

LẬP TRÌNH JAVA

JDBC

ThS. Dương Hữu Thành
Khoa CNTT, Đại học Mở Tp.HCM
thanh.dh@ou.edu.vn



1



Nội dung chính

- 1. Giới thiệu MySQL**
- 2. Giới thiệu JDBC**
- 3. Kiến trúc JDBC**
- 4. JDBC API**
- 5. Sử dụng JDBC tương tác MySQL**
- 6. Giao tác**
- 7. Batch Query**



Giới thiệu MySQL

- MySQL là một **cơ sở dữ liệu quan hệ, mã nguồn mở** được sử dụng phổ biến trong phát triển các ứng dụng từ nhỏ đến lớn.
- MySQL hỗ trợ trên nhiều hệ điều hành và các ngôn ngữ lập trình khác nhau như PHP, Java, Python, Ruby, v.v. MySQL làm việc tốt ngay cả với các tập dữ liệu lớn.
- MySQL sử dụng hình thức chuẩn của ngôn ngữ **truy vấn dữ liệu SQL** phổ biến nên cú pháp truy vấn thông dụng của nó cũng gần giống với các truy vấn trong MS SQL Server.

Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

3

3



Giới thiệu MySQL

- Tải bản **MySQL Community Server** tại
<https://dev.mysql.com/downloads/mysql/>
sau đó double-click vào file cài đặt và thực hiện theo các hướng dẫn để cài.
- Trong quá trình cài đặt nên chọn cài luôn công cụ **MySQL Workbench** hỗ trợ tương tác với cơ sở dữ liệu MySQL dễ dàng hơn.

Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

4

4



Giới thiệu MySQL

- Chèn dữ liệu

```
INSERT INTO <tên-bảng> [(cột-1, cột-2,  
..., cột-n)] VALUES (giá-trị-1, giá-trị-2,  
..., giá-trị-n);
```

- Cập nhật dữ liệu

```
UPDATE <tên-bảng>  
SET <cột-1> = <giá-trị-1> [, ..., <cột-n>  
= <giá-trị-n>]  
[WHERE <điều-kiện>];
```



Giới thiệu MySQL

- Xoá dữ liệu

```
DELETE FROM <tên-bảng>  
[WHERE <điều-kiện>]
```

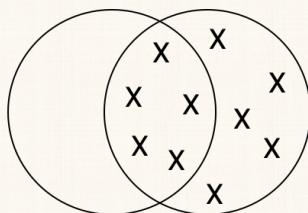
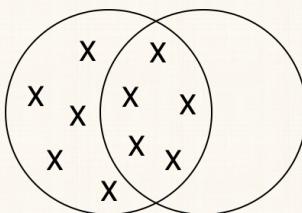
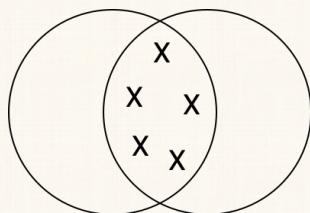
- Truy vấn dữ liệu

```
SELECT <cột-1> [<cột-2>, ..., <cột-n>]  
FROM <tên-bảng>  
[WHERE <điều-kiện>]  
[GROUP BY <cột-i>]  
[HAVING <điều-kiện>]  
[ORDER BY <cột-i> [ASC/DESC]]  
[OFFSET M] [LIMIT N]
```



Giới thiệu MySQL

- Phép kết:



Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

7

7



Giới thiệu MySQL

- Stored procedure

```
CREATE PROCEDURE [procedure_name] ([IN/OUT  
param1, IN/OUT param2,...])  
BEGIN  
    [sql_statements]  
END
```

- [procedure_name]: tên stored procedure
- param1, param2, ...: danh sách các tham số.
- IN/OUT chỉ định tham số đầu vào hoặc đầu ra.

Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

8

8



Giới thiệu MySQL

- Gọi sử dụng stored procedure

```
CALL procedure_name ([param1, param2,...]);
```

- Ví dụ

```
CREATE PROCEDURE countCategory(OUT c INT)
BEGIN
    SELECT COUNT(*) INTO c
    FROM category;
END
```

```
CALL countCategory(@c);
SELECT @c;
```



Giới thiệu MySQL

- Ví dụ

```
CREATE PROCEDURE getCatById(IN param INT)
BEGIN
    SELECT * FROM category WHERE id=param;
END
```

```
CALL getCatById(1);
```



Giới thiệu JDBC

- JDBC → Java Database Connectivity
- JDBC cung cấp các **API** để phát triển các ứng dụng tương tác với cơ sở dữ liệu quan hệ bằng Java.
- JDBC hỗ trợ nhiều cơ sở dữ liệu quan hệ khác nhau như MySQL, Postgres SQL, MS SQL Server, v.v.

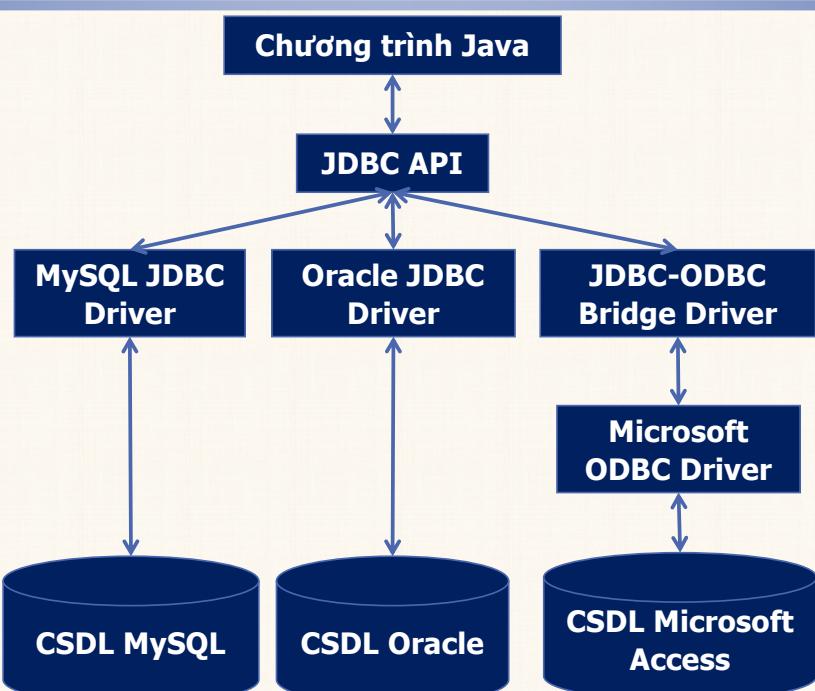
Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

11

11



Kiến trúc JDBC



Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

12

12



Kiến trúc JDBC

- JDBC API là một **tập các giao diện** và lớp viết bằng Java để truy cập, thao tác trên cơ sở dữ liệu quan hệ.
- Để tương tác với từng cơ sở dữ liệu **cần có một driver** tương ứng của nó, thường được cung cấp bởi nhà cung cấp cơ sở dữ liệu đó, các driver này phải hiện thực hóa lại JDBC API.



JDBC API

- **DriverManager**: lớp này **quản lý** driver của cơ sở dữ liệu.
- **Driver**: giao diện này xử lý các giao tiếp cơ sở dữ liệu, thường ta không tương tác trực tiếp giao diện này, mà thay vào đó sử dụng lớp DriverManager.
- **Connection**: giao diện này chứa các phương thức để thiết lập kết nối đến cơ sở dữ liệu.



JDBC API

- **Statement:** giao diện này cung cấp các phương thức để **thực thi câu truy vấn**, stored procedure.
- **ResultSet:** đối tượng của giao diện này đại diện cho **dữ liệu được trả về** khi thực thi câu truy vấn.
- **SQLException:** lớp này dùng **xử lý các lỗi** xảy ra trong quá trình tương tác với cơ sở dữ liệu.



