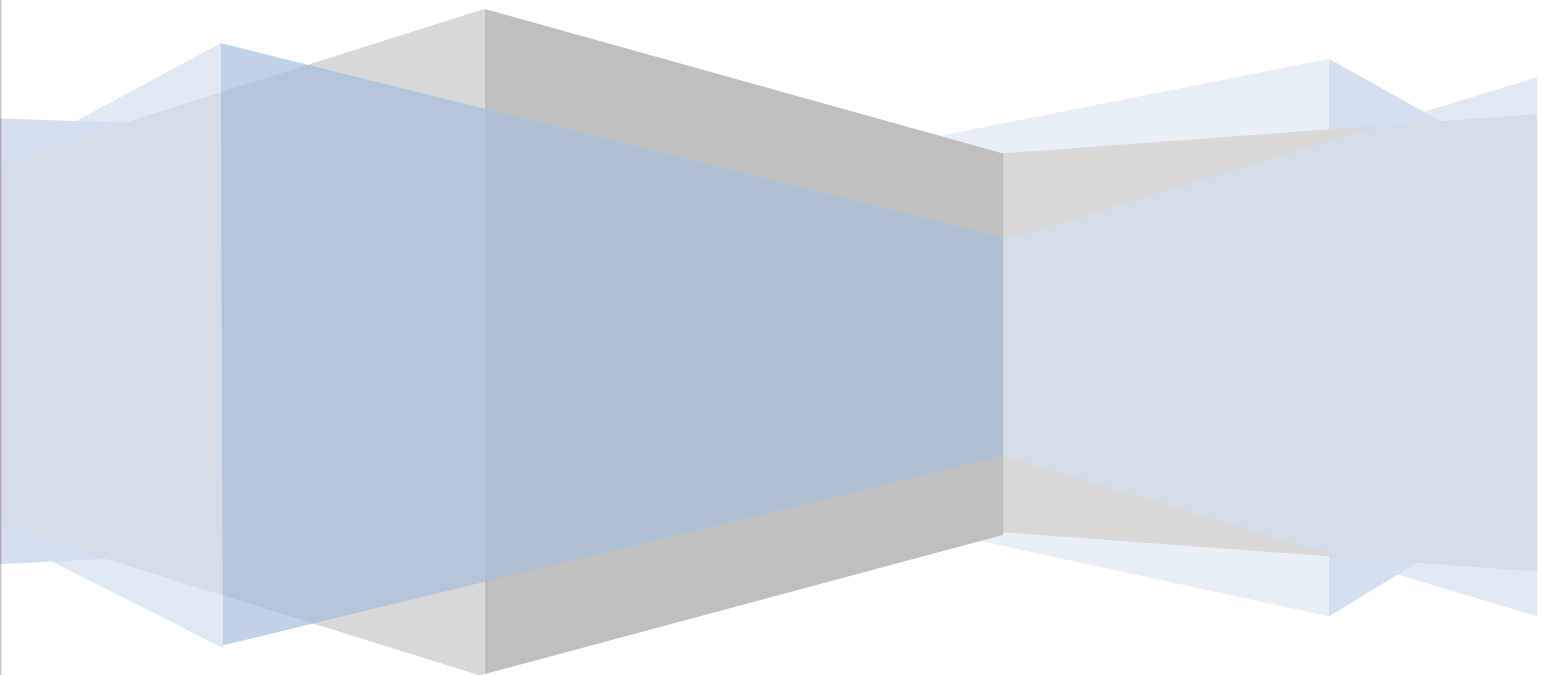
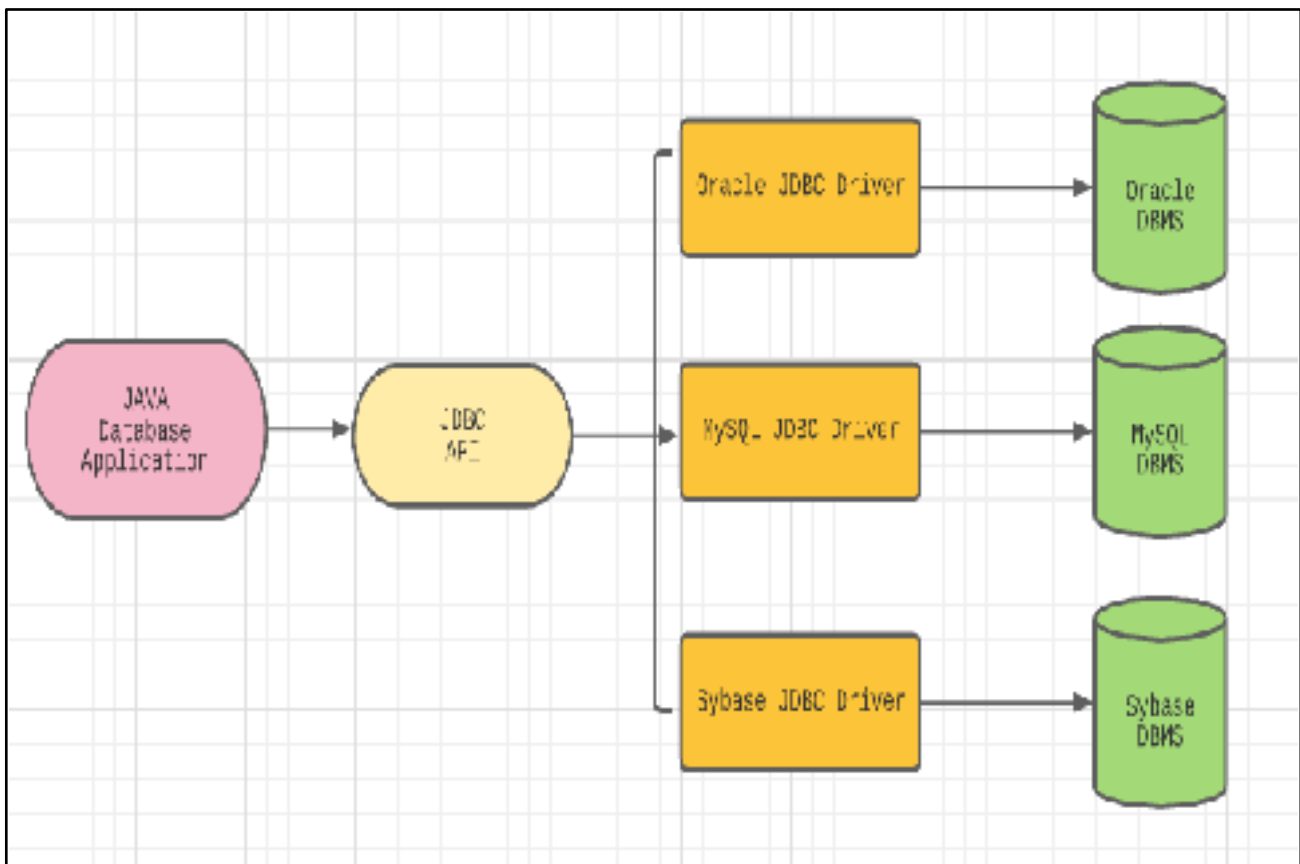


자바 데이터베이스 프로그래밍



JDBC 란?

- Java Database Connectivity
- 자바 프로그램에서 다른 기종 간의 데이터베이스를 표준화된 방법으로 접속할 수 있도록 만든 API (Application Programming Interface) 이다.

