидва боки ключового слова **JOIN**.

1.***2 Об'єднання (Union).*** Іноді буває потрібно отримати два списки записів з таблиць у вигляді одного. Для цієї мети може бути використано ключове слово **UNION**, яке дозволяє об'єднати результуючі набори даних двох запитів в один набір даних.

Технічних обмежень на кількість запитів в операторі **UNION** не існує. Загальний синтаксис наступний.

```
< инструкция SELECT 1> UNION [ALL | DISTINCT]
< инструкция SELECT 2> UNION [ALL | DISTINCT]
```

UNION – показує, що результуючі набори будуть об'єднані в один результуючий набір. Дублікати рядків за замовчуванням видаляються.

ALL – об'єднуються і дублікати рядків з усіх результуючих наборів.

DISTINCT – з результуючого набору видаляються дублікати рядків. Стовпці, що містять значення **NULL**, вважаються дублікатами. За замовчуванням приймається **DISTINCT**.

Існує лише одне важливе правило, про яке слід пам'ятати при використанні оператора **UNION**: порядок, кількість і тип даних стовпців повинні бути ідентичні у всіх запитах.

Типи даних не обов'язково повинні бути ідентичні, але вони повинні бути сумісні. Наприклад, типи **CHAR** і **VARCHAR** є сумісними. За

замовчуванням результуючий набір використовує розмір найбільшого з сумісних типів.

Імена полів підсумкового набору беруться тільки з першого запиту, тому створення псевдонімів полів виконується в ньому.

Для отримання відсортованого набору даних розділ **ORDER BY** вказується після останнього оператора **SELECT**. На відміну від сполуки, записи в підсумковий набір додаються один за одним.

1.3 Вкладені запити

У SQL передбачена можливість об'єднувати запити в один шляхом перетворення одного з них в підзапит (вкладений запит). В одному запиті може бути кілька підзапитів, синтаксис у такого запиту наступний:

```
SELECT ім’я _стовпця FROM ім’я_таблиці WHERE частина умови IN
(SELECT ім’я _стовпця FROM ім’я_таблиці WHERE частина умови IN
(SELECT ім’я _стовпця FROM ім’я_таблиці WHERE умова))
```

Підзапити можуть вибирати тільки один стовпець, значення якого вони будуть повертати зовнішнім запитом. Спроба вибрати кілька стовпців призведе до помилки. Варто відзначити, що не рекомендується створювати запити зі ступенем вкладення більше трьох. Це призводить до збільшення часу виконання і до складності сприйняття коду.

Наведений синтаксис вкладених запитів, скоріше найбільш уживаний, але зовсім не єдиний. Ми можемо використовувати будь-які оператори, які використовуються з ключовим словом **WHERE**.

2. Завдання для самостійної роботи.

Група складності 1.

1. Знайти на яку суму купив продукти покупець 'Сидоров'.
2. Вивести список покупців і вартості їх покупок в порядку зростання вартості.
3. Які покупці купили продуктів на суму 15 грн і більше.
4. Які покупці купили продукти з групи «Овочі» з урахуванням номера замовлення.
5. Вивести список покупців і куплених ними продуктів.
6. Вивести повний список усіх замовлень.
7. Вивести список продуктів, які найбільше купували.

Група складності 2.

1. Показати всі продукти, які купив покупець 'Петров', їх ціну, вагу і вартість.

2. Вивести список покупців, які купували продукти (продукти розподілити по групах).

3. Вивести список покупців, які не зробили жодної покупки.

4. Вивести кількість куплених продуктів кожної групи (вказати назву групи продуктів, кількість покупок продуктів цієї групи і їх сумарну вага).

5. Вивести список покупців, які купили продукти всіх груп.

3. Питання для самоперевірки:

1. Перерахуйте чотири види з'єднань.

2. При якому з'єднанні записи, для яких немає пари в пов'язаної таблиці, в результат не включаються?

3. При якому з'єднанні умова **ON** опускається?

4. Які сполуки застосовуються для отримання повного набору записів однієї з таблиць?

5. За допомогою якого з'єднання можна отримати декартів добуток таблиць?