Sử dụng JDBC tương tác MySQL

- Các bước tương tác cơ sở dữ liệu bằng JDBC
 - **Bước 1. Nạp Driver**
 - **Bước 2. Thiết lập kết nối**
 - **Bước 3. Thực thi truy vấn**
 - **Bước 4. Xử lý kết quả trả về**
- **Nếu sử dụng maven cần thêm dependency:**

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.19</version>
</dependency>
```



Nạp Driver

- Cú pháp nạp Driver

```
Class.forName(<driver-path>)
```

- Đối với MySQL

- MySQL 5.*

```
Class.forName("com.mysql.jdbc.Driver")
```

- MySQL 8.*

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

- Chú ý ngoại lệ **ClassNotFoundException** sẽ được ném ra nếu lớp Driver không tìm thấy.



Thiết lập kết nối

- Cú pháp

```
import java.sql.Connection  
DriverManager.getConnection("databaseUrl",  
username, password)
```

- databaseUrl của MySQL

- jdbc:mysql://hostname:port/databaseName

- Đóng kết nối bằng phương thức close().

```
Connection connection  
= DriverManager.getConnection(  
    "jdbc:mysql://localhost/bookstore",  
    "root", "123456"  
) ;
```



Thực thi truy vấn

- Sử dụng Statement
 - connection.createStatement()
 - executeQuery → thao tác dữ liệu
 - executeUpdate → định nghĩa dữ liệu

```
Statement statement = connection.createStatement();
String query = "INSERT INTO author (author_code,
first_name, last_name) VALUES ('AUTHOR0001', 'Thanh',
'Duong');";
int result = statement.executeUpdate(query);
```



Thực thi truy vấn

- Sử dụng PreparedStatement: dùng tạo các câu lệnh SQL cần truyền đối số,
 - Để tạo đối tượng kiểu này dùng phương thức **prepareStatement()** của Connection.

```
String query = "INSERT INTO author (author_code ,
first_name, last_name)VALUES (?, ?, ?);";
PreparedStatement preparedStatement
        = connection.prepareStatement(query);
preparedStatement.setString(1, "AUTHOR0010");
preparedStatement.setString(2, "Le");
preparedStatement.setString(3, "Duong");
preparedStatement.executeUpdate();
```



Xử lý kết quả trả về

- Phương thức `executeQuery()` trả về một thể hiện của `java.sql.ResultSet` chứa tất cả các dòng dữ liệu tìm thấy.
- Để lấy dữ liệu một dòng của ResultSet, ta có thể dựa vào **chỉ số cột** hoặc **tên cột** trong cơ sở dữ liệu bằng các phương thức tương ứng từng kiểu dữ liệu của cột muốn lấy.



Xử lý dữ liệu trả về

- Ví dụ

```
Statement statement = connection.createStatement();

String query = "SELECT * FROM author "
    + "WHERE author_code='AUTHOR0001'";
ResultSet result = statement.executeQuery(query);

while (result.next()) {
    System.out.println("First Name: "
        + result.getString("first_name"));
    System.out.println("Last Name: "
        + result.getString("last_name"));
}
```



Xử lý dữ liệu trả về

- Ví dụ: đếm số lượng sản phẩm từng danh mục.

```
Statement stm = conn.createStatement();
String sql = "SELECT C.id, C.name, count(P.id) AS num "
        + "FROM product P RIGHT OUTER JOIN category C "
        + "ON C.id=P.category_id "
        + "GROUP BY C.id, C.name ";

ResultSet rs = stm.executeQuery(sql);
while (rs.next()) {
    System.out.printf("%d: %s - %d\n",
        rs.getInt("C.id"),
        rs.getString("C.name"),
        rs.getInt("num"));
}
```



Xử lý dữ liệu trả về

- Việc di chuyển cursor của ResultSet phụ thuộc tham số RSType lúc tạo Statement trong các phương thức sau:

```
createStatement(int RSType, int RSCurrency)
prepareStatement(String q, int RSType, int RSCurrency)
prepareCall(String q, int RSType, int RSCurrency)
```



Xử lý dữ liệu trả về

- Trong đó `RSType` có thể nhận giá trị:
 - `ResultSet.TYPE_FORWARD_ONLY`: cursor chỉ di chuyển tới (mặc định).
 - `ResultSet.TYPE_SCROLL_INSENSITIVE`: cursor có thể di chuyển tới hoặc lùi, không quan tâm sự thay đổi dưới cơ sở dữ liệu bởi thread khác.
 - `ResultSet.TYPE_SCROLL_SENSITIVE`: cursor có thể di chuyển tới hoặc lùi, phản ảnh những thay đổi dưới cơ sở dữ liệu bởi thread khác.



Các phương thức di chuyển cursor

- `first()`: di chuyển cursor đến dòng đầu.
- `last()`: di chuyển cursor đến dòng cuối.
- `previous()`: di chuyển cursor đến dòng trước, trả về false nếu đang ở dòng đầu.
- `next()`: di chuyển cursor đến dòng sau, trả về false nếu đang ở dòng cuối.
- `absolute(int row)`: di chuyển cursor đến dòng chỉ định.
- `relative(int row)`: di chuyển cursor $|row|$ dòng chỉ định từ vị trí hiện tại, nếu $row > 0$ thì di chuyển tới, còn $row < 0$ thì di chuyển lùi.



Sử dụng CallableStatement

- Sử dụng CallableStatement thực thi stored procedure.
- Đối tượng CallableStatement được tạo bằng cách sử dụng phương thức **prepareCall()** của Connection.
- Ví dụ ta có stored procedure như sau:

```
CREATE PROCEDURE countCategory(OUT c INT)
BEGIN
    SELECT COUNT(*) INTO c
    FROM category;
END
```



Sử dụng CallableStatement

- Thực thi bằng CallableStatement

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost/demo",
    "root", "12345678");

CallableStatement stm = conn.prepareCall("{call
countCategory(?)}");
stm.registerOutParameter(1, Types.INTEGER);
stm.execute();
System.out.println(stm.getInt(1));

stm.close();
conn.close();
```



Sử dụng CallableStatement

- Ví dụ stored procedure như sau:

```
CREATE PROCEDURE getCatById(IN param INT)
BEGIN
    SELECT * FROM category WHERE id=param;
END
```

- Thực thi

```
CallableStatement stm = conn.prepareCall("{call
getCatById(?) }");
stm.setInt(1, 1);
ResultSet rs = stm.executeQuery();
while (rs.next()) {
    System.out.printf("%d - %s\n",
                      rs.getInt("id"), rs.getString("name"));
}
```



Giao tác

- Giao tác (transaction) đại diện cho một **đơn vị xử lý** có nhiều thao tác, được hỗ trợ bởi hầu hết các hệ quản trị cơ sở dữ liệu quan hệ.
- Giao diện Connection mặc định sử dụng chế độ autocommit để làm việc với giao tác, tức là mỗi câu lệnh (Statement) trong kết nối sẽ được thực thi và commit lưu trữ ngay xuống cơ sở dữ liệu.



Tính chất ACID Giao tác

- Tính nguyên tố (Atomicity): một giao tác là một đơn vị xử lý nguyên tố, không thể chia nhỏ, tức là một giao tác thành công nếu tất cả các thao tác của nó thành công, ngược lại tồn tại một thao tác thất bại thì toàn bộ giao tác thất bại.
- Tính nhất quán (Consistency): một giao tác phải chuyển cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác.



Tính chất ACID Giao tác

- Tính cô lập (Isolation): có thể có nhiều giao tác xử lý cùng một đơn vị dữ liệu ở cùng một thời điểm, nhưng một giao tác phải được cô lập với các giao tác khác, tức là giao tác này không thể thấy được tác động của giao tác khác nếu nó chưa hoàn thành.
- Tính bền vững (Durability): một giao tác khi hoàn tất, kết quả của giao tác sẽ tác động bền vững và không thể xoá đi dưới cơ sở dữ liệu từ những vấn đề của hệ thống.



Giao tác

- Dùng phương thức **setAutoCommit(false)** để tắt chế độ autocommit, khi đó các câu lệnh trong một kết nối sẽ được nhóm lại và xử lý như một đơn vị đến khi lệnh gặp
 - **commit()** được thực thi để lưu những thay đổi xuống cơ sở dữ liệu
 - **rollback()** để phục hồi lại những thay đổi đã thực hiện trong giao tác.

Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

33

33



Giao tác

- Giao tác

```
String query = "INSERT INTO author ("  
        + " author_code , first_name, last_name)"  
        + " VALUES ('%s', '%s', '%s');";  
  
conn.setAutoCommit(false);  
Statement stm = conn.createStatement();  
stm.executeUpdate(String.format(query,  
                                "AUTHOR0023", "Tai", "Nguyen"));  
stm.executeUpdate(String.format(query,  
                                "AUTHOR0024", "Phát", "Lê"));  
conn.commit();
```

Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

34

34



Batch Query

- Ta có thể gom các lệnh truy vấn liên quan thành một lô (batch) để thực thi cùng một lúc để giảm bớt được số lần tương tác với cơ sở dữ liệu và có thể cải thiện được hiệu năng thực thi của chương trình.
- Kiểm tra hệ quản trị cơ sở dữ liệu có hỗ trợ khả năng này hay không.
 - `DatabaseMetaData.supportsBatchUpdates()`



Batch Query

- Phương thức `addBatch()` của Statement dùng thêm một câu lệnh vào batch.
- Phương thức `executeBatch()` thực thi batch, nó trả về mảng các số nguyên, mỗi phần tử đại diện số dòng bị ảnh hưởng thực thi bởi lệnh truy vấn tương ứng.
- Xóa các lệnh đã được thêm vào batch bằng `clearBatch()`.



Batch Query

- Ví dụ

```
DatabaseMetaData metadata = conn.getMetaData();

if (metadata.supportsBatchUpdates()) {
    conn.setAutoCommit(false);

    Statement stm = conn.createStatement();
    stm.addBatch(String.format(query,
        "AUTHOR0025", "Lộc", "Trần"));
    stm.addBatch(String.format(query,
        "AUTHOR0026", "Phúc", "Hồ"));
    int[] kq = stm.executeBatch();

    conn.commit();
}
```



Spring JDBC

- Sử dụng Spring JDBC tương tác cơ sở dữ liệu giúp giảm bớt một số mã nguồn không cần thiết, giúp phát triển ứng dụng nhanh và hiệu quả hơn.
- Nó hỗ trợ trong tất cả các công đoạn tương tác cơ sở dữ liệu từ mở kết nối, thực thi truy vấn, xử lý kết quả trả về, xử lý ngoại lệ cho đến khi đóng kết nối.
- Lớp JdbcTemplate là lớp quan trọng nhất trong Spring JDBC quản lý tất cả các tương tác với cơ sở dữ liệu.



Spring JDBC

▪ Cấu hình bean

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
              value="com.mysql.cj.jdbc.Driver" />
    <property name="url"
              value="jdbc:mysql://localhost:3306/endb" />
    <property name="username" value="root" />
    <property name="password" value="12345678" />
</bean>
```



Spring JDBC

▪ Tạo đối tượng JdbcTemplate

```
ApplicationContext context
        = new ClassPathXmlApplicationContext("Beans.xml");
DataSource dataSource = (DataSource)
context.getBean("dataSource");
JdbcTemplate t = new JdbcTemplate(dataSource);
```

▪ Ví dụ cập nhật dữ liệu

```
String sql = "UPDATE category SET name=? WHERE id=?";
t.update(sql, "TEST", "Cate01");
```



Spring JDBC

▪ Truy vấn dữ liệu

```
List<Category> cats = t.query("SELECT * FROM category",
                               (rs, rowNum) ->{
    Category c = new Category();
    c.setId(rs.getString("id"));
    c.setName(rs.getString("name"));
    return c;
});
cats.stream().forEach(c -> System.out.println(c.getName()));
```



Spring JDBC

▪ Truy vấn dữ liệu có truyền tham số

```
List<Question> questions
= JdbcUtils.getTemplate().query("SELECT * FROM question LIMIT ?",
                               (rs, i) -> {
    Question q = new Question();
    q.setId(rs.getString("id"));
    q.setContent(rs.getString("content"));

    return q;
}, num);
```

Q&A

Dương Hữu Thành, Khoa CNTT, Đại học Mở Tp.HCM

43