

MySQL 기반의 SQL과 JDBC



김정현

객체 생성

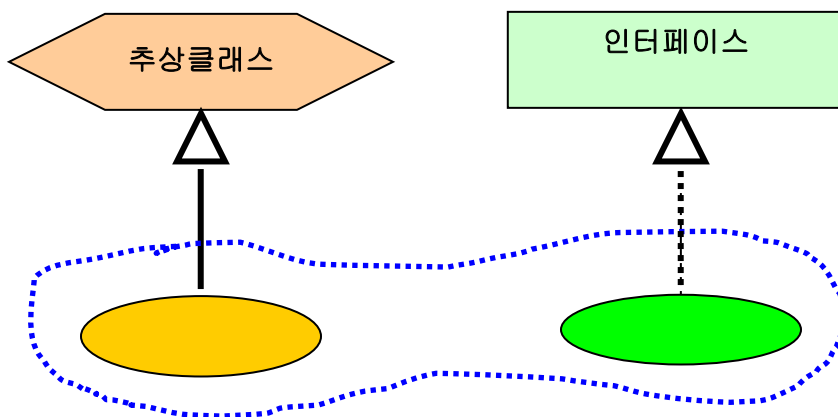
일반 클래스의 객체 생성은 new 와 생성자 메서드를 사용한다. new 와 생성자 메서드를 사용하는 객체 생성은 Java 구문에서 지원되는 일반적인 객체 생성 방법이다.

그러나 클래스들 중에는 이 일반적인 객체 생성을 통해서 객체를 생성할 수 없는 것들도 있다. 이런 경우에는 해당 클래스에서 제공되는 static 형 메서드를 호출하여 객체 생성을 대신 할 수 있다.

그렇다면 왜 생성자 메서드를 통해서 객체를 생성하지 않고 해당 클래스에서 제공되는 static 형 메서드를 호출하여 객체 생성을 대신하도록 하는 것일까?

- 여러 이유로 자식 클래스의 객체 생성을 대신 하여 사용되도록 하려는 경우
- 클래스의 객체 생성을 여러 번 하더라도 하나의 객체만을 생성하려는 경우

추상 클래스와 인터페이스의 경우에는 new 와 생성자 메서드를 사용하여 객체를 생성할 수 없다. 자식 클래스를 만들어 대신 객체 생성하여 사용한다.



JDBC 의 경우 대부분의 API 가 인터페이스이다.

Connection, Statement, ResultSet, PreparedStatement

대부분의 JDBC 프로그램에서는 위의 API 들에서 제공되는 메서드를 호출해야 한다.

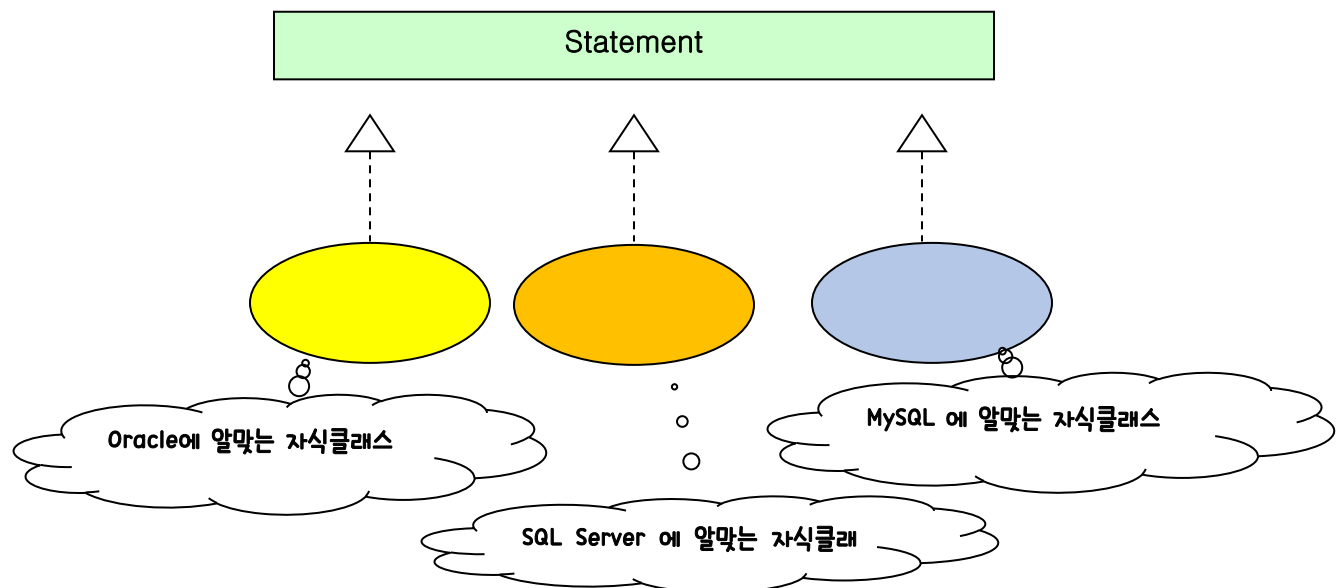
예를 들어.....

Connection : createStatement(), getMetaData().....

Statement : executeQuery(), executeUpdate().....

ResultSet : next(), getXXX().....

이 API 들의 객체를 생성하기 위해서는 이 API 들을 상속하여 구현하고 있는 자식 클래스가 필요한데 그 자식 클래스들을 바로 JDBC 드라이버가 제공한다. JDBC 드라이버라는 것은 JDBC 에서 인터페이스를 설계되어 있는 API 들의 자식 클래스들을 제공하는 프로그램이라고 할 수 있다. 즉 어떠한 DB 서버용 드라이버냐에 따라서 제공되는 자식 클래스들의 수행 코드가 다르게 만들어져 있는 것이다. JDBC API 내에서는 JDBC 드라이버가 제공하는 각 인터페이스들의 자식 클래스가 어떠한 이름의 클래스인지 모르고도 프로그래밍 가능하도록 팩토리 메서드라는 것을 제공하고 있다. 일반 메서드로서 다른 클래스의 객체생성을 대신해주는 메서드를 팩토리 메서드라고 한다.



JDBC는 Java 프로그램에서 DB 서버를 접속하여 데이터를 처리하는 기능을 구현하고자 할 때 사용되는 Java 기술

- JDBC 기술의 구성

- JDBC API (java.sql, javax.sql) -> 공통적(모든 DB 서버에 대해)
- JDBC Driver -> DB 서버마다 달라진다.

- 주요 JDBC API

[인터페이스]

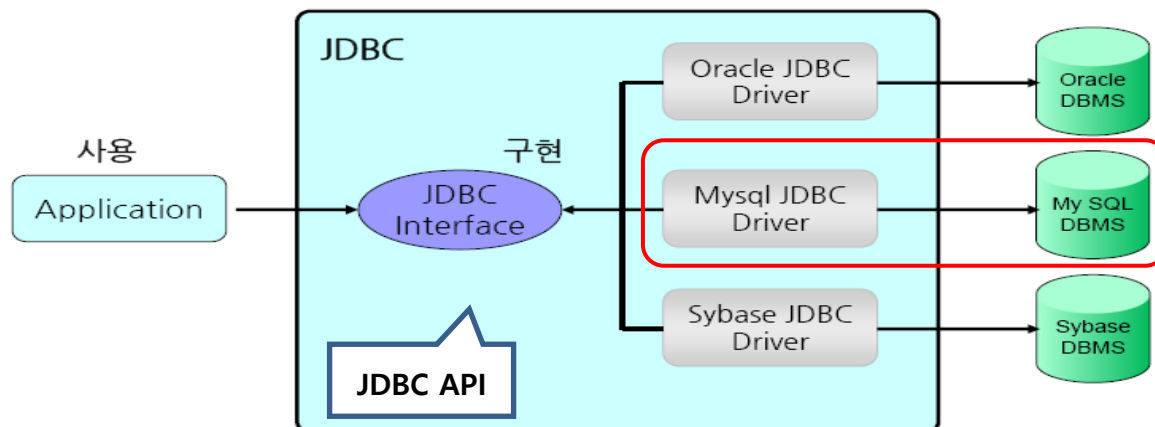
Connection, Statement, PreparedStatement, ResultSet
DatabaseMetaData, ResultSetMetaData

[클래스]

DriverManager, Date, Time, Timestamp

❖ JDBC의 개념

- 자바 언어에서 Database에 접근할 수 있게 해주는 Programming API



[MySQL 용 JDBC 드라이버 설치]

The screenshot shows the MySQL Product Archives page for MySQL Connector/J (Archived Versions). A yellow warning box states: "Please note that these are old versions. New releases will have recent bug fixes and features! To download the latest release of MySQL Connector/J, please visit [MySQL Downloads](#)." Below this, there are two dropdown menus: "Product Version:" set to "8.0.17" and "Operating System:" set to "Platform Independent". Below the dropdowns is a table of download options.