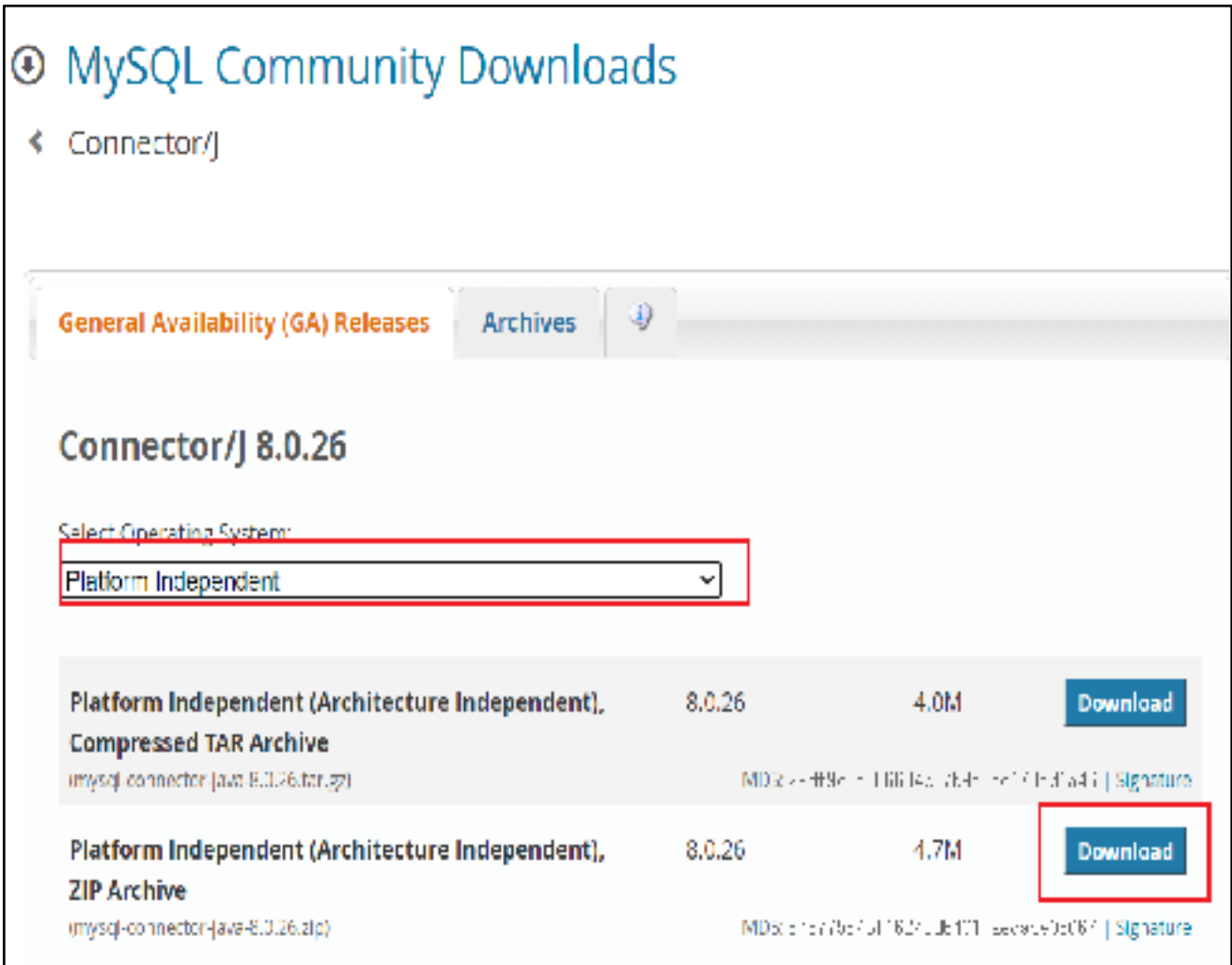


※ JDBC Driver

- 데이터베이스에 대한 연결(Connection)을 제공하고 클라이언트와 데이터베이스 간에 쿼리 및 결과를 전송하는 기능을 제공한다.
- 클라이언트에 JDBC 드라이버를 설치한다.

MySQL JDBC Driver 다운로드

· <https://dev.mysql.com/downloads/connector/j/>



MySQL Community Downloads

Connector/J

General Availability (GA) Releases Archives

Connector/J 8.0.26

Select Operating System:

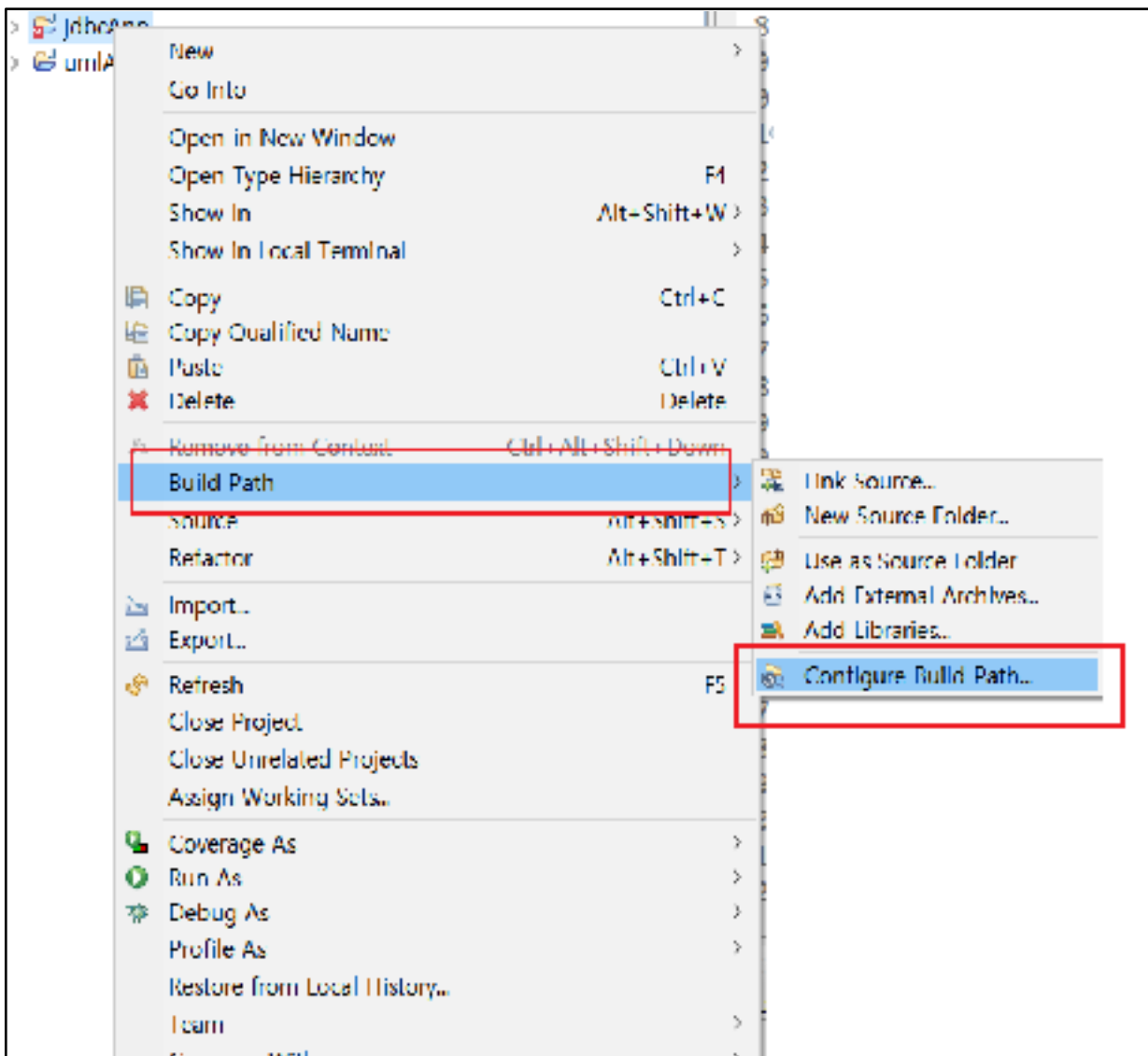
Platform Independent

Platform Independent (Architecture Independent), Compressed TAR Archive <small>(mysql-connector-java-8.0.26.tar.gz)</small>	8.0.26	4.0M	Download
Platform Independent (Architecture Independent), ZIP Archive <small>(mysql-connector-java-8.0.26.zip)</small>	8.0.26	4.7M	Download

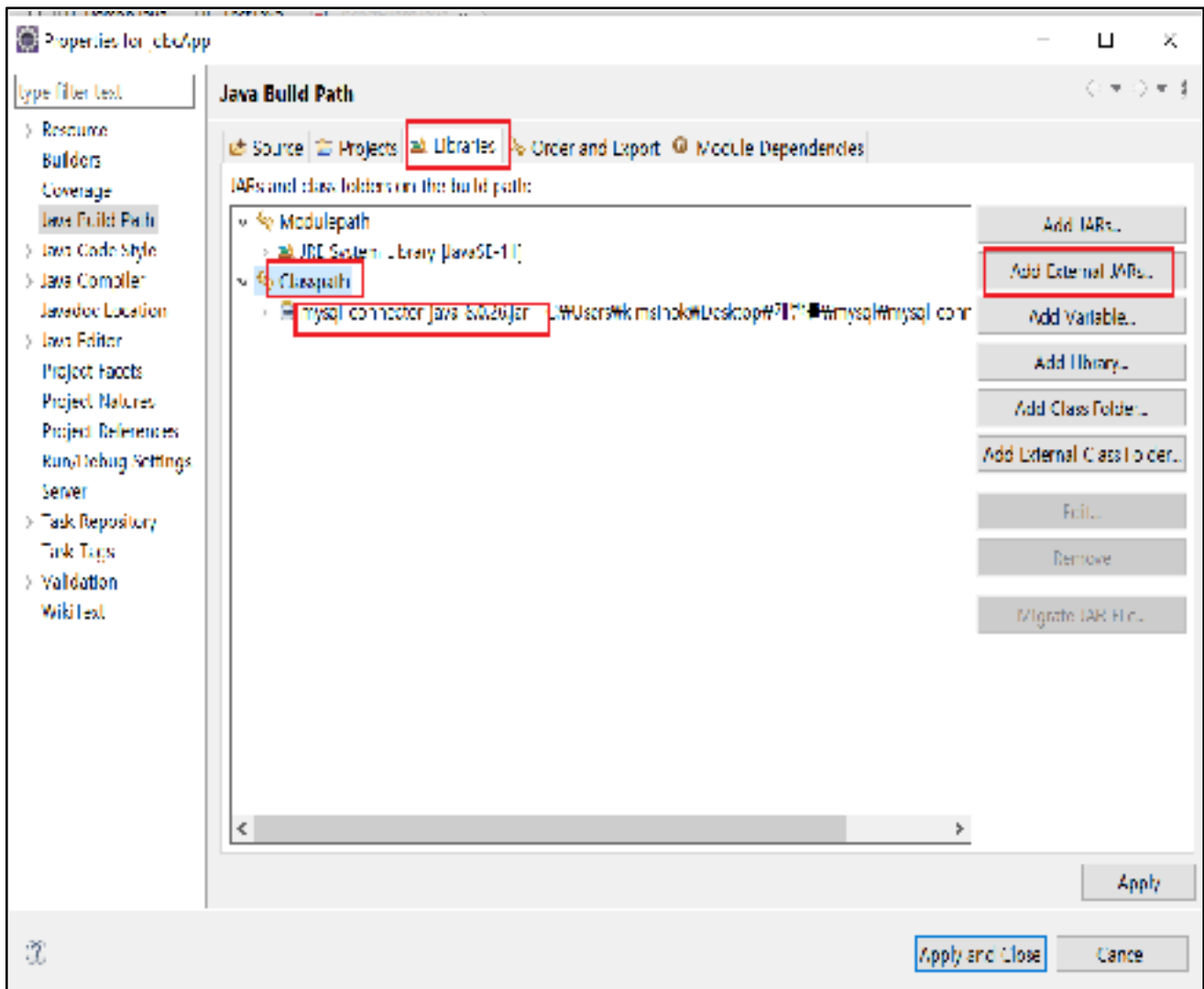
※ mysql-connector-java-8.0.26.zip 파일을 다운로드 한 후 압축을 풀면 폴더 안에 **mysql-connector-java-8.0.26.jar** 라는 파일이 MySQL JDBC Driver 이다.

Eclipse - MySQL JDBC Driver 설정

1. 프로젝트를 선택한 후 마우스 우 클릭한다.
2. Build Path -> Configure Build Path... 선택한다.



3. Libraries 탭에서 Classpath 항목을 선택 한 후 Add External JARs... 버튼을 선택한다.
4. **mysql-connector-java-8.0.26.jar** (MySQL JDBC Driver) 파일을 선택한다.

