Лабораторна робота №7

Тема: Робота з базами даних в Java з допомогою.

Мета: Вивчити особливості роботи з базами даних у Java.

Завдання: Здобути навички створення та відлагодження проектів обробки даних з використанням технології JDBC мовою Java.

(max: 10 балів)

Теоретичні відомості:

7.1. Загальні відомості.

Для зберігання даних можна використовувати різні бази даних – Oracle, MSSQL Server, MySQL, Postgres і т.д. Всі ці системи управління базами даних мають свої особливості. Головне, що їх об'єднує це взаємодія зі сховищем даних за допомогою команд SQL. Щоб визначити єдиний механізм взаємодії з цими СУБД в Java починаючи ще з 1996 року було введено спеціальний прикладний інтерфейс API, який названо **JDBC**.

Відповідно, якщо потрібно в додатку на мові Java взаємодіяти з базою даних, то необхідно використовувати функціональні можливості JDBC. Даний API входить до складу Java, зокрема, для роботи з JDBC в програмі Java досить підключити пакет java.sql.

Для роботи з певною СУБД необхідний спеціальний драйвер. Кожен розробник СУБД, зазвичай, надає свій драйвер для роботи з JDBC. Наприклад, якщо потрібно працювати з MySQL, то потрібний спеціальний драйвер для роботи саме з MySQL. Як правило, більшість драйверів вільно доступні на сайтах відповідних СУБД. Зазвичай, вони представляють **JAR-архіви**. Перевага JDBC полягає саме в тому, що ми абстрагуємося від будови конкретної бази даних та використовуємо уніфікований інтерфейс, який єдиний для всіх.

Для взаємодії з базою даних через JDBC використовуються **запити SQL**. У той же час можливості SQL для роботи з кожною конкретною СУБД можуть відрізнятися. Наприклад, в MS SQL Server це T-SQL, в Oracle - це PL/SQL. Але, в загальному, ці різновиди мови SQL не суттєво відрізняються.

7.2. Встановлення драйвера та підключення до бази даних MySQL.

Однією з найбільш популярних СУБД є MySQL. Для роботи з MySQL в Java потрібно встановити офіційний драйвер MySQL **Connector/J**, завантажити який можна зі веб-сторінки *https://dev.mysql.com/downloads/connector/j/*. Після цього потрібно розпакувати завантажений драйвер та розмістити його в папку з програмою. Перевірити взаємодію з MySQL через завантажений драйвер можна виклонуючи наступний код програми:

```
public class Program {
public static void main (String [] args) {
    //java -classpath c:\Java\mysql-connector-java-8.0.11.jar; c:\Java
    Program
    try {
        Class.forName("com.mysql.cj.jdbc.Driver").
        GetDeclaredConstructor().NewInstance();
         System.out.println("З'єднання успішне!");
      }
      catch (Exception ex) {
         System.out.println("З'єднання немає ...");
         System.out.println(ex);
      }
  }
}
```

Для завантаження драйвера призначений рядок:

```
Class.forName("com.mysql.cj.jdbc.Driver").GetDeclaredConstructor().NewInstance();
```

Метод *Class.forName()* в якості параметра приймає рядок, який представляє повний шлях до класу драйвера з урахуванням всіх пакетів. У випадку MySQL це шлях *"com.mysql.cj.jdbc.Driver"*. Таким чином, метод ***Class.forName*** завантажує клас драйвера, який буде використовуватися. Далі викликається **метод *getDeclaredConstructor()***, який повертає конструктор даного класу. І в кінці викликається **метод *newInstance()***, який створює за допомогою конструктора об'єкт даного класу. Після чого можна взаємодіяти з сервером MySQL.

Для підключення до бази даних спочатку потрібно на сервері MySQL **створити порожню базу даних**, яку назвемо *shop* і з якою буде здійснюватись взаємодія. Для створення бази даних застосовується вираз SQL:

```
CREATE DATABASE shop;
```

Його можна виконати або з консольного клієнта MySQL Command Line Client, або з графічного клієнта MySQL Workbench, які встановлюються разом з сервером MySQL. Для **підключення до бази даних** необхідно створити об'єкт *java.sql.Connection*. Для його створення використовується метод:

```
Connection DriverManager.getConnection(url, username, password)
```

Метод *DriverManager.getConnection* приймає параметри: адресу ресурсу даних, логін і пароль. В якості логіна і пароля передаються логін і пароль від сервера MySQL. Адреса локальної бази даних MySQL вказується у наступному форматі: *jdbc:mysql://localhost/назва_БД.*

Приклад створення підключення до створеної вище локальної бази даних store:

```
Connection conn = DriverManager.getConnection("jdbc: mysql: // localhost / shop", "root", "password");
```

Після завершення роботи з підключенням його слід закрити за допомогою методу close():

```
Connection conn = DriverManager.getConnection("jdbc: mysql: // localhost / shop", "root", "password");
// ... робота з БД
conn.close();
```

Або можна використовувати конструкцію *try*:

```
try (Connection conn = DriverManager.getConnection("jdbc: mysql: // localhost / shop", "root", "password")) {
   // ... робота з БД
}
```

7.3. Виконання SQL-команд.