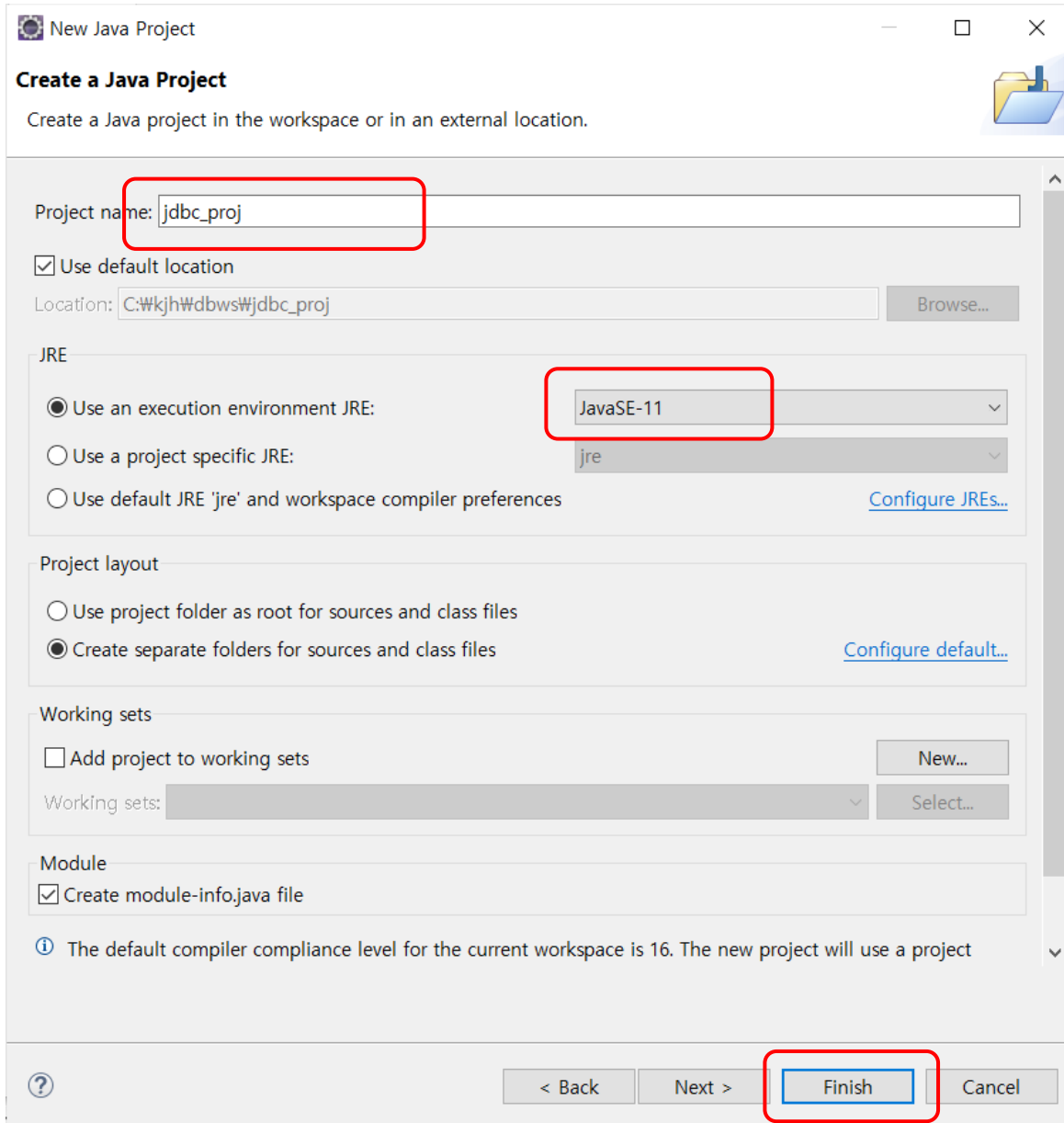
Platform Independent (Architecture Independent), Compressed TAR Archive	Jun 7, 2019	3.6M	Download
(mysql-connector-java-8.0.17.tar.gz)		MD5: f4a7b4ca814488d15a73f71a93df3f9c Signature	
Platform Independent (Architecture Independent), ZIP Archive	Jun 7, 2019	4.3M	Download
(mysql-connector-java-8.0.17.zip)		MD5: 479fc6d22fd43f01ae93cacf9dbb9e82 Signature	

다운로드된 zip 파일의 압축을 풀어서 mysql-connectoe-java-8.0.17.jar 파일만 c:\Program Files\MySQL 폴더에 저장한다.

The screenshot shows a Windows File Explorer window titled "MySQL". The address bar shows the path: "로컬 디스크 (C:) > Program Files > MySQL". The left sidebar shows a list of folders: "kjh", "backup", "dbws", "DJANGO", "eclipse", "employee", and "PYDATA". The main pane shows a list of files and folders:

이름	수정된 날짜	유형	크기
MySQL Server 8.0	2021-12-05 오전 8:03	파일 폴더	
MySQL Workbench 8.0 CE	2021-12-05 오전 8:23	파일 폴더	
mysql-connector-java-8.0.17.jar	2019-06-07 오전 9:49	Executable Jar File	2,268KB

[Java Project 생성]

 New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: jdbc_proj

☒ Use default location

Location: C:\kjh\workspace\jdbc_proj [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-11

☐ Use a project specific JRE: jre

☐ Use default JRE 'jre' and workspace compiler preferences [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)


Working sets


☐ Add project to working sets [New...](#)