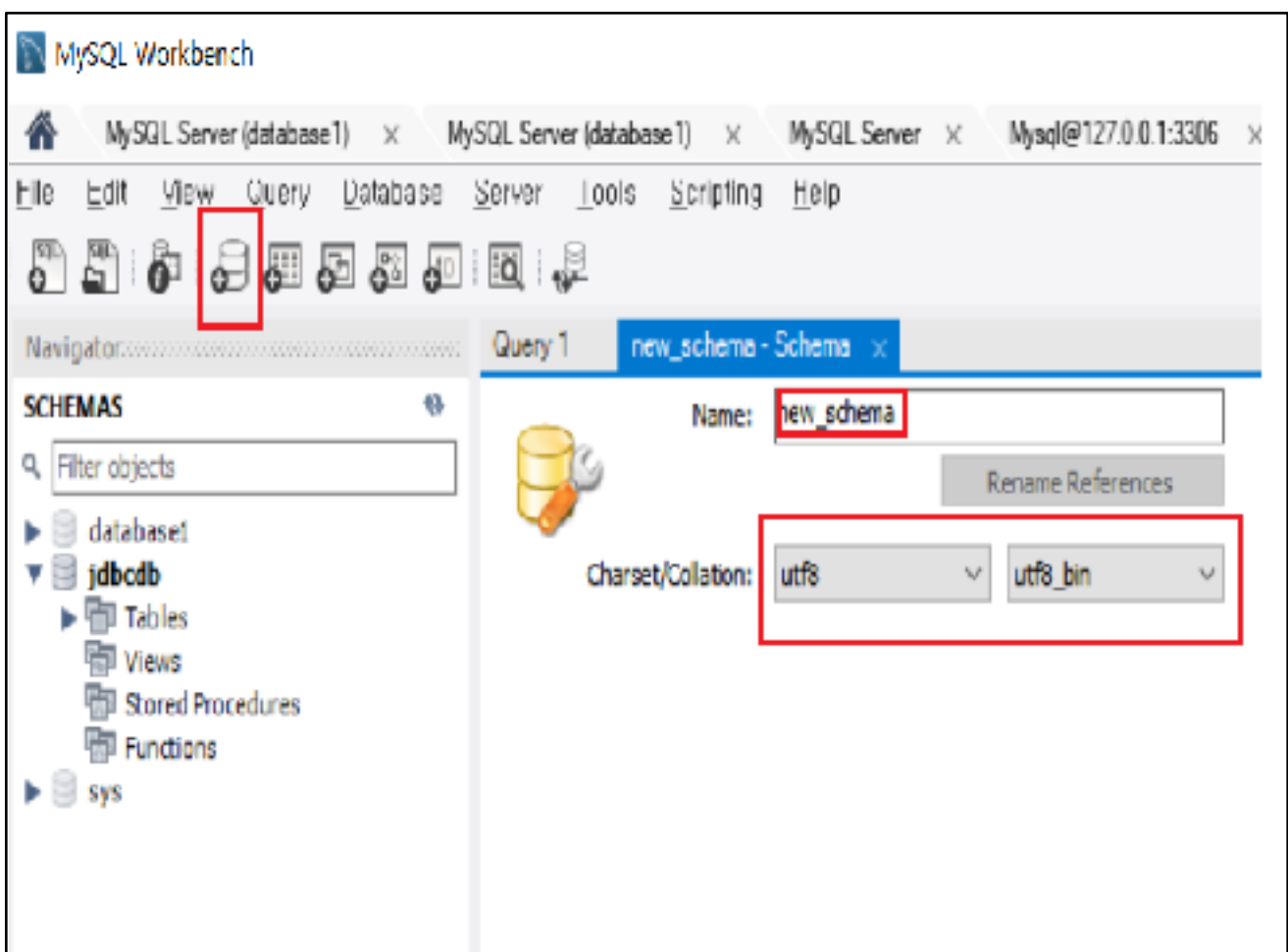


MySQL Workbench - 스키마 생성

[참고] <https://goddaehee.tistory.com/278> 사이트

Schema Name : jdbcdb

Charset : utf8, utf8_bin

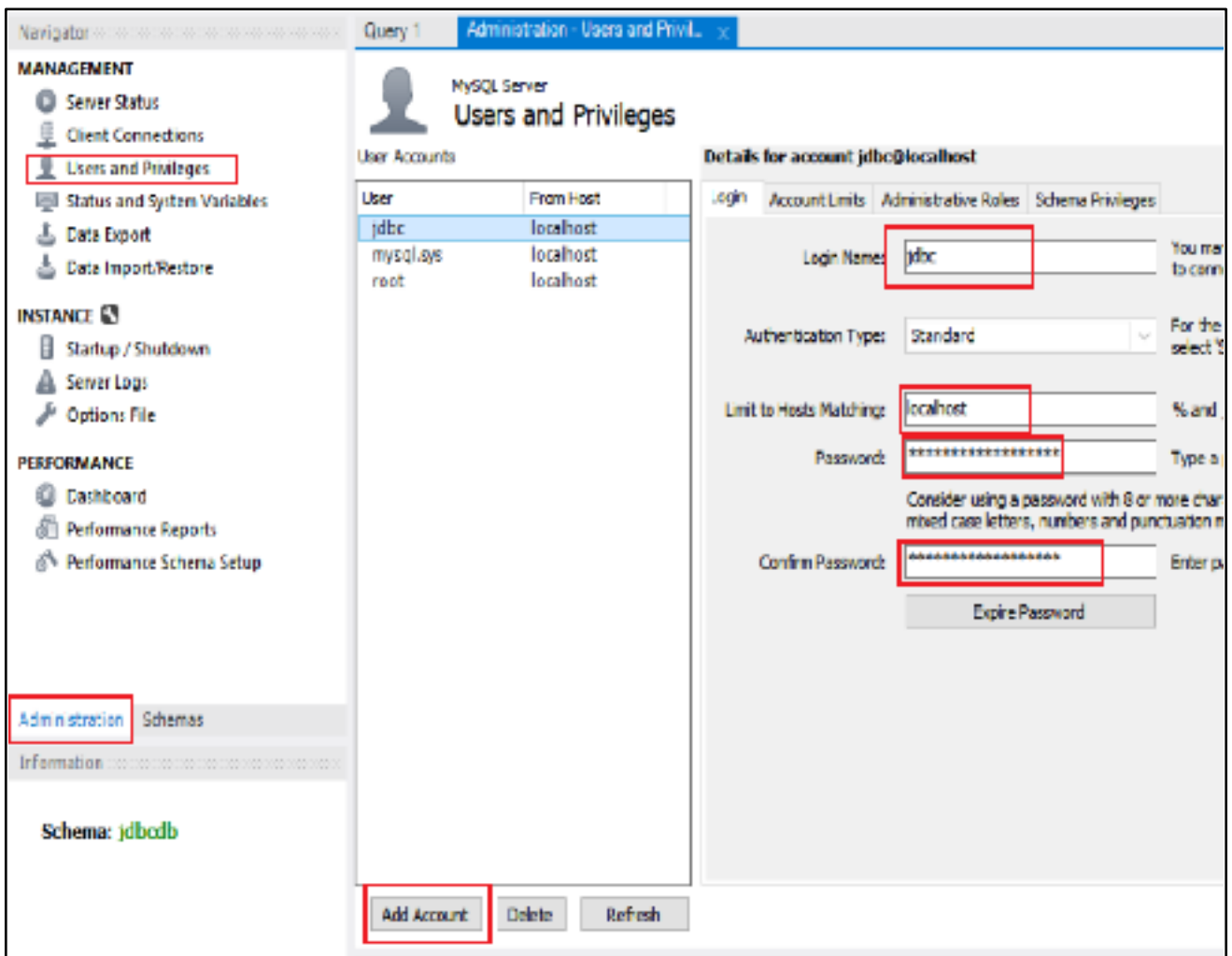


MySQL Workbench - 사용자 계정 생성

Login Name : jdbc

Limit to Hosts Matching : localhost

Password : jdbc1234



The screenshot shows the MySQL Workbench interface with the 'Users and Privileges' window open. The left sidebar has 'Users and Privileges' selected under the 'MANAGEMENT' section. The main window displays a table of existing users and a detailed configuration panel for a new user 'jdbc@localhost'.

User	From Host
jdbc	localhost
mysql.sys	localhost
root	localhost

Details for account jdbc@localhost

- Login Name:** jdbc
- Authentication Type:** Standard
- Limit to Hosts Matching:** localhost
- Password:** jdbc1234
- Confirm Password:** jdbc1234

At the bottom of the window, the 'Add Account' button is highlighted.