Для взаємодії з базою даних додаток відправляє серверу MySQL **команди на мові SQL**. Щоб виконати команду, спочатку необхідно створити **об'єкт *Statement***. Для його створення у об'єкта ***Connection*** викликається метод ***createStatement()*:**

```
Statement stat = conn.createStatement();
```

Для виконання команд SQL в класі Statement визначено три методи:

- *executeUpdate()* виконує такі команди, як *INSERT*, *UPDATE*, *DELETE*, *CREATE TABLE*, *DROP TABLE*, результатом є повернення кількості рядків, змінених операцією (наприклад, кількість доданих, змінених або видалених рядків), або 0, якщо жоден рядок не змінений операцією чи команда не змінює вміст таблиці (наприклад, команда створення нової таблиці);
- *executeQuery()* виконує команду *SELECT*, результатом є повернення об'єкта *ResultSet*, який містить результати запиту.
- *execute()* виконує будь-які команди і повертає значення boolean: *true* - якщо команда повертає набір рядків (*SELECT*), інакше повертається *false*.

Розглянемо **метод *executeUpdate()***, як параметр в нього передається команда SQL:

```
int executeUpdate("SQL-команда")
```

Раніше була створена база даних *store*, яка залишається порожньою, в ній немає таблиць і відповідно даних. Створимо таблицю і додамо в неї початкові дані:

```
import java.sql. *;
 public class Program {
   public static void main (String[] args) {
      try {
         String url = "jdbc: mysql: // localhost / shop? ServerTimezone =
         Europe / Moscow & useSSL = false";
         String user = "root";
         String pass = "password";
         Class.forName("com.mysql.cj.jdbc.Driver").
         GetDeclaredConstructor().NewInstance();
         // команда створення таблиці
         String sqlCommand = "CREATE TABLE products(Id INT
         PRIMARY KEY AUTO_INCREMENT,
         ProductName VARCHAR(20), Price INT)";
```

```
            try (Connection conn = DriverManager.getConnection(url,
              user, pass)) {
              Statement stat = conn.createStatement();
              // створення таблиці
              stat.executeUpdate(sqlCommand);
              System.out.println("База даних створена!");
            }
          }
          catch (Exception ex) {
            System.out.println("З'єднання немає ...");
            System.out.println(ex);
          }
        }
      }
```

В даному випадку виконується команда *CREATE TABLE products (Id INT PRIMARY KEY AUTO_INCREMENT, ProductName VARCHAR (20), Price INT)*, яка створює таблицю *Products* з трьома стовпцями: *Id* – ідентифікатор рядка, *ProductName* – назва товару (рядок) і *Price* – ціна товару (число).

При цьому якщо необхідно виконати одразу кілька команд, то необов'язково створювати новий об'єкт *Statement*:

```
Statement stat = conn.createStatement();
stat.executeUpdate("SQL-команда1");
stat.executeUpdate("SQL-команда2");
stat.executeUpdate("SQL-команда3");
```

Для додавання, редагування та видалення даних можна використовувати **метод *executeUpdate***. За допомогою результату методу можна проконтролювати, скільки рядків було додано, змінено або видалено.

Візьмемо таблицю ***Products*:**

```
CREATE TABLE Products(
  Id INT PRIMARY KEY AUTO_INCREMENT,
  ProductName VARCHAR (20), Price INT)
```

І додамо в цю таблицю декілька записів:

```
import java.sql. *;
public class Program {
  public static void main (String[] args) {
    try {
      String url = "jdbc: mysql: // localhost / shop? ServerTimezone =
      Europe / Moscow & useSSL = false";
      String user = "root";
      String pass = "password";
      Class.forName("com.mysql.cj.jdbc.Driver").
      GetDeclaredConstructor().NewInstance();
      try(Connection conn = DriverManager.getConnection(url,
        user, pass)){
        Statement stat = conn.createStatement();
        int rows = stat.executeUpdate("INSERT Products
        (ProductName, Price) VALUES ( 'Nokia 9', 6200),
        ('Galaxy S9', 4300), ('iPhone X', 13100)");
        System.out.printf("Додані %d рядків", rows);
      }
    }
    catch(Exception ex) {
      System.out.println("З'єднання немає ...");
      System.out.println(ex);
    }
  }
}
```

Для додавання даних в БД застосовується команда ***INSERT***. В даному випадку в таблицю ***Products*** додаються три записи. Після виконання програми на екрані консолі можна побачити кількість доданих об'єктів.