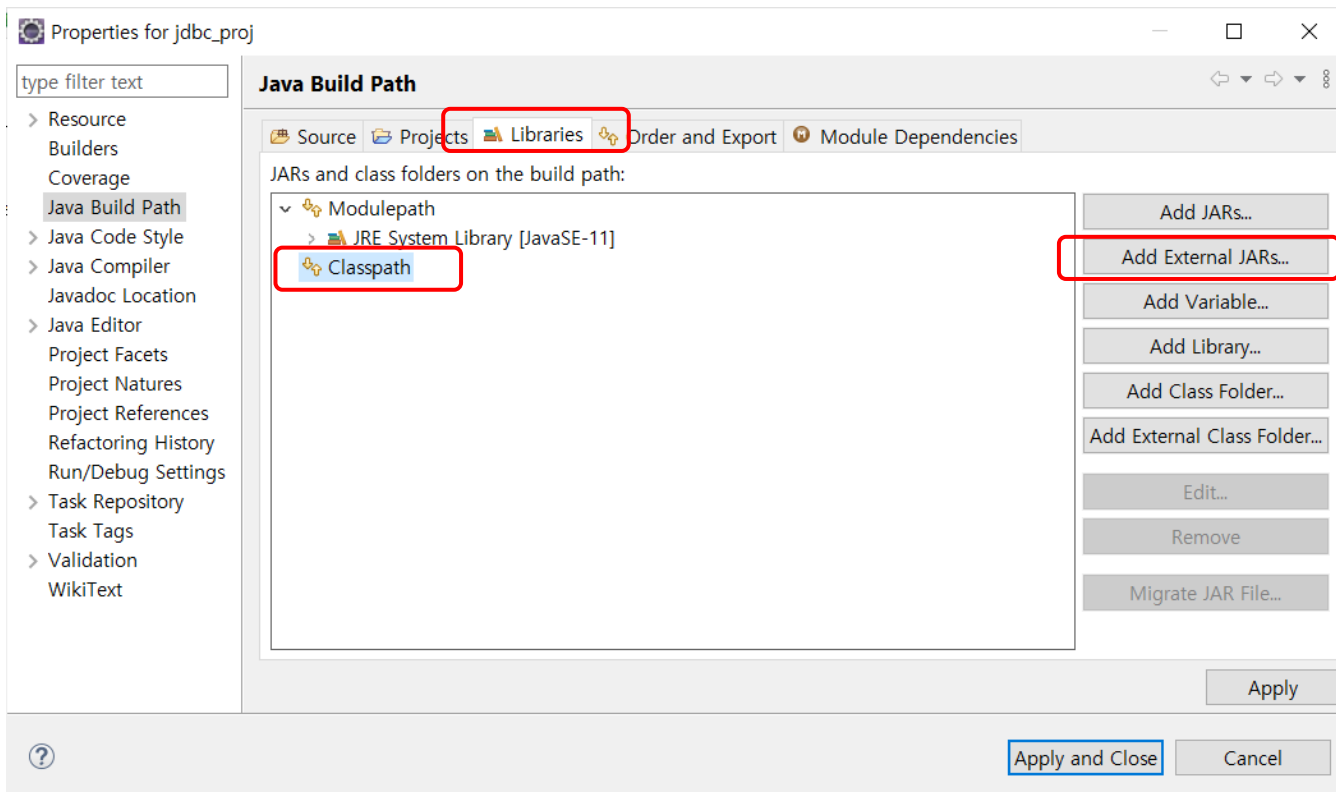
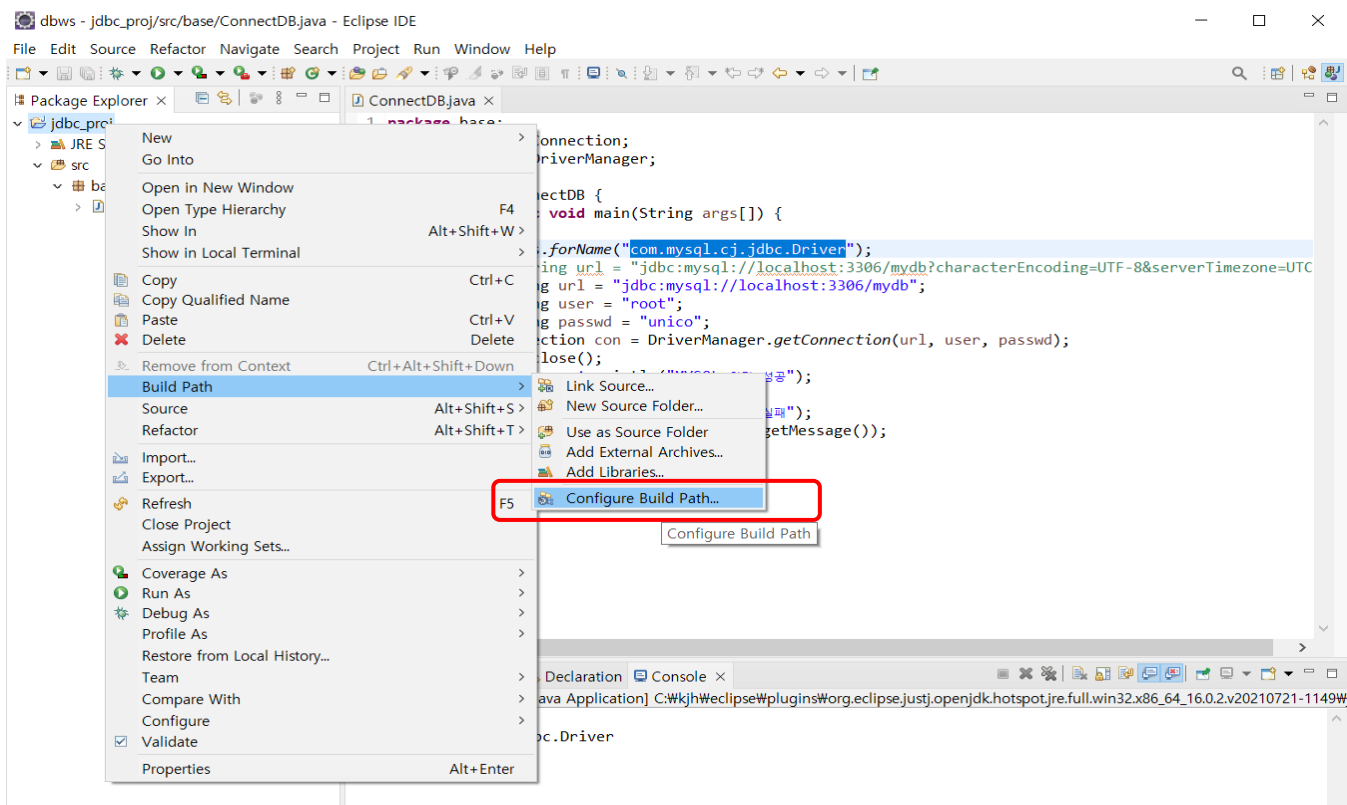
Working sets: [Select...](#)

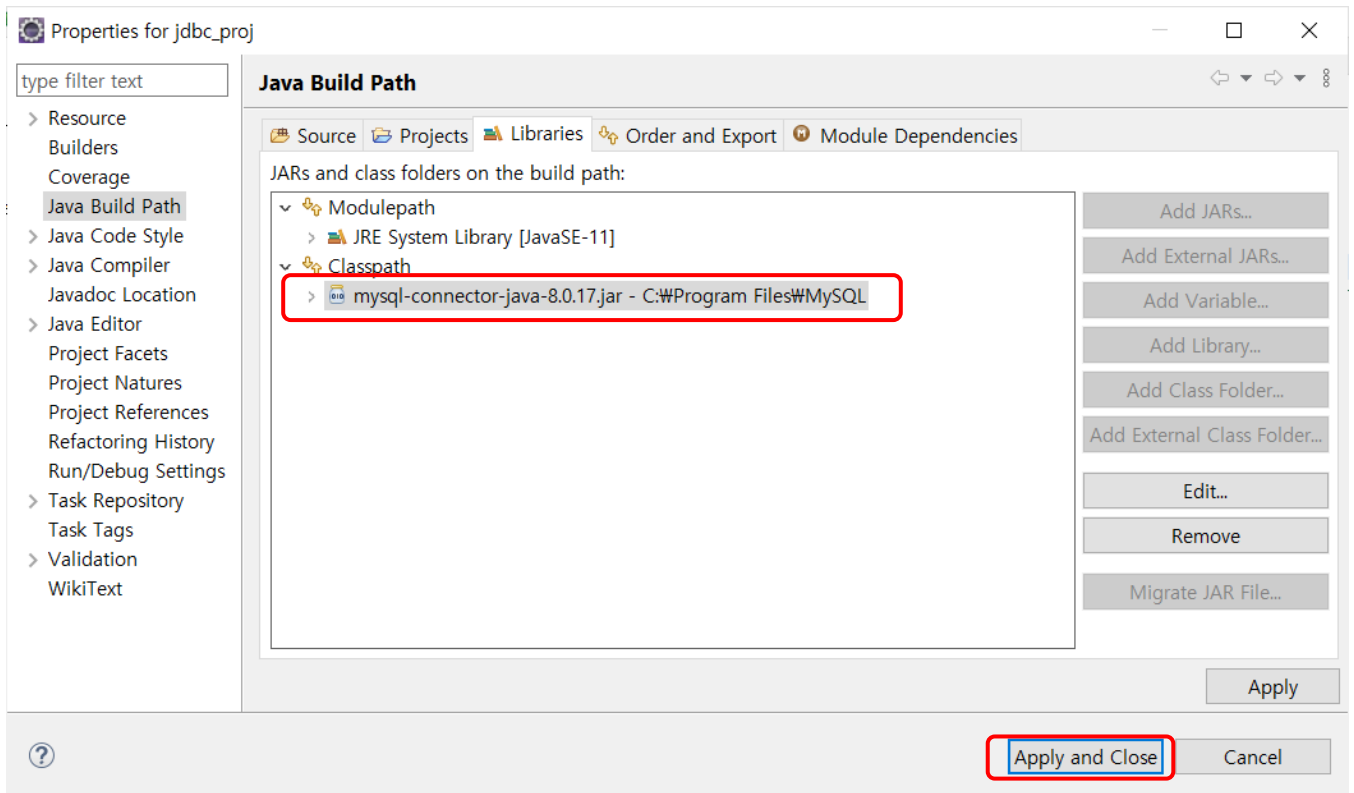
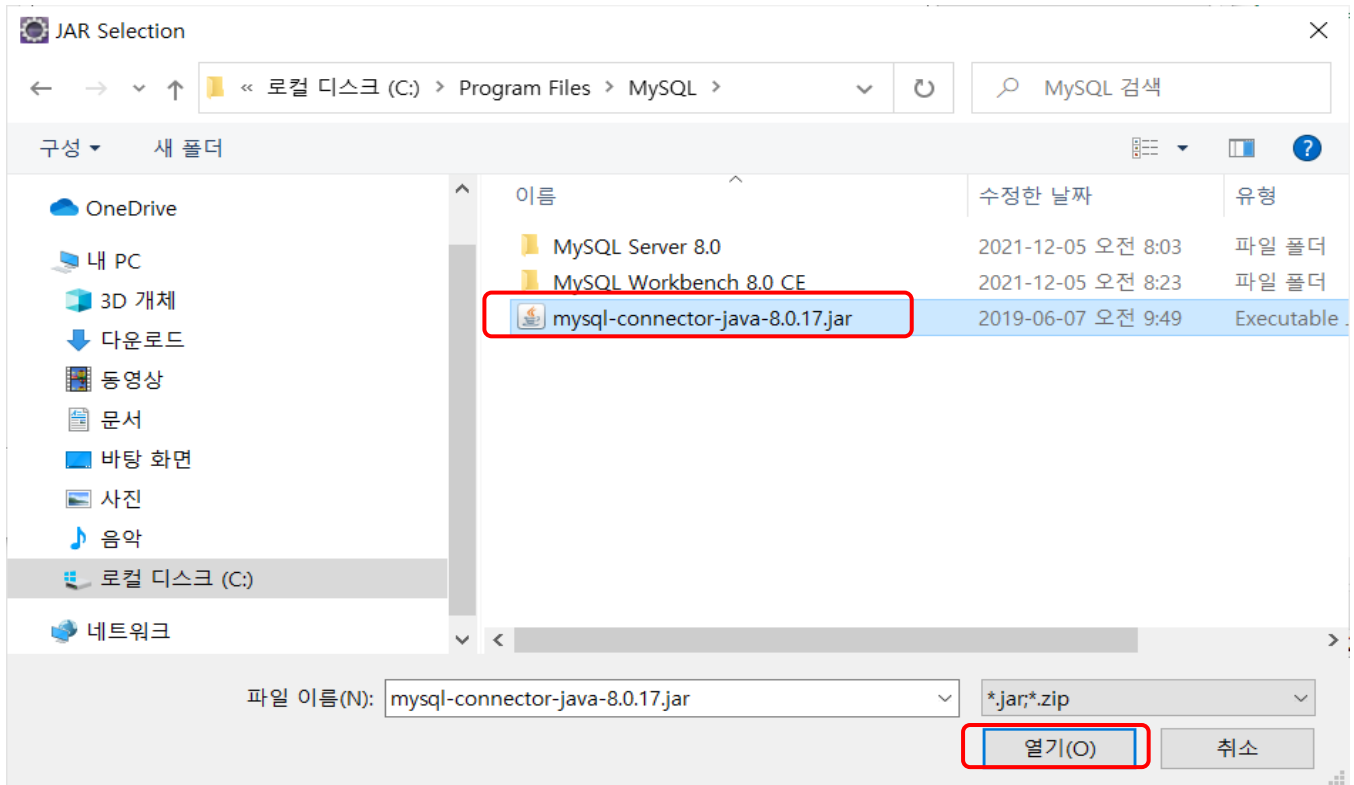
Module