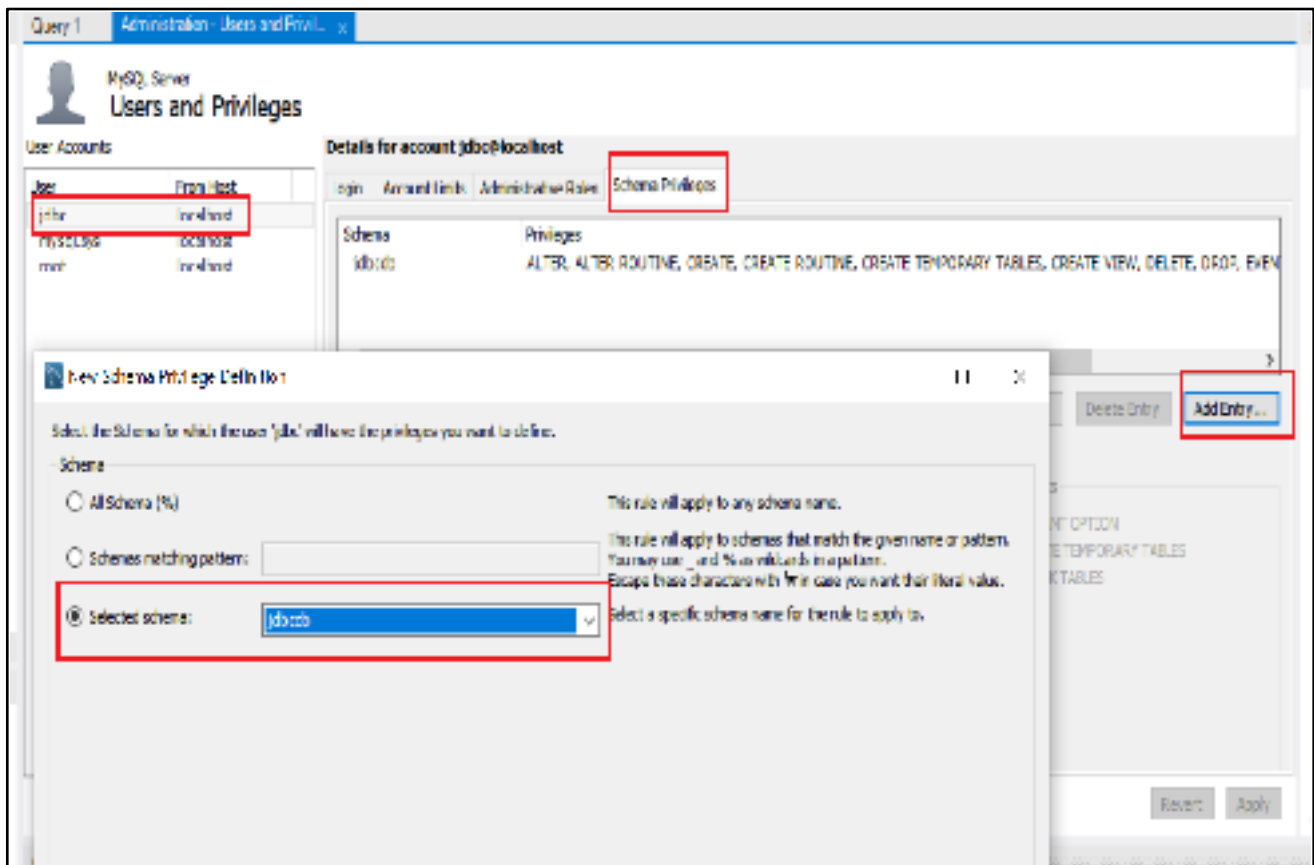
MySQL Workbench - 권한 부여

User : jdbc 선택

Schema Privileges 탭 선택

Add Entry... 버튼 선택

Selected schema : jdbcdb 선택



Select “ALL” 버튼 선택

Details for account jdbc@localhost

Login
Account Limits
Administrative Roles
Schema Privileges

Schema	Privileges
jdbcdb	ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, EVENT

Schema and Host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.

The user 'jdbc@localhost' will have the following access rights to the schema 'jdbcdb':

Object Rights

☒ SELECT
☒ INSERT
☒ UPDATE
☒ DELETE
☒ EXECUTE
☒ SHOW VIEW

DDL Rights

☒ CREATE
☒ ALTER
☒ REFERENCES
☒ INDEX
☒ CREATE VIEW
☒ CREATE ROUTINE
☒ ALTER ROUTINE
☒ EVENT
☒ DROP
☒ TRIGGER

Other Rights

☐ GRANT OPTION
☒ CREATE TEMPORARY TABLES
☒ LOCK TABLES

Unselect All
Select "ALL"

테이블 생성 : CREATE TABLE 문

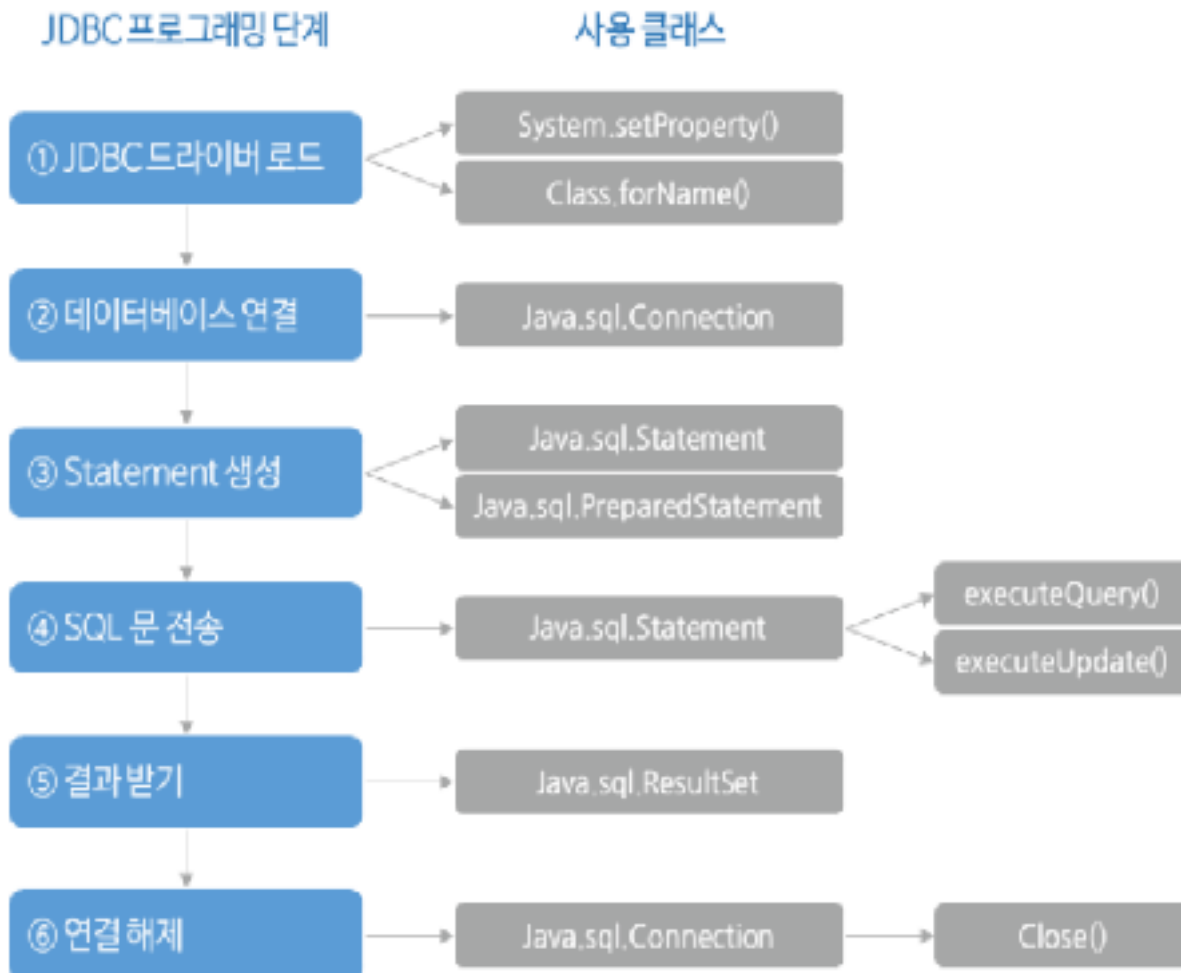
· 고객 테이블은 아이디, 이름, 나이, 등급, 직업, 적립금 컬럼으로 구성되고, 고객 아이디 컬럼이 기본키이다. 이름과 등급 컬럼은 값을 반드시 입력해야 하고, 적립금 컬럼은 값을 입력하지 않으면 0 이 기본으로 입력되도록 테이블을 생성해보자.