Змінимо рядки в таблиці, наприклад, зменшимо ціну товару на 5000 одиниць. Для зміни застосовується команда ***UPDATE***. При створеному зв'язку та об'єкті ***stat*** класу ***Statement*** рядок зміни даних може виглядати наступним чином:

```
int rows = stat.executeUpdate("UPDATE Products SET Price = Price - 5000");
```

Для видалення записів з бази даних можна використовувати команду ***DELETE***, наприклад:

```
int rows = stat.executeUpdate("DELETE FROM Products WHERE Id = 3");
```

Для вибірки даних за допомогою команди ***SELECT*** застосовується **метод *executeQuery***:

```
executeQuery("SQL-команда")
```

Метод повертає об'єкт ***ResultSet***, який містить всі отримані дані. Як ці дані отримати? В об'єкті ***ResultSet*** ітератор встановлюється у позицію перед першим рядком, щоб переміститися до першого рядка (і до всіх наступних) необхідно викликати метод ***next()***. Поки у наборі ***ResultSet*** є доступні рядки, метод ***next*** буде повертати ***true***. Типове переміщення по набору рядків:

```
ResultSet resSet = stat.executeQuery("SELECT * FROM Products");
while(resSet.next()) {
   // отримання вмісту рядків
}
```

Тобто поки в ***resSet*** є доступні рядки, буде виконуватися цикл ***while***, який буде переходити до наступного рядка в наборі.

Після переходу до рядка можна отримати його вмістиме. Для цього у ***ResultSet*** визначено ряд методів. Наведемо деякі з них:

- *getBoolean()* повертає значення типу boolean;
- *getDate()* повертає значення типу Date;
- *getFloat()* повертає значення типу float;
- *getLong()* повертає значення типу long;
- *getDouble()* повертає значення типу double;
- *getInt()* повертає значення типу int;
- *getNString()* повертає значення типу String;
- *getString()* повертає значення типу String.

Залежно від того, дані якого типу зберігаються в тому чи іншому стовпці, можна використовувати той чи інший метод. Кожен з цих методів має дві версії:

- *int getInt (int columnIndex);*
- *int getInt (String columnLabel)*.

Перша версія отримує дані з стовпця з номером ***columnIndex***. Друга версія отримує дані з стовпця з назвою ***columnLabel***. Наприклад для таблиці:

```
CREATE TABLE products (
   Id INT PRIMARY KEY AUTO_INCREMENT,
   ProductName VARCHAR (20),
   Price INT
)
```

Отримаємо з неї дані:

```
import java.sql. *;
public class Program {
   public static void main (String [] args) {
      try {
         String url = "jdbc: mysql: // localhost / shop? ServerTimezone =
         Europe / Moscow & useSSL = false";
         String user = "root";
         String pass = "password";
         Class.forName("com.mysql.cj.jdbc.Driver").
         GetDeclaredConstructor().NewInstance();
         try (Connection conn = DriverManager.getConnection (url,
         user, pass)) {
            Statement stat = conn.createStatement();
```

```
                ResultSet resSet = stat.executeQuery ( "SELECT *
                FROM Products");
                while (resSet.next()) { // для поточного рядка
                   int id = resSet.getInt (1);
                   String n = resSet.getString (2);
                   int p = resSet.getInt (3);
                   System.out.printf("%d. %s - %d \n", id, n, p);
                }
             }
          }
          catch(Exception ex) {
             System.out.println("З'єднання немає ...");
             System.out.println(ex);
          }
       }
    }
```

Перший стовпець в таблиці – стовпець ***Id*** являє тип int, тому для отримання його значення використовується **метод *getInt()***. Другий стовпець – ***ProductName*** представляє рядок, тому, для отримання його даних застосовується **метод *getString()***. Таким чином, між типом даних стовпця таблиці і типом даних, що повертає метод є відповідність. Не можна, наприклад, отримати значення стовпця ***ProductName*** за допомогою методу ***getInt***.

Індексація стовпців починається з 1, а не з 0.