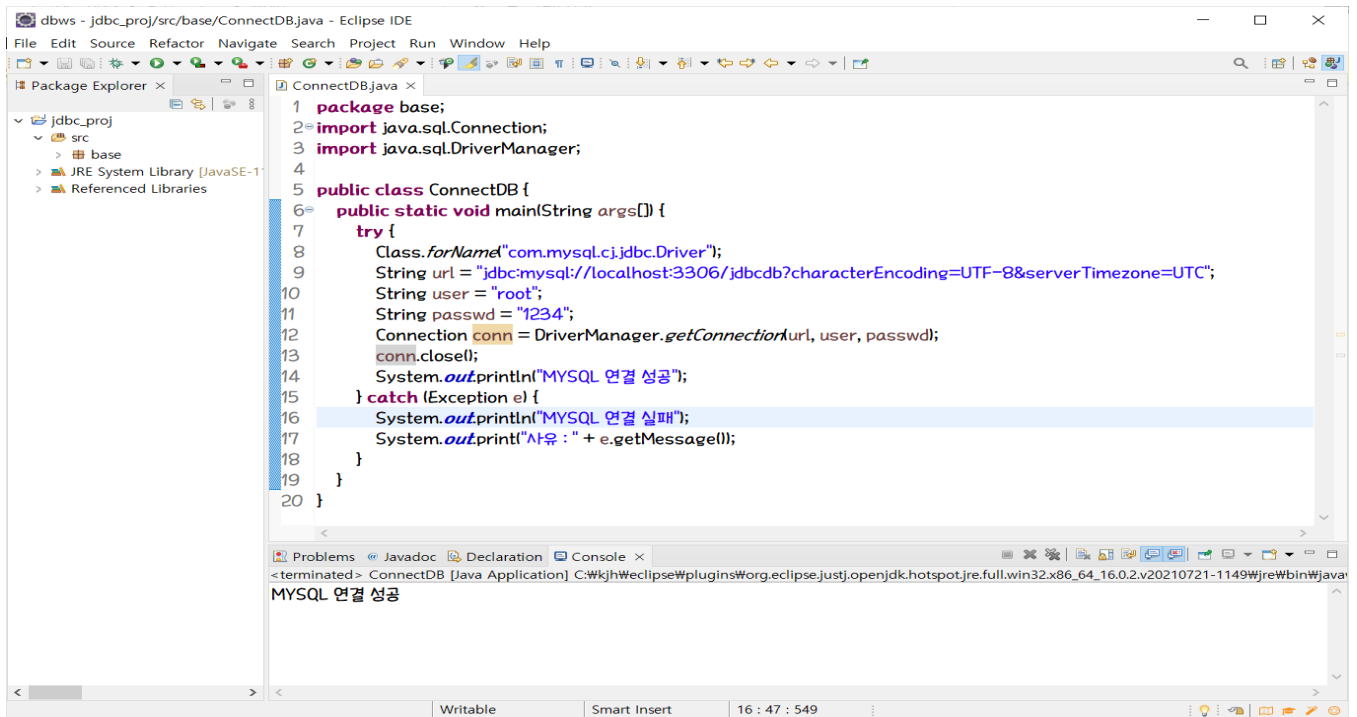
☒ Create module-info.java file

 The default compiler compliance level for the current workspace is 16. The new project will use a project

 [< Back](#) [Next >](#) **Finish** [Cancel](#)







[예제 1]

```
package level1;
```

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

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

```
public class ConnectDB {
```

```
    public static void main(String args[]) {
```

```
        try {
```

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

```
            String url =
```

```
                "jdbc:mysql://localhost:3306/jdbcd?characterEncoding=UTF-8&serverTimezone=UTC";
```

```
            String user = "root";
```

```
            String passwd = "1234";
```

```
            Connection conn = DriverManager.getConnection(url, user, passwd);
```

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

```
            System.out.println("MYSQL 연결 성공");
```

```
        } catch (Exception e) {
```

```
            System.out.println("MYSQL 연결 실패");
```

```
            System.out.println("사유 : " + e.getMessage());
```

```
        }
```

```
    }
```

```
}
```

```
jdbc:mysql://[hostname][:port]/dbname[?param1=value1][&param2=value2]...
```

Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language. This API includes a framework whereby different drivers can be installed dynamically to access different data sources. Although the JDBC™ API is mainly geared to passing SQL statements to a database, it provides for reading and writing data from any data source with a tabular format. The reader/writer facility, available through the `javax.sql.RowSet` group of interfaces, can be customized to use and update data from a spread sheet, flat file, or any other tabular data source.

What the JDBC™ 4.3 API Includes

The JDBC™ 4.3 API includes both the **java.sql** package, referred to as the JDBC core API, and the **javax.sql** package, referred to as the JDBC Optional Package API. This complete JDBC API is included in the Java™ Standard Edition (Java SE™), version 7. The **javax.sql** package extends the functionality of the JDBC API from a client-side API to a server-side API, and it is an essential part of the Java™ Enterprise Edition (Java EE™) technology.