6. Чим відрізняється порядок записів в підсумковому наборі, отриманому за допомогою з'єднання **Join** і об'єднання **Union**?

7. З якого запиту беруться імена полів підсумкового набору при об'єднанні (**Union**)?

ЛАБОРАТОРНАЯЛАБОРАТОРНА РОБОТА №5 СТВОРЕННЯ ТА ЗМІНА БАЗИ ДАНИХ ТА ТАБЛИЦЬ. ТРИГЕРИ

Мета практичної роботи: Вивчити призначення тригерів, умов їх активації, синтаксису та семантики команд для їх створення, модифікації, перейменування, програмування та видалення, а також придбання навичок їх проєктування.

Виконання роботи.

1. Познайомитися з тригерами. Тригери – це іменовані об'єкти бази даних, які пов'язані з таблицею, і активізуються при настанні певних подій, пов'язаних з цією самою таблицею. Подій для таблиць лише кілька: **INSERT**, **DELETE, UPDATE.**

Тригери допомагають, коли необхідно, наприклад, виконати перевірки значень, які будуть вставлені в таблицю, або виконувати обчислення на значеннях, що включаються в модифікації.

Створення тригеру:

```
CREATE
  [DEFINER = {user | CURRENT_USER}]
  TRIGGER trigger_name trigger_time trigger_event
  ON tbl_name FOR EACH ROW trigger_stmt
```

CREATE TRIGGER. Ця інструкція створює новий тригер. Тригер стає пов'язаним з таблицею з ім'ям tbl_name (ім'я постійної таблиці). Не можна пов'язувати тригер з view або таблицею **TEMPORARY**.

Коли тригер активізований, пропозиція **DEFINER** визначає привілеї. У тригері за замовчуванням використовується

DEFINER = CURRENT_USER,

тобто права користувача системи, а можна встановити права й іншого користувача.

trigger_time задає час дії. Тригери можуть активізуватися як до (**BEFORE**) так і після (**AFTER**) зміни таблиці.

trigger_event вказує вид інструкції, яка активізує тригер. Тут trigger_event може бути одним з наступних варіантів:

INSERT: всякий раз, коли новий рядок вставлено в таблицю. Наприклад, через команди **INSERT, LOAD DATA** або **REPLACE**;

UPDATE: всякий раз, коли рядок змінюється. Наприклад, через інструкцію **UPDATE**;

DELETE: всякий раз, коли рядок видалено з таблиці. Наприклад, через інструкції **DELETE** і **REPLACE**. Однак, інструкція **DROP TABLE** щодо таблиці не активізує тригер, тому що не використовує **DELETE**.

Виходить, що тригер може бути одному з 6 станів:
INSERT (BEFORE | AFTER)
UPDATE (BEFORE | AFTER)
DELETE (BEFORE | AFTER)

Не може бути двох тригерів для даної таблиці, які мають ті ж самі час дії та подію. Наприклад, не можна мати два тригера **BEFORE UPDATE** для таблиці. Але Ви можете мати **BEFORE UPDATE** і **BEFORE INSERT** або **BEFORE UPDATE** і **AFTER UPDATE**.

trigger_stmt задає інструкцію, яка буде виконана, коли тригер активізується. Якщо Ви хочете виконувати багато інструкцій, використовуйте операційну конструкцію **BEGIN ... END**. Це також дає можливість Вам використовувати ті ж самі інструкції, які є допустимими всередині збережених підпрограм.

2. Виконання практичної частини

Необхідно при додаванні запису в таблицю user, пароль перетворювати в хеш md5(), а ім'я та по батькові перетворювати в ініціали.

Створюємо БД з ім'ям `user`.