Однак не завжди можна точно знати порядок даних в отриманому наборі. У цьому випадку замість номерів стовпців можна передати назви стовпців:

```
ResultSet resSet = stat.executeQuery("SELECT * FROM
Products");
while (resSet.next()) {
    int id = resSet.getInt("Id");
    String n = resSet.getString("ProductName");
    int p = resSet.getInt("Price");
    System.out.printf("%d.%s -%d \n", id, n, p);
}
```

7.4. Створення БД засобами SQL-команд

Щоб створити нову базу даних з допомогою SQL призначено команду **`CREATE DATABASE`**, опис якої подано нижче. Тут і надалі, квадратні дужки позначають необов'язковий аргумент команди, символ "|" позначає вибір між аргументами.

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] ім'я_бази
    [[DEFAULT] CHARACTER SET кодування]
    [[DEFAULT] COLLATE набір_правил]
```

ім'я_бази – назва бази даних (латинські літери і цифри без пропусків);
кодування – набір символів і кодів (`koi8u`, `latin1`, `utf8`, `cp1250` тощо);
набір_правил – правила порівняння рядків символів (див. результат команди `show collation`).

Нижче наведені деякі допоміжні команди для роботи в SQL для середовища СУБД **PostgresSQL**. Кожна команда і кожен запит в командному рядку повинні завершуватись розділяючим символом ";".

1. Перегляд існуючих баз даних:

```
SHOW DATABASES
```

2. Вибір бази даних для подальшої роботи:

```
USE DATABASE ім'я_бази
```

3. Перегляд таблиць в базі даних:

```
SHOW TABLES [FOR ім'я_бази]
```

4. Перегляд опису таблиці в базі:

```
DESCRIBE ім'я_таблиці
```

5. Виконати набір команд з зовнішнього файлу:

```
SOURCE назва_файлу
```

6. Вивести результати виконання подальших команд у зовнішній файл:

```
\T назва_файлу
```

Для роботи зі схемою бази даних існують такі основні команди:

`ALTER DATABASE` – зміна опису бази даних;

`CREATE TABLE` – створення нової таблиці;

`ALTER TABLE` – зміна структури таблиці;

`DELETE TABLE` – видалення таблиці з бази даних;

`CREATE INDEX` – створення нового індексу (для швидкого пошуку даних);

`DROP INDEX` – видалення індексу;

`DROP DATABASE` – видалення бази даних.

Розглянемо команду створення таблиці в SQL та її основні аргументи.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] ім'я_таблиці
      [(опис_таблиці,...)]
      [додаткові_параметри]
      ... [вибірка_даних]
```

опис_таблиці:

```
назва_поля   опис_поля
| [CONSTRAINT [ім'я_обмеження]] PRIMARY KEY (назва_поля,...)
[тип_обмеження]
| {INDEX|KEY} [ім'я_обмеження] (назва_поля,...)[ тип_обмеження]
| [CONSTRAINT [ім'я_обмеження]] UNIQUE
[INDEX|KEY] [ім'я_обмеження](назва_поля,...)
[тип_обмеження]
| {FULLTEXT|SPATIAL} [INDEX|KEY]
[ім'я_обмеження] (назва_поля,...)
[тип_обмеження]
| [CONSTRAINT [ім'я_обмеження]] FOREIGN KEY
[ім'я_обмеження] (назва_поля,...)
опис_зв'язку
| CHECK (вираз)
```

опис_поля:

```
тип_даних [NOT NULL | NULL] [DEFAULT значення_за_замовчуванням]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
```

опис_зв'язку:

```
REFERENCES ім'я_таблиці (назва_поля,
      ...) [ON DELETE дія]
      [ON UPDATE дія]
```

дія:

`CASCADE`

Одночасне видалення, або оновлення відповідного значення у зовнішній таблиці.

`RESTRICT`

Аналог `NO ACTION`. Дія над значенням поля ігнорується, якщо існує відповідне йому значення у зовнішній таблиці. Опція задана за замовчуванням.

`SET NULL`

При дії над значенням у первинній таблиці, відповідне значення у зовнішній таблиці замінюється на `NULL`.

додаткові_параметри:

```
{ENGINE|TYPE} [=] тип_таблиці
| AUTO_INCREMENT [=] значення_приросту_лічильника
| AVG_ROW_LENGTH [=] значення
| [DEFAULT] CHARACTER SET [=] кодування
| CHECKSUM [=] {0 | 1}
| [DEFAULT] COLLATE [=] набір_правил
| COMMENT [=] 'коментар до таблиці'
| DATA DIRECTORY [=] 'абсолютний шлях'
| DELAY_KEY_WRITE [=] {0 | 1}
| INDEX DIRECTORY [=] 'абсолютний шлях'
| MAX_ROWS [=] значення
| MIN_ROWS [=] значення
| ROW_FORMAT {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
```

вибірка_даних:

```
[IGNORE | REPLACE] [AS] SELECT ... (вибір даних з інших таблиць)
```

вираз:

Логічний вираз, що повертає TRUE або FALSE.