· SQL 스크립트

```
CREATE TABLE customer (  
    id VARCHAR(20) NOT NULL,  
    name VARCHAR(25) NOT NULL,  
    age INT,  
    grade VARCHAR(20) NOT NULL,  
    point INT DEFAULT 0,  
    PRIMARY KEY(id)  
);
```

id	name	age	grade	point
java1	일길동	10	silver	5
java2	이길동	20	gold	0
java3	삼길동	30	vip	5

JDBC API 로 작성된 Java Application 실행 프로세스



[출처] <https://o-jing.tistory.com/28>

Connection 객체 생성

- 데이터베이스 연결
- URL 형식 : `jdbc:mysql://MySQL Server IP:port/Schema_name`

```
import java.sql.Connection;
import java.sql.DriverManager;

public class DBConn {
    public static Connection getConnection() throws Exception {
        //1. MySQL JDBC 드라이버 로딩
        Class.forName("com.mysql.cj.jdbc.Driver");

        //2. MySQL 데이터베이스 연결(Connection)
        String url = "jdbc:mysql://localhost:3306/jdbcdadb";
        String user = "jdbc";
        String password = "jdbc1234";
        return DriverManager.getConnection(url, user, password);
    }
}
```

Statement 와 PreparedStatement 차이

- SQL 문을 실행하는 객체이다.

Statement

1. 단일 SQL 문을 수행할 때 속도가 빠르다.
2. 쿼리에 인자를 부여할 수 없다.
3. 매번 컴파일을 수행한다.

PreparedStatement

1. 쿼리에 인자를 부여할 수 있다.
2. 처음 프리컴파일 된 후, 이후에는 컴파일을 수행하지 않는다.
3. 동일한 SQL 문을 반복적으로 수행할 때 효율적이다.

고객 등록 - INSERT 문

```
Connection conn = null;
PreparedStatement pstmt = null;
try {
    // MySQL 데이터베이스 연결 객체 생성
    conn = DBConn.getConnection();

    // SQL문을 수행하기 위한 PreparedStatement 객체 생성
    sql = new StringBuffer();
    sql.append("INSERT INTO customer ( id, name, age, grade, point ) ");
    sql.append("VALUES ( ?, ?, ?, ?, ? )");
    pstmt = conn.prepareStatement(sql.toString());

    // ? (IN 파라미터)를 특정한 값으로 설정한다.
    // IN 파라미터 인덱스는 1 부터 시작한다.
    pstmt.setString(1, "java1");
    pstmt.setString(2, "일길동");
    pstmt.setInt(3, 10);
    pstmt.setString(4, "silver");
    pstmt.setInt(5, 5);

    // DML(INSERT, UPDATE, DELETE)을 수행한 후 데이터베이스에서 변경된 행의 수를 반환한다.
    int rowCount = pstmt.executeUpdate();
    if(rowCount == 1) {
        System.out.println("고객이 등록되었습니다.");
    }
}
```

```
} catch (Exception e) {  
    e.printStackTrace();  
}  
} finally {  
    // 연결 해제  
    try {  
        if(pstmt != null) pstmt.close();  
        if(conn != null) conn.close();  
    } catch (Exception e2) {  
        e2.printStackTrace();  
    }  
}
```

고객 목록 조회 - SELECT 문

- 고객 아이디 순으로 전체 고객 정보를 조회해보자.

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    // 1. Connection 객체 생성
    conn = DBConn.getConnection();

    // 2. SQL문 전송하기 위한 Statement 객체 생성
    stmt = conn.createStatement();

    //3. 쿼리 수행 후 결과에 대한 정보를 담고 있는 ResultSet 객체 생성
    StringBuffer sql = new StringBuffer();
    sql.append("SELECT id, name, age, grade, point ");
    sql.append("FROM customer ");
    sql.append("ORDER BY id ASC");
    rs = stmt.executeQuery(sql.toString());
}
```



```
while ( rs.next() ) {
    String id = rs.getString(1);
    String name = rs.getString(2);
    int age = rs.getInt(3);
    String grade = rs.getString(4);
    int point = rs.getInt(5);
    System.out.printf("아이디 : %s\t", id);
    System.out.printf("이름 : %s\t", name);
    System.out.printf("나이 : %d\t", age);
    System.out.printf("등급 : %s\t", grade);
    System.out.printf("포인트 : %d\n", point);
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {    //4. 연결 해제
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
```

고객 정보 변경 - UPDATE 문

- 아이디가 'java1' 인 고객의 등급을 'vip', 포인트는 10으로 변경해보자.

```
Connection conn = null;
PreparedStatement pstmt = null;
ResultSet rs = null;
try {
    conn = DBConn.getConnection();

    StringBuffer sql = new StringBuffer();
    sql.append("UPDATE customer      ");
    sql.append("SET grade = ?, point = ?  ");
    sql.append("WHERE id = ?");
    pstmt = conn.prepareStatement(sql.toString());

    pstmt.setString(1, "vip");
    pstmt.setInt(2, 10);
    pstmt.setString(3, "java1");

    int rowCount = pstmt.executeUpdate();
    if (rowCount == 1) {
        System.out.println("고객 정보가 업데이트 되었습니다.");
    }
} catch (Exception e1) {
    e1.printStackTrace();
} finally {
```

```
try {  
    if(rs != null) rs.close();  
    if(pstmt != null) pstmt.close();  
    if(conn != null) conn.close();  
} catch (Exception e2) {  
    e2.printStackTrace();  
}  
}
```

고객 정보 삭제 - DELETE 문

- 아이디가 'java1'인 고객의 정보를 삭제해보자.