Створюємо таблицю з ім'ям `user` і заголовком таблиці:

Атрибути	Тип	Порівняння	Null	За замовчуванням
fam	text	*cp1251_general_ci*	Нет	*Нет*
name	text	*cp1251_general_ci*	Нет	*Нет*
otch	text	*cp1251_general_ci*	Нет	*Нет*
pass	text	*cp1251_general_ci*	Нет	*Нет*
login	text	*cp1251_general_ci*		

Вирішення

1. У таблицу `user` вставлємо запис:

```
INSERT INTO `user` SET `fam`='Ногаченко', `name`='Максим', `otch` = 'Валерьевич', `pass` = 'pass1', `login` = 'maxnag';
```

2. Створюємо тригер с ііім'ям `user_insert`

```
DELIMITER |
CREATE TRIGGER `user_insert` BEFORE INSERT ON `user`.`user`
    FOR EACH ROW
      BEGIN
        SET NEW.name = LEFT(NEW.name,1);
  SET NEW.otch = LEFT(NEW.otch,1);
  SET NEW.pass = md5(NEW.pass);
END;
```

Що таке **NEW** в тілі тригера:

NEW — для доступу до нових записів

OLD - для доступу до старих записів

Наприклад , якщо оновлюється прізвище, то нове значення доступно через NEW.fam, а старе OLD.fam

3. У таблицю `user` вставляємо ще запис:

```
INSERT INTO `user` SET `fam`='Иванов', `name`='Иван', `otch` = 'Иванович', `pass` = 'passw', `login` = 'ivan';
```

4. В таблиці `user` зараз 2 запису

fam	name	otch	pass	login
Ногаченко	Максим	Валерійович	pass1	maxnag
Иванов	И	И	d79096188b670c2f81b7001f73801117	ivan

Як видно лише кількома рядками можна прибрати цілі методи, які ми використовували при реєстрації нового користувача. Тепер ще треба

створити тригер на UPDATE таблиці, з таким же тілом, щоб користувач не зміг записати повне ім'я, по батькові та пароль не в МД5 ();

5. Створюємо тригер на BEFORE UPDATEDELIMITER |

```
CREATE TRIGGER `user_update` BEFORE UPDATE ON `user`.`user`
    FOR EACH ROW
      BEGIN
        SET NEW.name = LEFT(NEW.name,1);
  SET NEW.otch = LEFT(NEW.otch,1);
  SET NEW.pass = md5(NEW.pass);
END;
```

6. Оновлюємо запис:

```
UPDATE `user` SET `fam`='Петров', `name`='Петр', `otch` = 'Петрович', `pass` = 'парол', `login` = 'petr' WHERE `fam`='Ногаченко';
```

7. Підсумок

fam	name	otch	pass	login
Петров	П	П	defe864f35f11a02625f57b1b6cc8e52	petr
Иванов	И	И	d79096188b670c2f81b7001f73801117	ivan

Зміна тригера - команди на зміну тригера просто немає.

8. Видалення тригера

Для видалення тригера використовується, як зазвичай оператор DROP, приклад:

```
DROP TRIGGER [IF EXISTS] [schema_name.] Trigger_name
```

де:

schema_name - назва БД,

trigger_name - назва тригера

9. Список створених тригерів

Показати тригер можна за допомогою команди

```
SHOW TRIGGERS [{FROM | IN} db_name] [LIKE 'pattern' | WHERE expr]
```

Це повна частина команди, в основному користуються командами:

```
SHOW TRIGGERS [FROM db_name] [LIKE 'pattern']
або
SHOW TRIGGERS
```

Як призупинити виконання SQL після тригера? Тобто, за певних умов, описаних в тригері треба зупинити виконання SQL, який і викликав цей тригер.

3. Завдання для самостійної роботи:

Для полегшення роботи з базою даних ***«Magazin»*** створити 6 тригерів, які забезпечують роботу БД в ситуаціях:

INSERT (BEFORE | AFTER)
UPDATE (BEFORE | AFTER)
DELETE (BEFORE | AFTER)

4. Питання для самоперевірки

1. Що таке зовнішній ключ, його призначення?

2.Чи можна забезпечити цілісність даних без зовнішніх ключів?

3. Що таке тригер? Принцип його роботи.

4. Які існують режими зв'язку, створені за допомогою зовнішнього ключа?

5. У чому різниця між породженою таблицею та таблицею, що породжує?

6. Чи може бути використаний механізм тригерів для перевірки коректності введення. Наприклад: рядок, що містить e-mail, повинен обов'язково містити символ '@'?

7. Що означає поняття подія, що тригується?

8. За допомогою якої синтаксичної конструкції задається одноразове виконання тригера?