Опис аргументів:

`ім'я_таблиці`
: Назва таблиці. Або назва_бази.назва_таблиці.

`тип_таблиці`
: В MySQL крім типів таблиць `MyISAM` та `InnoDB` існують типи `MEMORY, BDB, ARCHIVE` тощо.

`тип_обмеження`
: Задає тип індексу для ключового поля: `USING {BTREE | HASH | RTREE}`.

`TEMPORARY`
: Створення тимчасової таблиці, яка буде знищена після завершення зв'язку з сервером.

`CONSTRAINT`
: Вказує на початок оголошення `PRIMARY KEY`, `UNIQUE`, або `FOREIGN KEY` обмеження.

`NULL | NOT NULL`
: Директива, що дозволяє/забороняє null-значення для даного поля.

`PRIMARY KEY`
: Вказує, що дане поле буде первинним ключем в таблиці.

`UNIQUE`
: Вказує на те, що в даному полі будуть зберігатися унікальні значення.

`FOREIGN KEY ... REFERENCES`
: Створює зовнішній ключ, зв'язаний із вказаним полем (полями).

`AVG_ROW_LENGTH`
: Приблизне значення середньої довжини рядків зі змінною довжиною.

`DATA DIRECTORY`
: Вказує шлях, за яким таблиця має зберігатись у файловій системі.

`CHECKSUM`
: Якщо параметр = 1, то для рядків таблиці буде рахуватись контрольна сума. Це сповільнює оновлення таблиці, але робить легшим пошук пошкоджених таблиць.

`ROW_FORMAT`
: Вказує на спосіб зберігання рядків таблиці (залежно від типу таблиці).

`FULLTEXT|SPATIAL`
: Тип індексу (повнотекстовий/просторовий; тільки для таблиць типу MyISAM).

Основні типи даних у СУБД MySQL:

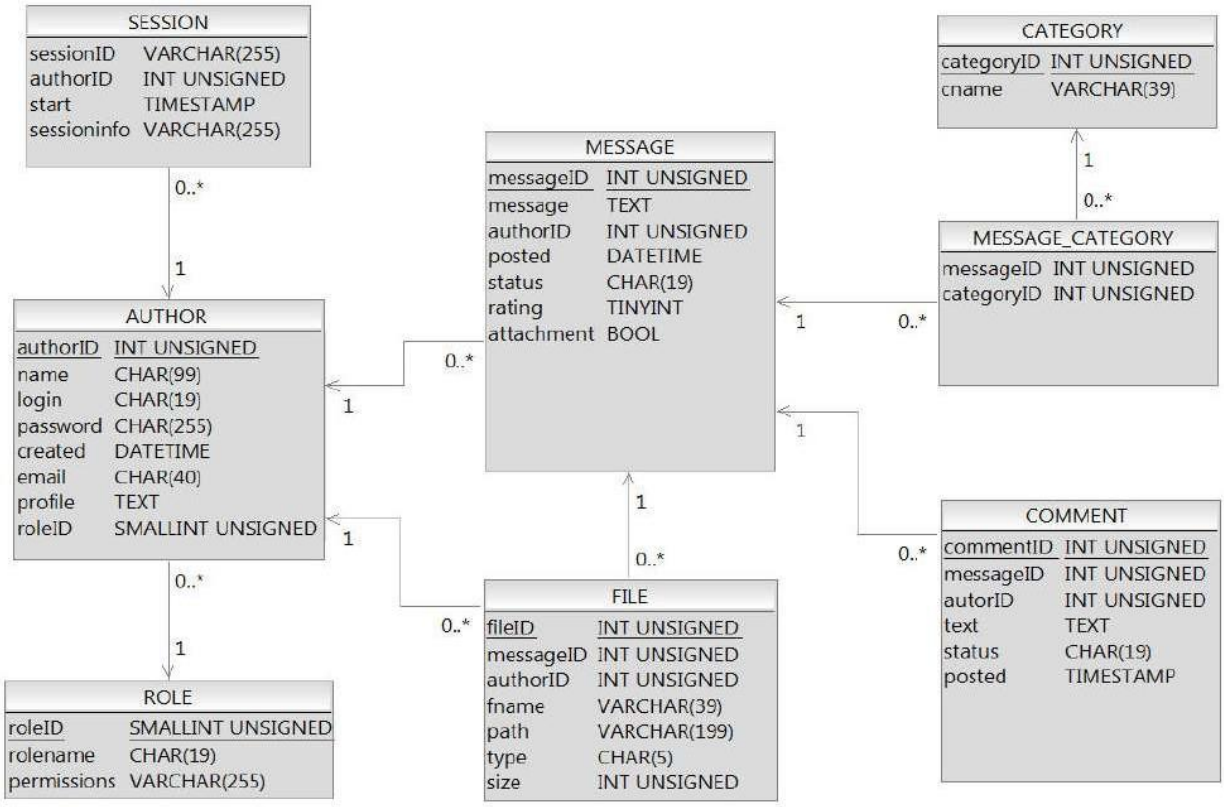
Тип даних	Опис
TINYINT[(*k*)] [UNSIGNED]	Ціле число з *k*-біт: -127 .. 128. UNSIGNED: 0 .. 255.
BOOL	Логічний тип (1-бітне число). Число 0 – фальш, відмінне від нуля – істина.
SMALLINT[(*k*)] [UNSIGNED]	Ціле число з *k*-біт: -32768 .. 32767. UNSIGNED: 0 .. 65535.
MEDIUMINT[(*k*)] [UNSIGNED]	Ціле число з *k*-біт: -8388608 .. 8388607. UNSIGNED: 0 .. 16777215.
INT[(*k*)] [UNSIGNED]	Ціле число з *k*-біт: -2147483648 .. 2147483647. UNSIGNED: 0 .. 4294967295.
BIGINT[(*k*)] [UNSIGNED]	-9223372036854775808 .. 9223372036854775807. UNSIGNED: 0 .. 18446744073709551615.
SERIAL	Синонім для типу BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
FLOAT[(*n*,*m*)] [UNSIGNED]	Число з плаваючою крапкою, де *n* – кількість всіх цифр, *m* – кількість цифр після крапки. Від -3.402823466E+38 до -1.175494351E-38 UNSIGNED: 1.175494351E-38 .. 3.402823466E+38
DOUBLE[(*n*,*m*)] [UNSIGNED]	Від -1.7976931348623157E+308 до -2.2250738585072014E-308 UNSIGNED: від 2.2250738585072014E-308 до 1.7976931348623157E+308.
DECIMAL[(*n*[,*m*])] [UNSIGNED]	Число з фіксованою крапкою. *n* – кількість цифр (максимально – 65), *m* – кількість цифр після крапки (максимально – 30, за замовчуванням – 0). UNSIGNED: від'ємні значення заборонені.
DATE	Дата. Від "1000-01-01" до "9999-12-31".
DATETIME	Дата і час. Від "1000-01-01 00:00:00" до "9999-12-31 23:59:59".
TIMESTAMP	Часова мітка. Може присвоюватись автоматично. Від "1970-01-01 00:00:01" до "2038-01-09 03:14:07"
TIME	Час у форматі "HH:MM:SS" (рядок або число).
CHAR[(*n*)]	Рядок з *n*-символів (макс. – 255, за замовчуванням – 1).
VARCHAR(*n*)	Рядок змінної довжини. Для кодування *utf8* максимальна довжина складає 21844 символи.
TEXT(*n*)	Рядок змінної довжини. Максимальна кількість однобайтових символів – 65535.
MEDIUMTEXT	16777215 однобайтових символів (16 Мб тексту).
BLOB	Бінарні дані (65535 байт).
MEDIUMBLOB	Бінарні дані (16 Мб)
LONGBLOB	Бінарні дані (4 Гб, залежно від налаштувань системи)
ENUM('знач1','знач2',...)	Перелік значень. Зберігається лише одне.
SET('знач1','знач2',...)	Множина значень. Зберігається одне, або більше (максимально – 64).

Можна дати декілька порад щодо розробки схеми бази даних і вибору типів даних. Вони дозволять уникнути повільного виконання запитів і потреби модифікації таблиць в майбутньому.