```
Connection conn = null;
PreparedStatement pstmt = null;
ResultSet rs = null;
try {
    conn = DBConn.getConnection();
    StringBuffer sql = new StringBuffer();
    sql.append("DELETE FROM customer ");
    sql.append("WHERE id = ?");
    pstmt = conn.prepareStatement(sql.toString());
    pstmt.setString(1, "java1");
    int rowCount = pstmt.executeUpdate();
    if (rowCount == 1) {
        System.out.println("고객 정보가 삭제되었습니다.");
    }
} catch (Exception e1) {
    e1.printStackTrace();
} finally {
    try {
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
```

CallableStatement

- SQL 의 스토어드 프로시저(Stored Procedure)를 호출하기 위해 사용되는 인터페이스

※ Stored Procedure 란?

- SQL 문을 데이터베이스에 저장해 놓고 함수처럼 호출해서 사용하는 것이다.

※ JDBC Type (java.sql.Types)이란 ?

- MySQL JDBC Driver 는 JDBC Type 사용하여 MySQL 데이터 형식을 Java 프로그래밍 언어가 인식할 수 있는 형식으로 변환하며 그 반대 과정도 수행한다.

- <https://mycup.tistory.com/235> 사이트 참고할 것.

Stored Procedure 생성

- 스토어드 프로시저 이름 : retrieveCustomerList
- 고객 아이디 순으로 고객 정보 목록을 조회하는 프로시저를 만들어 보자.

```
CREATE PROCEDURE retrieveCustomerList ( )
```

```
BEGIN
```

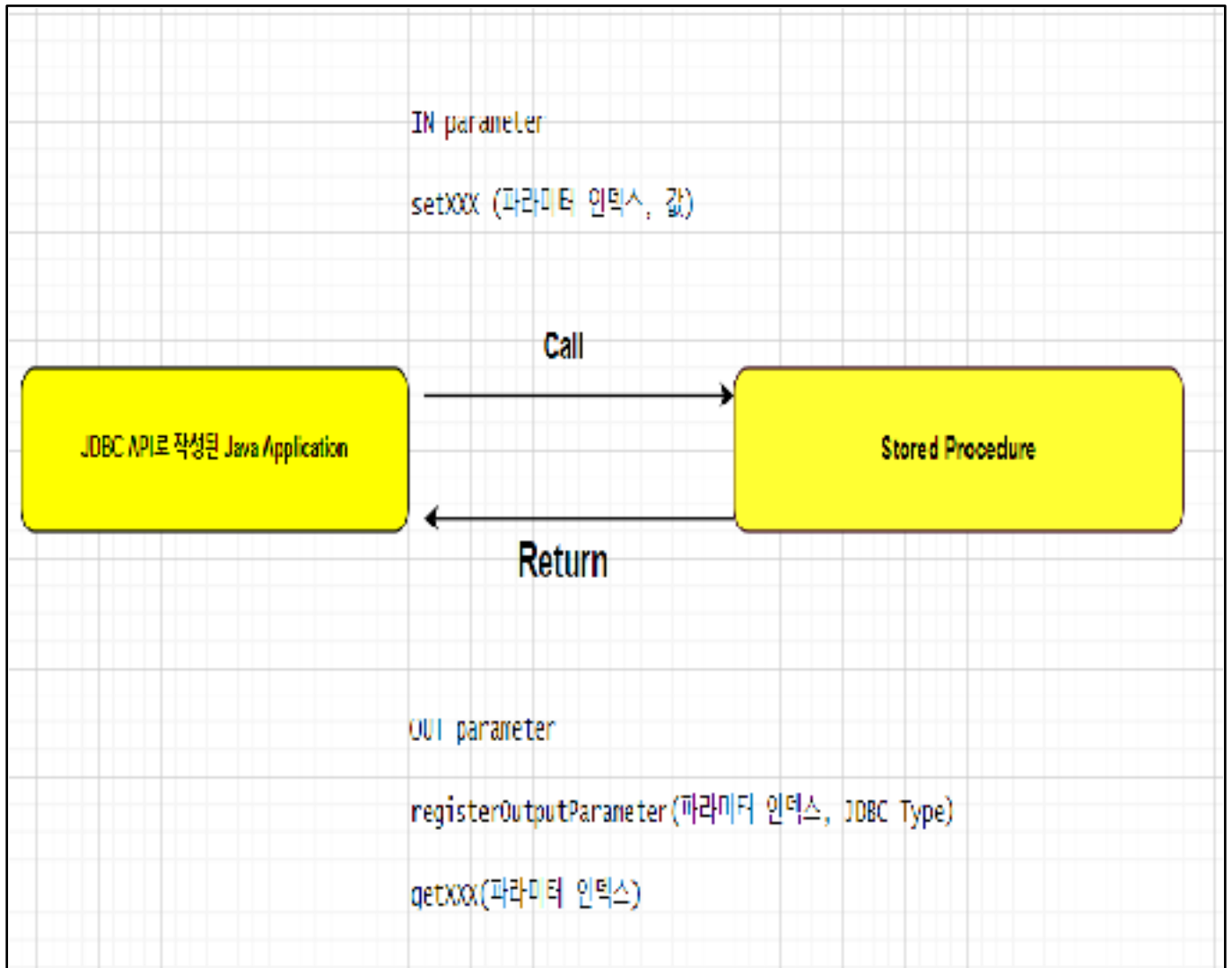
```
    SELECT id, name, age, grade, point
```

```
    FROM customer
```

```
    ORDER BY id asc;
```

```
END
```

Stored Procedure 동작 원리



Stored Procedure 호출

```
Connection conn = null;
CallableStatement cstmt = null;
ResultSet rs = null;
try {
    // 1. Connection 객체 생성
    conn = DBConn.getConnection();
    //2. 스토어드 프로시저를 호출할 CallableStatement 객체 생성
    cstmt = conn.prepareCall("Call retrieveCustomerList( )");
    rs = cstmt.executeQuery();
    ArrayList<CustomerVO> customerList = new ArrayList<CustomerVO>();
    while (rs.next()) {
        String id = rs.getString(1);
        String name = rs.getString(2);
        int age = rs.getInt(3);
        String grade = rs.getString(4);
        int point = rs.getInt(5);
        CustomerVO customer = new CustomerVO(id, name, age, grade, point);
        customerList.add(customer);
    }
    if (customerList.isEmpty()) {
        System.out.println("고객정보가 존재하지 않습니다.");
    } else {
        for(CustomerVO customer : customerList) {
            System.out.println(customer.toString());
        }
    }
}
```



```
} catch (Exception e) {  
    e.printStackTrace();  
}  
} finally {  
    try {  
        if(rs != null) rs.close();  
        if(cstmt != null) cstmt.close();  
        if(conn != null) conn.close();  
    } catch (Exception e2) {  
        e2.printStackTrace();  
    }  
}
```

Stored Procedure 생성

- 스토어드 프로시저 이름 : retrieveCustomerById
- 고객 아이디에 해당하는 고객 이름과 나이를 조회하는 스토어드 프로시저를 생성해보자.