Versions

The JDBC 4.3 API incorporates all of the previous JDBC API versions:

- The JDBC 4.2 API
 - The JDBC 4.1 API
 - The JDBC 4.0 API
 - The JDBC 3.0 API
 - The JDBC 2.1 core API
 - The JDBC 2.0 Optional Package API
- Note that the JDBC 2.1 core API and the JDBC 2.0 Optional Package API together are referred to as the JDBC 2.0 API.)
- The JDBC 1.2 API
 - The JDBC 1.0 API

Module java.sql

Defines the JDBC API.

Module Graph:

```

graph TD
    java_sql[java.sql] --> java_logging[java.logging]
    java_sql --> java_xml[java.xml]
    java_sql --> java_transaction_xa[java.transaction.xa]
    java_logging --> java_base[java.base]
    java_xml --> java_base
    java_transaction_xa --> java_base
  
```

Packages

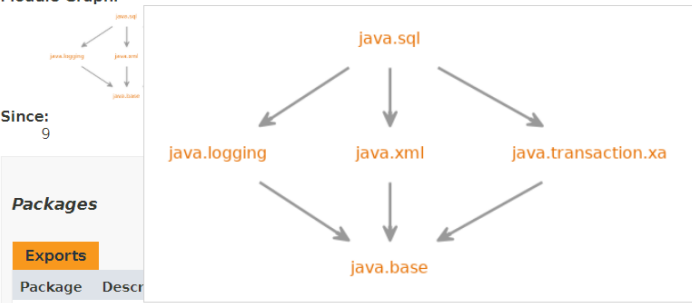
Package	Description
java.sql	Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language.
javax.sql	Provides the API for server side data source access and processing from the Java™ programming language.

Indirect Exports

From	Packages
java.logging	java.util.logging
java.transaction.xa	javax.transaction.xa
java.xml	javax.xml.catalog javax.xml.datatype javax.xml.namespace javax.xml.parsers javax.xml.stream javax.xml.stream.events javax.xml.stream.util javax.xml.transform javax.xml.transform.dom javax.xml.transform.sax javax.xml.transform.stax javax.xml.validation javax.xml.xpath org.w3c.dom.org.w3c.dom.bootstrap org.w3c.dom.events org.w3c.dom.ls org.w3c.dom.ranges org.w3c.dom.traversal org.w3c.dom.views org.xml.sax org.xml.sax.ext org.xml.sax.helpers

Defines the JDBC API.

Module Graph:



Since:

Packages

Exports

Package	Descr
java.sql	Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language.
javax.sql	Provides the API for server side data source access and processing from the Java™ programming language.
Indirect Exports	
From	Packages
java.logging	java.util.logging
java.transaction.xa	javax.transaction.xa
java.xml	javax.xml.catalog javax.xml.datatype javax.xml.namespace javax.xml.parsers javax.xml.stream javax.xml.stream.events javax.xml.stream.util javax.xml.transform javax.xml.transform.dom javax.xml.transform.sax javax.xml.transform.stax javax.xml.transform.stream javax.xml.validation javax.xml.xpath org.w3c.dom org.w3c.dom.bootstrap org.w3c.dom.events org.w3c.dom.ls org.w3c.dom.ranges org.w3c.dom.traversal org.w3c.dom.views org.xml.sax org.xml.sax.ext org.xml.sax.helpers

<https://onlinehelp.unitedplanet.com/apidocs/jdk11/api/java.sql/module-graph.png>

java.sql (Java SE 11 & JDK 11) x +

onlinehelp.unitedplanet.com/apidocs/jdk11/api/java.sql/java/sql/package-summary.html

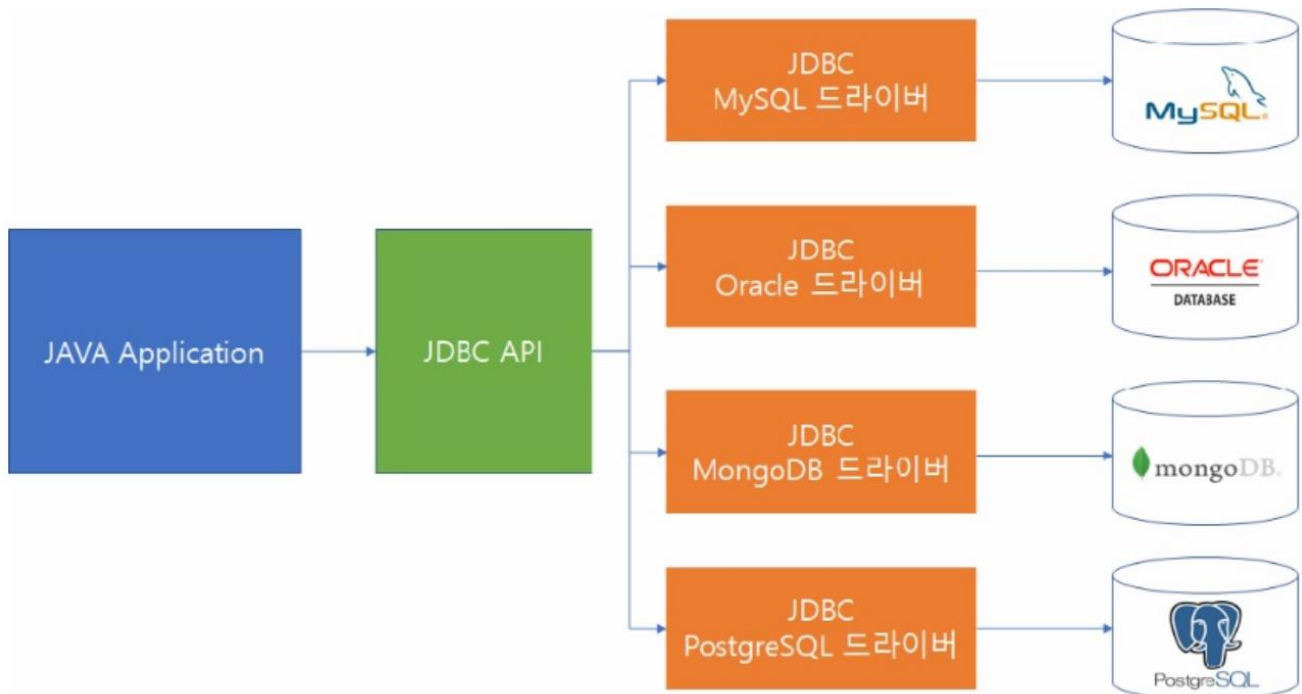
OVERVIEW MODULE **PACKAGE** CLASS USE TREE DEPRECATED INDEX HELP

Java SE 11 & JDK 11

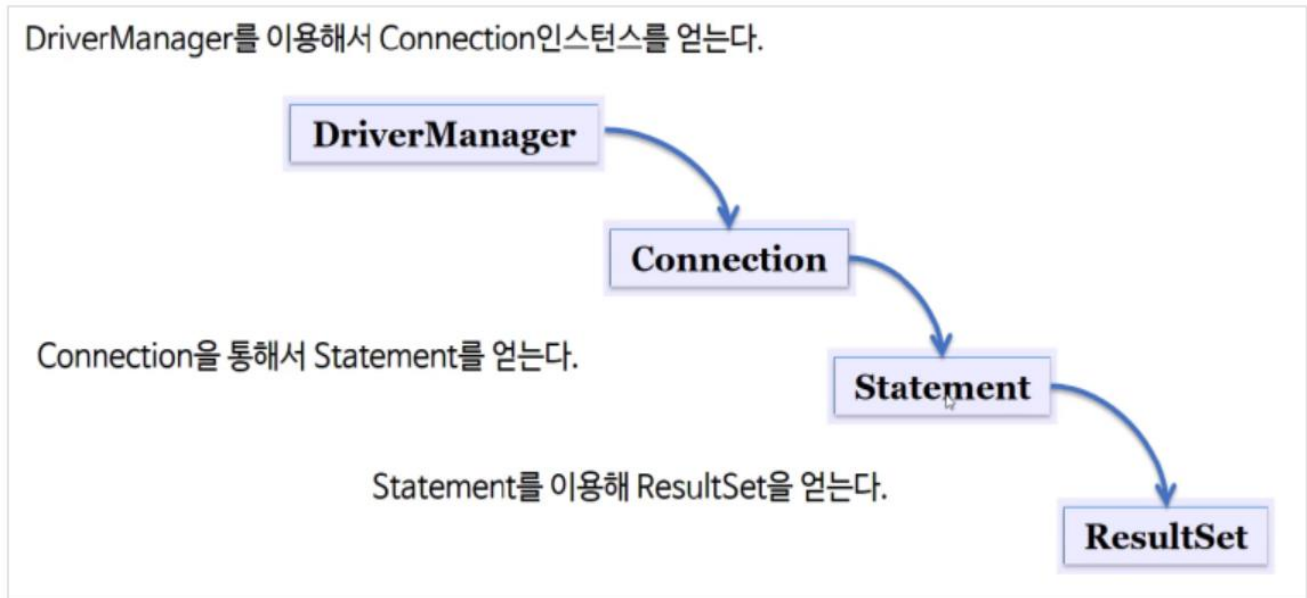
ALL CLASSES SEARCH:

The java.sql package contains API for the following:

- Making a connection with a database via the DriverManager facility
 - DriverManager class -- makes a connection with a driver
 - SQLPermission class -- provides permission when code running within a Security Manager, such as an applet, attempts to set up a logging stream through the DriverManager
 - Driver interface -- provides the API for registering and connecting drivers based on JDBC technology ("JDBC drivers"); generally used only by the DriverManager class
 - DriverPropertyInfo class -- provides properties for a JDBC driver; not used by the general user
- Sending SQL statements to a database
 - Statement -- used to send basic SQL statements
 - PreparedStatement -- used to send prepared statements or basic SQL statements (derived from Statement)
 - CallableStatement -- used to call database stored procedures (derived from PreparedStatement)
 - Connection interface -- provides methods for creating statements and managing connections and their properties
 - Savepoint -- provides savepoints in a transaction
- Retrieving and updating the results of a query
 - ResultSet interface
- Standard mappings for SQL types to classes and interfaces in the Java programming language
 - Array interface -- mapping for SQL ARRAY
 - Blob interface -- mapping for SQL BLOB
 - Clob interface -- mapping for SQL CLOB
 - Date class -- mapping for SQL DATE
 - NClob interface -- mapping for SQL NCLOB
 - Ref interface -- mapping for SQL REF
 - RowId interface -- mapping for SQL ROWID
 - Struct interface -- mapping for SQL STRUCT
 - SQLXML interface -- mapping for SQL XML
 - Time class -- mapping for SQL TIME
 - Timestamp class -- mapping for SQL TIMESTAMP
 - Types class -- provides constants for SQL types
- Custom mapping an SQL user-defined type (UDT) to a class in the Java programming language
 - SQLData interface -- specifies the mapping of a UDT to an instance of this class
 - SQLInput interface -- provides methods for reading UDT attributes from a stream
 - SQLOutput interface -- provides methods for writing UDT attributes back to a stream
- Metadata
 - DatabaseMetaData interface -- provides information about the database
 - ResultSetMetaData interface -- provides information about the columns of a ResultSet object
 - ParameterMetaData interface -- provides information about the parameters to PreparedStatement commands
- Exceptions
 - SQLException -- thrown by most methods when there is a problem accessing data and by some methods for other reasons
 - SQLWarning -- thrown to indicate a warning



[데이터 읽기]



1. Driver 로드

DriverManager로 어떤 DB를 사용할 것인지 드라이버를 로드한다.

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

각 DB마다 고유의 드라이버 이름이 있다.

2. Connection 얻기

DB를 결정 후, 연결을 위해 연결 정보(DB 서버 url, ID, PW등)를 입력한다.

```
String url =
```

```
"jdbc:mysql://localhost:3306/sqlldb?characterEncoding=UTF-8&serverTimezone=UTC";
```

```
String user = "root";
```

```
String passwd = "1234";
```

```
Connection conn = DriverManager.getConnection(url, user, passwd);
```

3. Statement 작성

DB 서버에 SQL 명령을 전달하여 실행시키기 위한 객체를 생성한다.

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

4. SELECT 명령을 실행하고 실행 결과를 ResultSet에 담기

SELECT 명령을 실행한 결과는 JDBC 드라이버가 ResultSet이라는 객체로 반환한다.

```
ResultSet rs = stmt.executeQuery("SELECT ename, sal FROM emp");
```

5. ResultSet 객체에서 값 꺼내오기

```
rs.next()
```

```
String name = rs.getString("ename")
```

```
int salary = rs.getInt("sal")
```

6. 커넥션 반환하기

반환 작업은 사용했던 객체를 역순으로 닫는다.

```
rs.close();
```

```
stmt.close();
```

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

- ResultSet

ResultSet은 설명했듯 Query의 실행 결과를 담는 객체로서 최초에는 결과 집합에서 0번째 줄을 가리키고 있다. 때문에 다음 줄을 가리키기 위해서는 `next()` 메서드를 수행한다.

```
while(rs.next()) {
    // 컬럼단위로 데이터를 추출한다.
}
```

ResultSet객체 →

SMITH	800
ALLEN	1600
WARD	1250

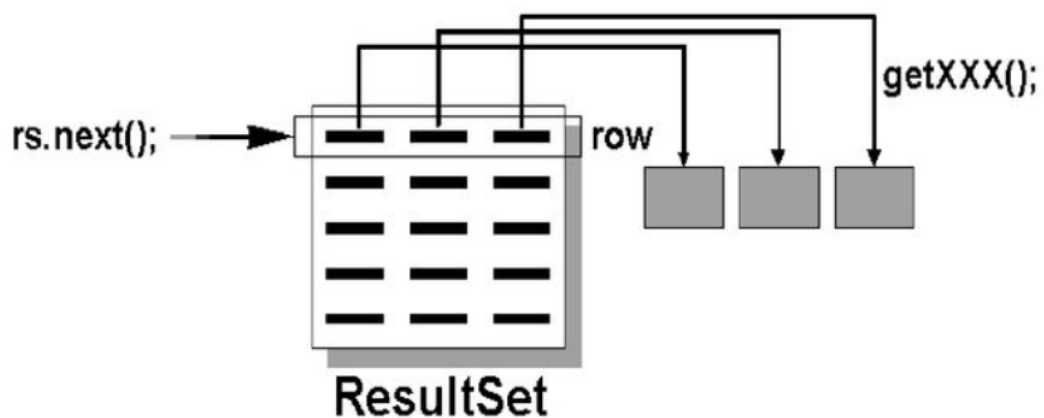
next() →

SMITH	800
ALLEN	1600
WARD	1250

next() →

SMITH	800
ALLEN	1600
WARD	1250

컬럼 단위로 데이터를 추출할 때는 컬럼의 타입에 따라서 메서드를 선택한다.



SQL 데이터 타입	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getXXX() 메소드																			
getByte	✖	×	×	×	×	×	×	×	×	×	×	×	×						
getShort	×	✖	×	×	×	×	×	×	×	×	×	×	×						
getInt	×	×	✖	×	×	×	×	×	×	×	×	×	×						
getLong	×	×	×	✖	×	×	×	×	×	×	×	×	×						
getFloat	×	×	×	×	✖	×	×	×	×	×	×	×	×						
getDouble	×	×	×	×	×	✖	✖	×	×	×	×	×	×						
getBigDecimal	×	×	×	×	×	×	×	✖	✖	×	×	×	×						
getBoolean	×	×	×	×	×	×	×	×	×	✖	×	×	×						
getString	×	×	×	×	×	×	×	×	×	×	✖	✖	×	×	×	×	×	×	×

[illegible]

```
ResultSet rs = stmt.executeQuery("SELECT ename, sal FROM emp");
while(rs.next()) {
    System.out.println(rs.getString(1));
    System.out.println(rs.getInt(2));
}
```

```
while(rs.next()) {
    System.out.println(rs.getString("ename"));
    System.out.println(rs.getInt("sal"));
}
```

- 객체에 읽어온 데이터 담기

하지만 rs는 일시적인 객체이다. DB 작업이 끝나면 폐기되어야 한다.