- Слід використовувати якомога менший тип даних для полів таблиць. Наприклад, для зберігання чисел від 1 до 64 краще використати тип `TINYINT(6)` замість `SMALLINT`. Це впливає на швидкість пошуку і вибірки даних.
- Слід використовувати рядки фіксованої довжини, якщо це можливо. Для цього всі поля таблиці повинні бути фіксованої довжини. Тобто, варто уникати типів `VARCHAR`, `TEXT` і `BLOB`. Це пришвидшить вибірку даних з середини рядків, оскільки ці дані будуть мати фіксовану адресу. При потребі використання полів з типами `TEXT` або `BLOB`, їх можна виділити в окрему таблицю.
- Якщо можливо, варто завжди використовувати поля з обмеженням `NOT NULL`. Хоча це може збільшувати об'єм бази на диску.
- SQL дозволяє використовувати різні типи таблиць в одній базі даних. Слід використовувати переваги різних типів (`MyISAM`, `INODB` тощо) залежно від характеру майбутнього використання таблиці.
- Потрібно створювати індекси, які пришвидчать пошук і вибірку даних.
- В рідкісних випадках можна денормалізувати схему з метою зменшення кількості операцій з об'єднання таблиць при складних запитах. Але при цьому ускладнюється задача збереження цілісності бази даних.

Приклад створення складно структурованої БД:

Нехай є деяка даталогічна модель (рис. 1), що описує предметну область – блог з відгуками. Вона вимагає визначення конкретних полів бази даних, їхніх типів, обмежень на значення, тощо. Для зв'язку коментарів і повідомлень встановлено обмеження цілісності «каскадне оновлення». Для полів `status` у таблицях `MESSAGE` та `COMMENT` визначено такий домен
– ("опубліковане", "неопубліковане", "видалене").



Створимо нову базу даних, виконавши такі команди:

```
CREATE DATABASE MyCMS CHARACTER SET utf8 COLLATE DEFAULT;
CREATE TABLE MyCMS.CATEGORY (
      categoryID INT UNSIGNED NOT NULL
```

```
      AUTO_INCREMENT, cname VARCHAR(39),
      PRIMARY KEY (categoryID)
);
CREATE TABLE MyCMS.ROLE (
       roleID SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
       rolename CHAR(19) NOT NULL,
       permissions VARCHAR(255) NOT
       NULL, PRIMARY KEY (roleID)  );
CREATE TABLE MyCMS.AUTHOR (
       authorID INT UNSIGNED NOT NULL
       AUTO_INCREMENT name CHAR(99) NOT NULL,
       login CHAR(19) NOT NULL,
       password CHAR(255) NOT
       NULL, created DATETIME
       NOT NULL, email CHAR(40)
       NOT NULL,
       profile TEXT,
       roleID SMALLINT UNSIGNED NOT
       NULL, PRIMARY KEY (authorID),
       CONSTRAINT author_role FOREIGN KEY (roleID)
       REFERENCES MyCMS.ROLE (roleID) ON DELETE NO ACTION ON
UPDATE NO ACTION  );
CREATE TABLE MyCMS.MESSAGE (
       messageID INT UNSIGNED NOT NULL AUTO_INCREMENT,
       message TEXT NOT NULL,
       authorID INT UNSIGNED NOT
       NULL, posted DATETIME NOT
       NULL,
       status ENUM ('published', 'unpublished', 'deleted',)
DEFAULT 'published',
       rating TINYINT,
       attachment BOOL DEFAULT 0,
       PRIMARY KEY (messageID),
       CONSTRAINT message_author FOREIGN KEY (authorID)
      REFERENCES MyCMS.AUTHOR (authorID) ON DELETE NO ACTION ON
UPDATE NO ACTION  );
CREATE TABLE MyCMS.COMMENT (
       commentID INT UNSIGNED NOT NULL
       AUTO_INCREMENT, messageID INT UNSIGNED NOT
       NULL,
       autorID INT UNSIGNED NOT
       NULL, text TEXT NOT NULL,
       status ENUM ('published', 'unpublished', 'deleted',)
DEFAULT 'published',
       posted TIMESTAMP NOT NULL,
       PRIMARY KEY (commentID),
       CONSTRAINT comment_message FOREIGN KEY (messageID)
       REFERENCES MyCMS.MESSAGE (messageID) ON DELETE CASCADE ON
UPDATE CASCADE  );
CREATE TABLE MyCMS.FILE (
       fileID INT UNSIGNED NOT NULL
       AUTO_INCREMENT, messageID INT UNSIGNED NOT
       NULL,
       authorID INT UNSIGNED NOT
       NULL, fname CHAR(39) NOT
       NULL,
```

```
        path CHAR(199) NOT
        NULL, type CHAR(5)
        NOT NULL, size INT
        UNSIGNED,
        UNIQUE UQ_File_Message (fileID, messageID),
        PRIMARY KEY (fileID),
        CONSTRAINT file_message FOREIGN KEY (messageID)
    REFERENCES MyCMS.MESSAGE (messageID) ON DELETE NO ACTION ON
UPDATE NO ACTION,
        CONSTRAINT file_author FOREIGN KEY (authorID)
    REFERENCES MyCMS.AUTHOR (authorID) ON DELETE NO ACTION ON
UPDATE NO ACTION
);
CREATE TABLE MyCMS.SESSION (
        sessionID VARCHAR(255) NOT
        NULL, authorID INT UNSIGNED
        NOT NULL, start TIMESTAMP NOT
        NULL,
        sessioninfo VARCHAR(255),
        CONSTRAINT session_author FOREIGN KEY (authorID)
        REFERENCES MyCMS.AUTHOR (authorID) ON DELETE NO ACTION ON
UPDATE NO ACTION
);
CREATE TABLE MyCMS.MESSAGE_CATEGORY (
        messageID INT UNSIGNED NOT
        NULL, categoryID INT UNSIGNED
        NOT NULL,
        CONSTRAINT cat_category FOREIGN KEY (categoryID)
 REFERENCES MyCMS.CATEGORY (categoryID) ON DELETE NO ACTION ON
UPDATE NO ACTION,
        CONSTRAINT cat_message FOREIGN KEY (messageID)
 REFERENCES MyCMS.MESSAGE (messageID) ON DELETE NO ACTION ON
UPDATE NO ACTION
);
```