9. За допомогою якої синтаксичної конструкції можна звернутися до рядків, що видаляються?

10. За допомогою якої синтаксичної конструкції можна звернутися до рядків, що додаються?

11. За допомогою якої синтаксичної конструкції можна звернутися до рядків, що модифікуються?

ЛАБОРАТОРНА РОБОТА №6
ПРОЦЕДУРИ ТА ФУНКЦІЇ, ЩО ЗБЕРІГАЮТЬСЯ

Мета практичної роботи: Вивчення синтаксису та семантики збережених функцій і процедур: способів їх ідентифікації, методів завдання і специфікації параметрів і значень, кодування тіла і викликів функцій і процедур, що зберігаються, застосування команд для створення, зміни та видалення системних і призначених для користувача як скалярних, так і системних, призначених для користувача, тимчасових (локальних або глобальних) і розширених збережених процедур, а також придбання навичок програмування, налагодження, тестування і включення в групу.

Виконання роботи.

1. Познайомитися з поняттям процедур та функцій, що зберігаються.

Починаючи з версії MySQL 5.0 було додано збережені функції і процедури – набір SQL команд, що представляють собою програмну логіку, яка зберігається в БД, звідки її надалі можна буде викликати.

По суті, збережені процедури представляють собою набір інструкцій, які виконуються як єдине ціле. Тим самим збережені процедури дозволяють спростити комплексні операції і винести їх в єдиний об'єкт. Щоб змінити процес покупки товару, достатньо буде змінити код процедури. Тобто процедура також спрощує управління кодом.

Також збережені процедури дозволяють обмежити доступ до даних в таблицях і тим самим зменшити ймовірність навмисних або неусвідомлених небажаних дій щодо цих даних.

І ще один важливий аспект – продуктивність. Збережені процедури зазвичай виконуються швидше, ніж звичайні SQL-інструкції. Все тому, що код процедури компілюється один раз при першому її запуску, а потім зберігається в компільованій формі.

Збережені підпрограми зберігаються всередині таблиці proc бази MySQL.

Для роботи з збереженими підпрограммами необхідні такі привілеї для користувача: **CREATE ROUTINE, ALTER ROUTINE, EXECUTE**.

Табл.1 – Набір команд, що використовуються разом з підпрограмами, що зберігаються

Назва	Опис
CREATE PROCEDURE	створення процедури
CREATE FUNCTION	створення функції
ALTER PROCEDURE	зміна процедури
ALTER FUNCTION	зміна функції
DROP PROCEDURE	видалення процедури
DROP FUNCTION	вилучити функцію
SHOW CREATE PROCEDURE proc_name	показати текст процедури proc_name
SHOW CREATE FUNCTION func_name	показати текст функції func_name
SHOW PROCEDURE STATUS LIKE ‘proc_name’	показати характеристики процедури proc_name
SHOW FUNCTION STATUS LIKE ‘func_name’	показати характеристики функції func_name
CALL proc_name()	викликати процедуру proc_name
DECLARE	визначення локальних змінних
SET	зміна значень локальних і глобальних змінних
SELECT … INTO	збереження значення зазначеного стовпця в змінну
IF	умовний оператор if-then-else-end
CASE … WHEN	оператор вибору
LOOP,REPEAT,WHILE	цикли
RETURNS	повернення значення з функції

1.1 Створення процедури, що зберігається

```
DELIMITER //
CREATE PROCEDURE `p2` ()
LANGUAGE SQL
DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A procedure'
BEGIN
SELECT 'Hello World !';
END//
```

Перша частина коду створює збережену процедуру. Наступна – містить необов'язкові параметри. Потім йде назва і, нарешті, тіло самої процедури.

Назви збережених процедур чутливі до регістру. Також не можна створювати кілька процедур з однаковою назвою. Усередині процедури не може бути виразів, що змінюють саму базу даних.

Характеристики збереженої процедури:

Language – з метою забезпечення переносимості, за замовчуванням вказано SQL;

Deterministic – якщо процедура весь час повертає один і той же результат, і приймає одні і ті ж входять параметри. Це для реплікації і процесу реєстрації. Значення за замовчуванням – **NOT DETERMINISTIC**;

SQL Security - під час виклику йде перевірка прав користувача;

INVOKER – це користувач, що викликає збережену процедуру;

DEFINER – це «творець» процедури. Значення за замовчуванням – **DEFINER**;

Comment: з метою документування, значення за замовчуванням – пуста строка.