```
CREATE PROCEDURE retrieveCustomerById (  
    IN v_id VARCHAR(20),  
    OUT v_name VARCHAR(25),  
    OUT v_age INT  
)  
  
BEGIN  
    SELECT name, age  
    INTO v_name, v_age  
    FROM customer  
    WHERE id = v_id;  
  
END
```

-----MySQL Workbench 실행 -----

```
CALL retrieveCustomerById('java2', @name, @age);  
select @name;  
select @age;
```

Stored Procedure 호출

```
Connection conn = null;
CallableStatement cstmt = null;
try {
    // 1. Connection 객체 생성
    conn = DBConn.getConnection();

    //2. 스토어드 프로시저를 호출할 CallableStatement 객체 생성
    cstmt = conn.prepareCall("Call retrieveCustomerById (?, ?, ?)");

    //3. IN, OUT 파라미터 설정
    // IN paramter : setXXX
    // OUT parameter : registerOutParameter(parameter index, JDBC Type)
    cstmt.setString(1, "java2"); // IN parameter
    cstmt.registerOutParameter(2, Types.VARCHAR); // OUT parameter
    cstmt.registerOutParameter(3, Types.INTEGER); // OUT parameter

    //4. 스토어드 프로시저 호출
    cstmt.execute();

    // OUT parameter : 프로시저 실행 후 호출한 프로그램으로 값을 반환한다.
    //5. 반환된 값을 받아서 처리한다.
    String name = cstmt.getString(2);
    int age = cstmt.getInt(3);
    System.out.println("이름 : " + name);
    System.out.println("나이 : " + age);
}
```

```
} catch (Exception e) {  
    e.printStackTrace();  
}  
} finally {  
    try {  
        if(rs != null) rs.close();  
        if(cstmt != null) cstmt.close();  
        if(conn != null) conn.close();  
    } catch (Exception e2) {  
        e2.printStackTrace();  
    }  
}  
}
```

트랜잭션 (Transaction)

※ 트랜잭션이란?

- 데이터베이스에서 한번에 처리해야 할 논리적인 작업 단위이다.

※ 트랜잭션의 특성 (ACID)

1. 원자성(Atomicity)

트랜잭션에 포함된 명령들은 모두 수행되거나, 모두 수행 안되어야 한다. 즉 어느 명령은 실행되고 어느 명령은 실행되지 않으면 안된다.

2. 일관성(Consistency)

트랜잭션이 완료된 뒤에는 일관적인 상태에 있어야 한다. 트랜잭션의 영향이 한 방향으로만 전달되어야 한다.

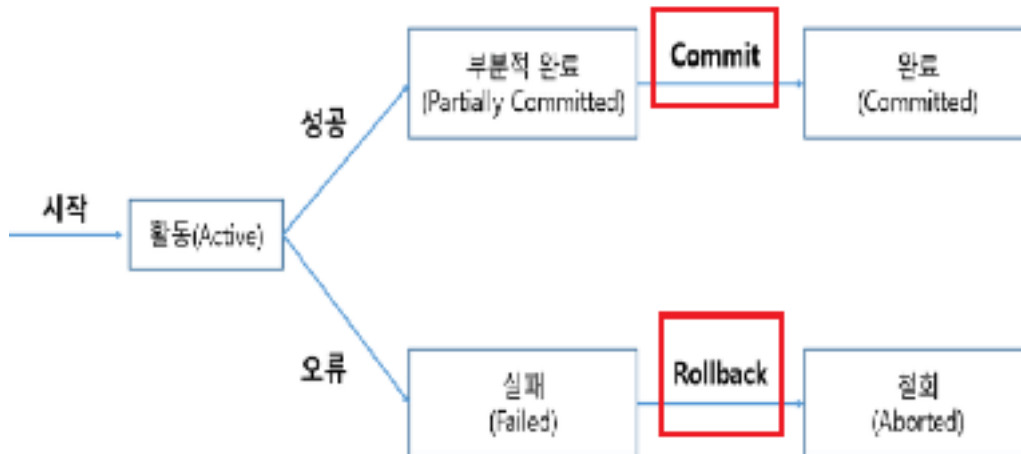
3. 고립성(Isolation)

트랜잭션은 다른 트랜잭션과 독립적으로 실행되는 것 처럼 보여야 한다. 트랜잭션의 부분상태를 다른 트랜잭션에 제공해서는 안된다.

4. 지속성(Durability)

트랜잭션의 결과는 반드시 데이터베이스에 반영되어야 한다.

※ 트랜잭션 상태



출처 : <https://limkydev.tistory.com/100>

활동 초기상태
트랜잭션이 실행중인 상태

실패 트랜잭션이 실행중에 오류가 발생하여 중단된 상태

철회 (Rollback) 트랜잭션이 비정상적으로 종료되어 롤백 연산을 수행한 상태
데이터베이스에 대한 갱신 작업이 취소된다

부분적 완료 트랜잭션이 마지막 연산까지 실행했지만, 커밋 연산이 실행되기 직전의 상태

완료 (Commit) 트랜잭션이 성공적으로 종료되어 커밋 연산을 실행한 후의 상태
데이터베이스에 변경내용을 저장한다

JDBC API 에서 트랜잭션 처리

※ 트랜잭션을 위한 메소드

- Connection setAutoCommit(boolean autoCommit)
기본값이 true (자동 커밋) 로 설정되어 있으므로 트랜잭션을 처리하려면 autoCommit 을 false 로 설정한다.
- commit() : 트랜잭션(Transaction)의 commit 을 수행한다.
- rollback() : 트랜잭션(Transaction)의 rollback 을 수행한다.

```
Connection conn = null;
boolean isSuccess= false;
try {
    conn = DBConn.getConnection();

    // 트랜잭션 시작
    conn.setAutoCommit ( false ) ;
    SQL 문 1;
    SQL 문 2;
    SQL 문 3;
    ...
    isSuccess = true;

} ...finally {
    try {
        if ( isSuccess ) {
            // 올바르게 로직이 수행되면 커밋
            conn.commit( );
        } else {
            // 에러가 발생하면 롤백
            conn.rollback ( ) ;
        }
    } catch(Exception ex) {
        ex.printStackTrace( );
    }
}
```


트랜잭션 예제 : 고객정보 등록 배치 처리

```
List<Customer> customerList = new ArrayList<Customer>();  
customerList.add(new Customer("mysql1", "일길동", 10, "silver", 0));  
customerList.add(new Customer("mysql2", "이길동", 20, "gold", 10));  
customerList.add(new Customer("mysql3", "삼길동", 30, "silver", 5));  
customerList.add(new Customer("mysql4", "사길동", 40, "vip", 0));  
customerList.add(new Customer("mysql5", "오길동", 50, "silver", 0));
```

```
Connection conn = null;
PreparedStatement pstmt = null;
boolean isSuccess = false;
try {
    conn = DBConn.getConnection();
    // 트랜잭션 시작
    conn.setAutoCommit(false);

    //배치 처리 코드
    StringBuffer sql = new StringBuffer();
    sql.append("INSERT INTO customer ( id, name, age, grade, point ) ");
    sql.append("VALUES ( ?, ?, ?, ?, ? ) ");
    pstmt = conn.prepareStatement(sql.toString());

    for(Customer customer : customerList) {
        pstmt.setString(1, customer.getId());
        pstmt.setString(2, customer.getName());
        pstmt.setInt(3, customer.getAge());
        pstmt.setString(4, customer.getGrade());
        pstmt.setInt(5, customer.getPoint());
        pstmt.addBatch();
        pstmt.clearParameters(); // IN 파라미터 초기화
    }
    // 데이터베이스에서 일괄처리를 수행한다.
    pstmt.executeBatch();
    isSuccess = true;
} catch (Exception e) {
    e.printStackTrace();
}
```

```
} finally {  
    try {  
        if(isSuccess) {  
            System.out.println("고객정보가 등록되었습니다.");  
            // 올바르게 로직이 수행되면 커밋  
            conn.commit();  
        } else {  
            // 에러가 발생하면 롤백  
            conn.rollback();  
        }  
        if(pstmt != null) pstmt.close();  
        if(conn != null) conn.close();  
    } catch (Exception e2) {  
        e2.printStackTrace();  
    }  
}  
  
} // end of finally clause
```