Індивідуальні завдання:

(Для виконання індивідуальних завдань № варінта є для 532 групи порядковим номером прізвища студента в списку групи, для 531 – вказаний у додатку (окремий документ). Усі проекти та, за наявності, тести до них завантажити у власні репозиторії на Git Hub. Посилання на репозиторії проектів вказати у звіті та обов'язково долучати скріни успішного виконання)

1. Розробити графічний програмний додаток для роботи з базою даних СКБД (PostreSQL, MySQL, MsSQL Server,…). Базу даних потрібно попередньо створити можливостями СКБД. Передбачити не менше 5 полів різного типу (придумайте самостійно). Початковий масив записів у файлі не менше 3. Для ідентифікації спроби введення з клавіатури некоректних даних описати виключення. Реалізуватиу програмі методи для:

- ***– додавання записів у БД;***
- ***– редагування записів у БД;***
- ***– знищення записів у БД;***
- ***– виведення впорядкованої за різними параметрами інформації з БД на екран.***
- ***– пошук та виведення даних за певним параметром.***

(4 бали)

№ варіанта	Клас	№ варіанта	Клас
1.	Пацієнти сімейного лікаря	13	Профілі клієнтів інтернет-магазину
2.	Викладачі університету	14	Камера схову
3.	Власники квартир ОСББ	15	Склад товарів
4.	Плейлист	16.	Каса продажу квитків
5.	Служба кур'єрської доставки	17.	Успішність студентів
6	Асортимент виробів	18.	Реєстр виборців закріплених за виборчою дільницею
7	Метеорологічні дані	19.	Відділ кадрів
8	Склад автозапчастин	20.	Розклад руху літаків
9	Бібліотека школи	21.	Меню ресторану
10	Розклад пар	22.	Записна книжка
11	Список файлів	23.	ЖРЕПи міста
12	Архів програмного забезпечення	24.	Абоненти кабельної мережі
25.		Рахунки банку	

2. Написати програмний додаток з багатовкладковим графічним інтерфейсом користувача, який би дозволяв здійснювати повноцінну роботу з базою даних, згідно свого варіанту, певної СКБД (PostreSQL, MySQL, MsSQL Server,…). База даних має містити щонайменше три таблиці, які пов'язані між собою відповідними зв'язками. Наприклад, база даних, яка складається з таблиць «Клієнт», «Замовлення», «Товар». Робота з базою даних через відповідний програмний додаток повинна забезпечувати додавання, видалення, редагування даних таблиць, вибірку всіх даних таблиць, вибірку за певними параметрами.

(6 балів)

№ варіанта	Клас	№ варіанта	Клас
1.	Залізничний вокзал	13	Аеропорт
2.	Футбольна команда	14	Бібліотека
3.	Салон краси	15	Укладання розкладу занять
4.	Обслуговування обмінного пункту валют	16.	Фонотека
5.	Пекарня	17.	Кінотеатр
6	Інформаційна система порту	18.	Аптека
7	Хореографічний колектив	19.	Магазин туристичного обладнання
8	Стоматологічна клініка	20.	Автомобільний салон
9	Прокат відеофільмів	21.	Фотосалон
10	Лікарня	22.	Дилерська мережа з продажу вин
11	Косметична продукція	23.	Податкова інспекція
12	Компанія радіомовлення	24.	Автовокзал
25.		Готель	