1.2 Виклик процедури, що зберігається. Щоб викликати збережену процедуру, необхідно надрукувати ключове слово **CALL**, а потім назву процедури, а в дужках вказати параметри (змінні або значення). Дужки обов'язкові.

```
CALL procedure(10, 'string parameter', @parameter var);
```

1.3 Змінення процедури**.** В **MySQL** є вираз **ALTER PROCEDURE** для зміни процедур, але воно підходить для зміни лише деяких характеристик. Якщо вам потрібно змінити параметри або тіло процедури, вам слід видалити і створити її заново.

1.4 Видалення збереженої процедури

```
DROP PROCEDURE IF EXISTS p2;
```

Це проста команда. Вираз **IF EXISTS** вказує помилку в разі, якщо такої процедури не існує.

1.5 Визначити параметри процедури

Як можна передавати в збережену процедуру параметри.

CREATE PROCEDURE proc1 () – порожній список параметрів.

CREATE PROCEDURE proc1 (IN varname DATA-TYPE) –

один вхідний параметр. Слово IN необов'язково, тому що параметри за замовчуванням - **IN** (вхідні).

CREATE PROCEDURE proc1 (OUT varname DATA-TYPE**)** - один параметр, що повертається.

CREATE PROCEDURE proc1 (INOUT varname DATA-TYPE) - один параметр, що одночасно входить і повертається.

Можна задавати кілька параметрів різних типів.

Приклад процедури з параметром IN:

```
   DELIMITER //

CREATE PROCEDURE `proc_IN` (IN var1 INT)
BEGIN
    SELECT var1 + 2 AS result;
END//
```

Приклад процедури з параметром OUT:

```
DELIMITER //

CREATE PROCEDURE `proc_OUT` (OUT var1 VARCHAR(100))
BEGIN
    SET var1 = 'This is a test';
END //
```

Приклад процедури з параметром INOUT:

```
DELIMITER //

CREATE PROCEDURE `proc_INOUT` (OUT var1 INT)
BEGIN
    SET var1 = var1 * 2;
END //
```

Змінні

Далі необхідно навчитися створювати змінні і зберігати їх усередині процедур. Ви повинні оголошувати їх явно на початку блоку BEGIN / END, разом з їх типами даних. Як тільки ви оголосили змінну, ви можете використовувати її там же, де змінні сесії, літерали або імена колонок.

Синтаксис оголошення змінної виглядає так:

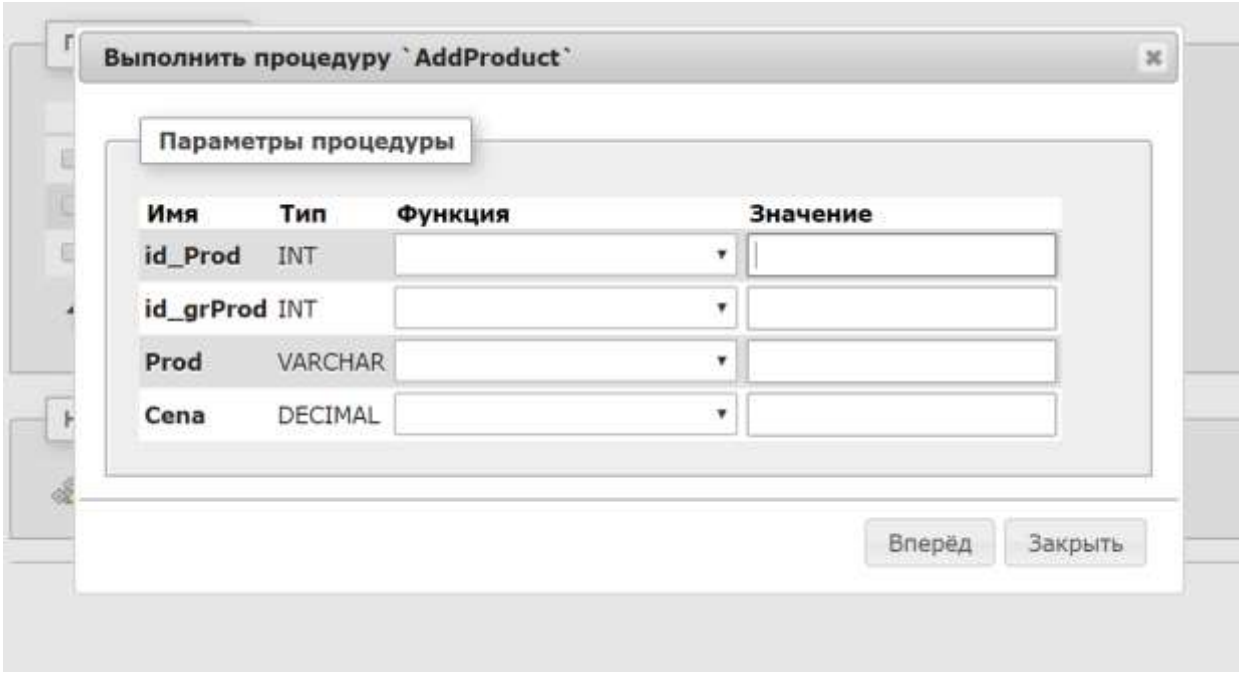
```
DECLARE varname DATA-TYPE DEFAULT defaultvalue;
```