따라서 USER 테이블에 맞게 User 클래스를 생성해주고,

해당 객체에 담을 수 있도록 한다.

```
class EmpVO {
    private String name;
    private int salary;
    /*
     * getter/setter 메소드들
     */
}
```

```
EmpVO vo = null;
```

```
List<EmpVO> list = new ArrayList<EmpVO>();
```

```
while(rs.next()) {
    vo = new EmpVO();
```



```

        vo.setName(rs.getString("ename"))
        vo.setSalary(rs.getInt("sal"))
        list.add(vo);
    }
}

```

[데이터 삽입/삭제/수정 하기]

접속된 DB 서버에서 Query(SELECT 명령)를 실행하려면 Statement 객체의 **executeQuery()** 메서드를 사용하며 INSERT, DELETE, UPDATE, CREATE TABLE, DROP TABLE 등의 SELECT 명령 이외의 명령을 실행할 때는 **executeUpdate()** 메서드를 사용한다.

executeQuery() 은 SELECT 명령의 실행 결과를 참조하는 **ResultSet 객체**를 반환하여 추출된 결과집합을 행단위로 옮겨가면서 각각의 열의 값을 정해진 메서드로 추출하지만 executeUpdate()는 int 값을 리턴한다. 리턴되는 값의 의미는 주어진 SQL 명령에 의해 **테이블에서 변화된 행의 갯수**를 의미한다.

executeQuery()	ResultSet		
executeUpdate()	int	INSERT	1
		DELETE	삭제된 행의 갯수
		UPDATE	수정된 행의 갯수
		DDL	0

- Statement 사용

```

Statement stmt = conn.createStatement();
stmt.executeUpdate("insert into student values ('둘리', 100)");
stmt.executeUpdate("insert into student values ('또치', 90)");

```

```
int delNum = stmt.executeUpdate("delete from student where name = '둘리'");
int updateNum = stmt.executeUpdate ("update student set score = 'dooly' where name = '둘리'");
```

```
String name = scan.nextLine();
int score = Integer.parseInt(scan.nextLine());
Statement stmt = conn.createStatement();
```

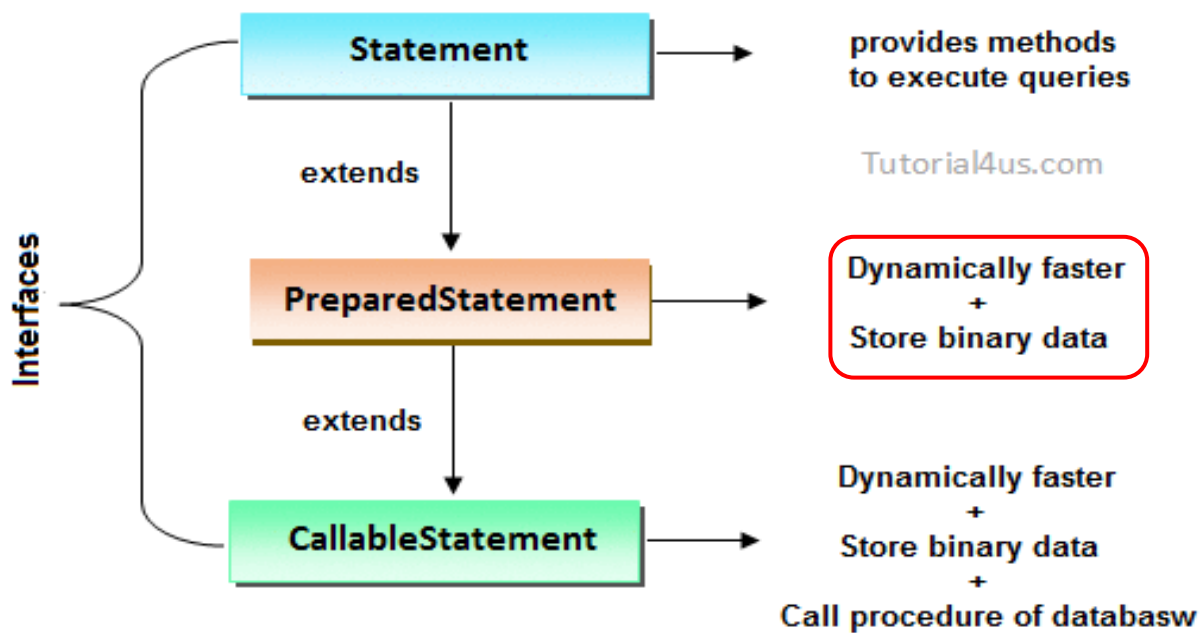
```
stmt.executeUpdate("insert into student values ('"+name+"','"+score+"')");
int delNum = stmt.executeUpdate("delete from student where name = '"+name+"'");
int updateNum = stmt.executeUpdate ("update student set score = "+score +
                                     " where name = '"+name+"'");
```

– PreparedStatement 사용

```
PreparedStatement pstmt = conn.prepareStatement("insert into student values (?, ?)");
pstmt.setString(1, name);
pstmt.setInt(2, score);
pstmt.executeUpdate();
```

```
PreparedStatement pstmt = conn.prepareStatement("delete from student where name = ?")
pstmt.setString(1, name);
int delNum = pstmt.executeUpdate();
```

```
PreparedStatement pstmt = conn.prepareStatement(
    "update student set score = ? where name = ?");
pstmt.setInt(1, score);
pstmt.setString(2, name);
int updateNum = pstmt.executeUpdate();
```



[예제2]

package level1;

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class SelectEmp {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException cnfe) {
            System.out.println("해당 클래스를 찾을 수 없습니다." + cnfe.getMessage());
        }
        return;
    }
}
```

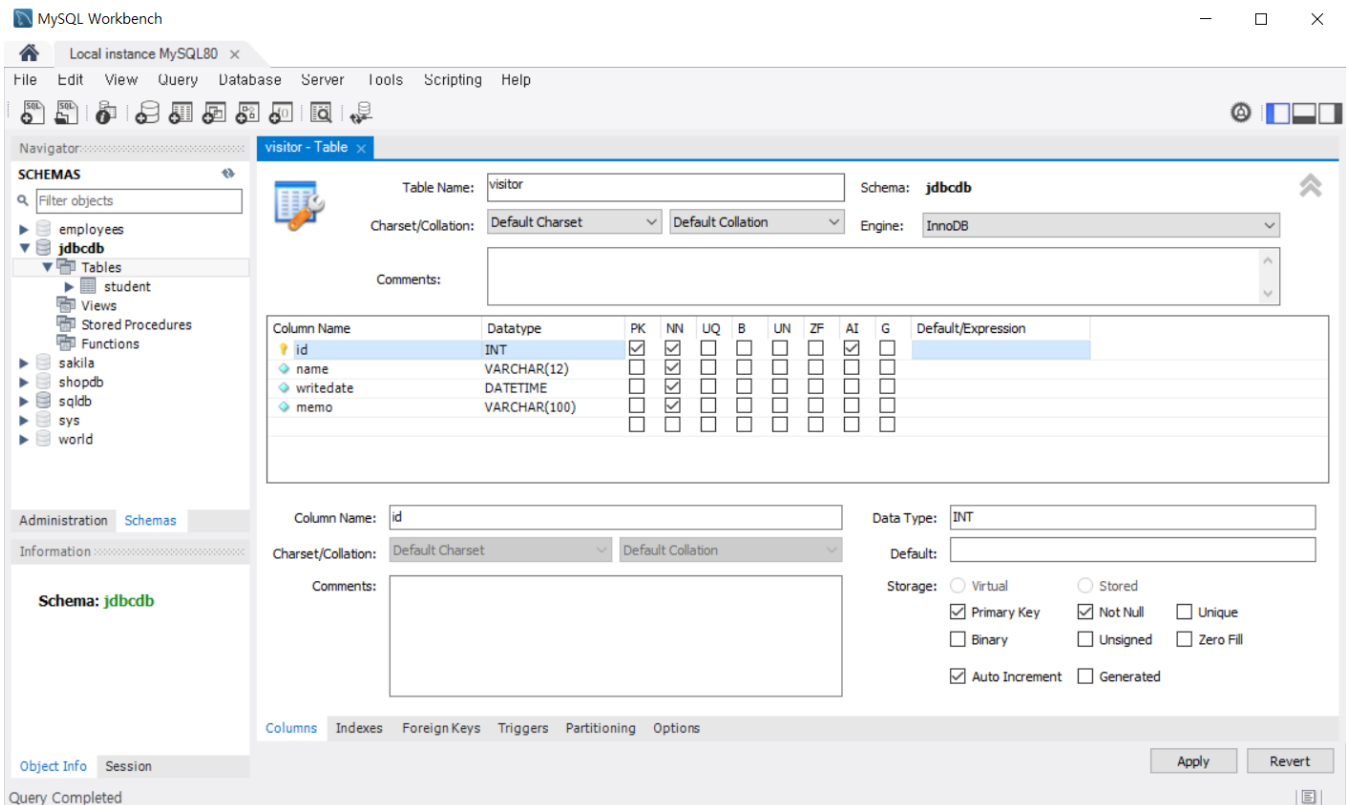
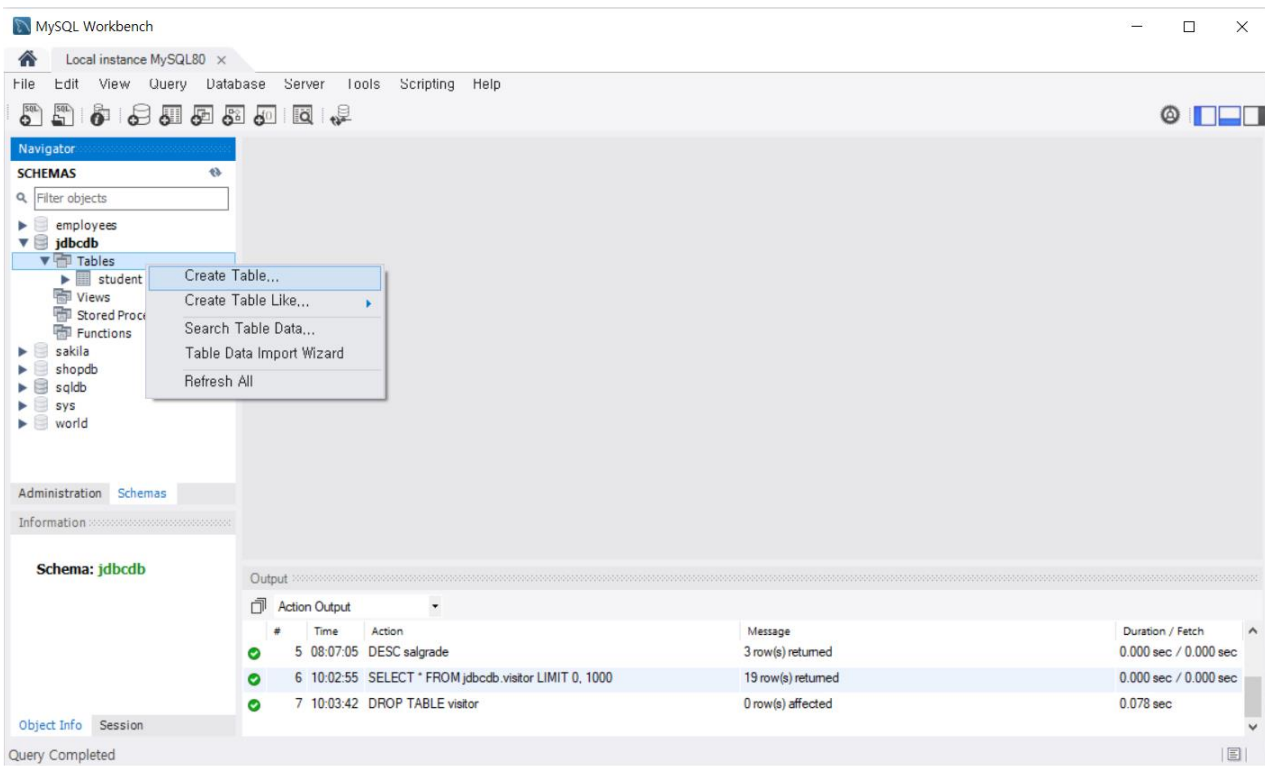
```