Робота зі змінними

Як тільки ви оголосили змінну, ви можете задати їй значення за допомогою команд SET або SELECT

2. Створення процедур за допомогою phpMyAdmin

Також ви можете управляти процедурами (додавати, видаляти, змінювати), а також виконувати їх за допомогою phpmyadmin:



Приклад

Розглянемо деякі приклади процедур для бази даних продукти magazin яку ми створили раніше.

```
DELIMITER //
CREATE PROCEDURE `ShowProducty`()
LANGUAGE SQL
DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'Выводит список продуктов'
BEGIN
    SELECT `id_Prod` AS Номер, `Prod` AS Продукт, `Cena` AS Цена
    FROM `Producty` ORDER BY `id_Prod` ASC;
END//
```

Викликаймо процедуру:

CALL ShowProducty()

Номер	Продукт	Цена
100	Яблоки	6.50
101	Груши	9.50
200	Картофель	5.20
201	Лук	4.50
202	Морковь	4.00
300	Гречка	11.50
301	Рис	8.70

```
DELIMITER //
CREATE PROCEDURE `AddProduct` (IN id_Prod INT, IN id_grProd INT,
      IN Prod varchar(20), IN Cena decimal(10,2))
LANGUAGE SQL
DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'Добавляет продукт'
BEGIN
   INSERT INTO `producty`(`id_Prod`, `id_grProd`, `Prod`, `Cena`)
    VALUES (id_Prod, id_grProd, Prod, Cena);
END//
```

CALL AddProduct(302, 3, 'Булгур', 20)

	id_Prod	id_grProd	Prod	Cena
Изменить Копировать Удалить	100	1	Яблоки	6.50
Изменить Копировать Удалить	200	2	Картофель	5.20
Изменить Копировать Удалить	101	1	Груши	9.50
Изменить Копировать Удалить	300	3	Гречка	11.50
Изменить Копировать Удалить	202	2	Морковь	4.00
Изменить Копировать Удалить	301	3	Рис	8.70
Изменить Копировать Удалить	201	2	Лук	4.50
Изменить Копировать Удалить	302	3	Булгур	20.00

```
CREATE PROCEDURE `minPrice` (OUT minPrice VARCHAR(100))
LANGUAGE SQL
DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'Викликає мінімальну цену'
```

```
BEGIN
  DECLARE a INT;
   DECLARE str VARCHAR(50);
  SET str = 'Мінімальна цена: ';
   SET a = (SELECT MIN(`Cena`) FROM `Producty`);
   SELECT CONCAT(str, a);
END
```

CONCAT(str, a)
Минимальная цена: 4

3. Створення функції

1. Створимо функцію cena_diff для підрахунку різниці між ціною продукту і середньою ціною всіх продуктів. Як прийнятого параметра вкажемо змінну a, тип даних DECIMAL, тому що ціна продукту має такий тип даних (рис.1).

```
CREATE FUNCTION cena_diff(a DECIMAL)
```

Рисунок 1- Створення функції

Функція буде повертати тип даних float (рис.2).

```
RETURNS FLOAT
```

Рисунок 2- Оголошення типу вихідних даних

1. Напишемо оператор BEGIN для початку роботи функції. За допомогою команди DECLARE оголосимо змінну avg_cena для зберігання середньої ціни продуктів, початкове значення задамо рівне нулю (рис.4).

```
BEGIN
DECLARE avg_cena FLOAT DEFAULT 0;
```

Рисунок 3 – Ініціалізація змінних

За допомогою оператора SELECT виберемо середнє значення цін, використовуючи функцію AVG і запишемо це значення в змінну avg_cena за допомогою оператора INTO (рис.4).

```
SELECT AVG(`Cena`) INTO avg_cena FROM `producty`;
```

Рисунок 4 – Выбор средней цены продуктов

2. Зазначимо, що функція буде повертати різницю між ціною продукту і середнім значенням цін по модулю (рис.5).

```
RETURN (ABS(a - avg_cena));
```

Рисунок 5 – Вказівка того, що буде повертати функція

3. Наприкінці функції напишемо оператор END. Для коректної роботи функції на початку і в кінці додамо роздільник DELIMITER, що вказує на символ-роздільник рядка з командою..

4. Для виклику функції використовуємо оператор SELECT і в якості вхідного параметра в функцію вкажемо поле 'Cena' (рис.6).

```
SELECT Prod.cena_diff(Cena) FROM `producty`
```

Рисунок 6 - Виклик функції

4. Завдання для самостійної роботи:

Для полегшення роботи з базою даних ***«Magazin»*** створити 3 процедури та 3 функції, які забезпечують роботу БД.

5. Питання для самоперевірки

1. Де зберігається збережена процедура?
2. Може збережена процедура викликати іншу збережену процедуру?
3. Яка SQL команда викликає збережену процедуру?
4. Какие існують типи параметрів процедури?
5. Чим різняться процедури з параметрами IN, OUT та INOUT?
6. Який синтаксис змінної в MySQL?
7. У чому відмінність процедур і функцій, що зберігаються?

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

Основні:

1 Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.

2 Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.

3. Швецов В.И. Базы данных. - Н.Новгород: Изд-во ННГУ, 2008. –276 с.

http://window.edu.ru/window_catalog/files/r61460/shvetsov-lectures.pdf.

4. Дейт К. *Дж.* Введение в системы баз данных = Introduction to Database Systems. — 8-е изд. — М.: «Вильямс», 2006. — 1328 с.

Додаткові:

1. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.

2. Кириллов В.В. Основы проектирования реляционных баз данных. Учебное пособие. – СПб: ГИТМО.

http://www.citforum.ru/database/dbguide/index.shtml.

3. Компания MySQL AB. MySQL. Справочник по языку: Пер. с англ. - М.: Издательский дом "Вильяме", 2005. - 432 с.

4. Черноусова А.М. Создание и использование баз данных: учебное пособие / А.М. Черноусова. – Оренбург: ГОУ ОГУ, 2009. – 244 с.

Навчальне видання

СІМОНОВА Ольга Геннадіївна
ОХОТСЬКА Олена Вадимівна
ШЕЛІХОВА Інеса Борисівна

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт та самостійної
роботи студентів спеціальності 122 «Комп'ютерні науки»
з курсу «Інтернет технології комп'ютерної графіки та анімації»

В авторській редакції

План 2021 р., поз. 110

Підп. до друку 17.09.2021 Формат 60x84 1/16. Папір офсетний.
Друк — цифровий. Гарнітура Гарнітура Times New Roman. Ум. друк. арк.1,8
Наклад 50 прим. Зам. № 21/05-02 Ціна договірна.

Видавництво ТОВ фірма «НТМТ»
Свідоцтво внесення до державну реєстру видавців ДК № 1748 від 15.04.2004 р. 61103, м. Харків, вул. Дерев'янка, 16