    }
    String url =
"jdbc:mysql://localhost:3306/sqlldb?characterEncoding=UTF-8&serverTimezone=UTC";
    String user = "root";
    String passwd = "1234";
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
        conn = DriverManager.getConnection(url, user, passwd);
        stmt = conn.createStatement();
        rs = stmt.executeQuery("select ename, sal, hiredate from emp");
        while(rs.next()) {
            System.out.println(rs.getString("ename")+" ",
                                +rs.getDate("hiredate")+" ", +rs.getInt("sal"));
        }
    } catch (SQLException se1) {
        System.out.println(se1.getMessage());
    } finally {
        try {
            rs.close();
            stmt.close();
            conn.close();
        } catch (SQLException se2) {
            System.out.println(se2.getMessage());
        }
    }
}
}

```

[visitor 테이블 생성]

jdbcdb 라는 데이터베이스를 생성한 후에 다음 내용을 참조하여 visitor 테이블을 생성한다.



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:

Default

Lock Type:

Default

```
1 CREATE TABLE `jdbcd`.'visitor' (  
2   `id` INT NOT NULL AUTO_INCREMENT,  
3   `name` VARCHAR(12) NOT NULL,  
4   `writedate` DATETIME NOT NULL,  
5   `memo` VARCHAR(100) NOT NULL,  
6   PRIMARY KEY (`id`));  
7
```

<

>

Back

Apply

Cancel

Review SQL Script

Apply SQL Script

Applying SQL script to the database

The following tasks will now be executed. Please monitor the execution. Press Show Logs to see the execution logs.

☒ Execute SQL Statements

SQL script was successfully applied to the database.

Show Logs

Back

Finish

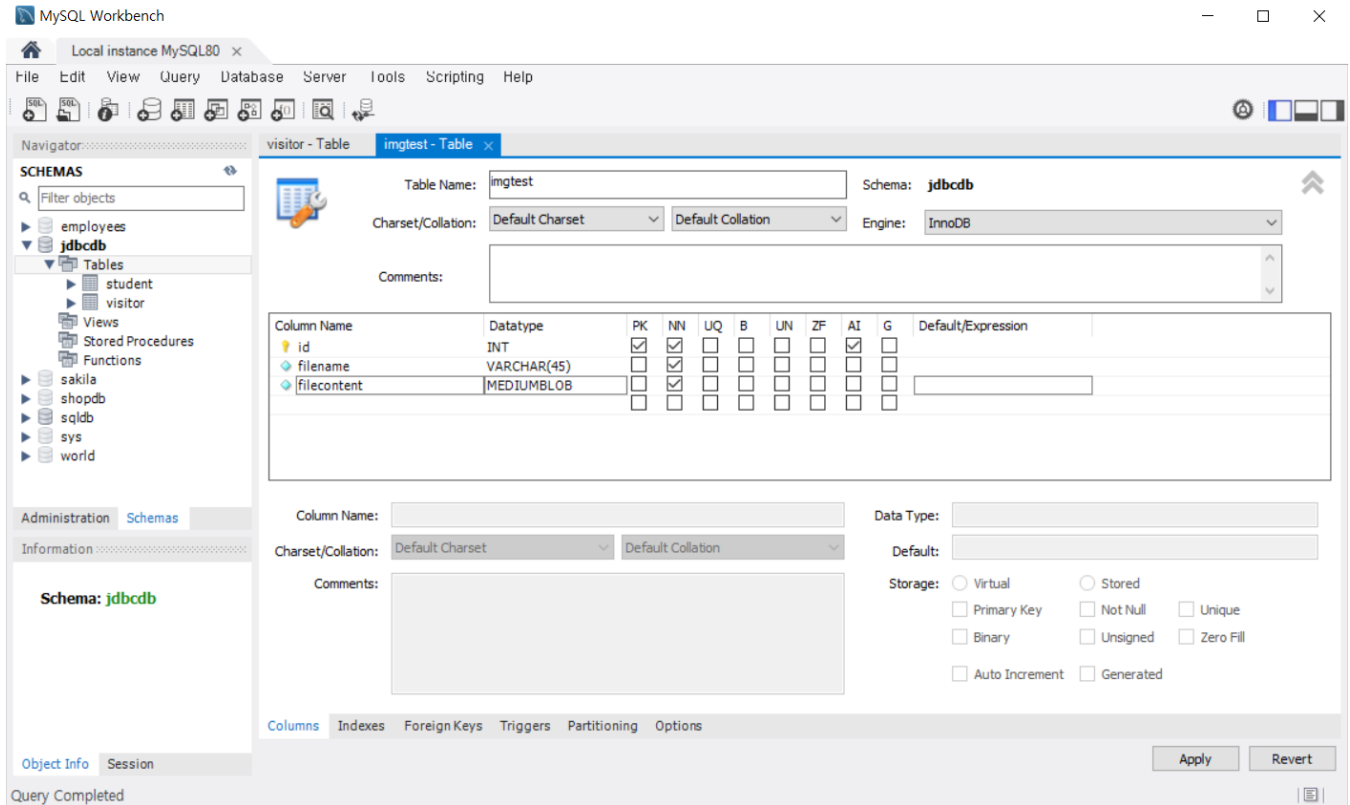
Cancel

Online DDL

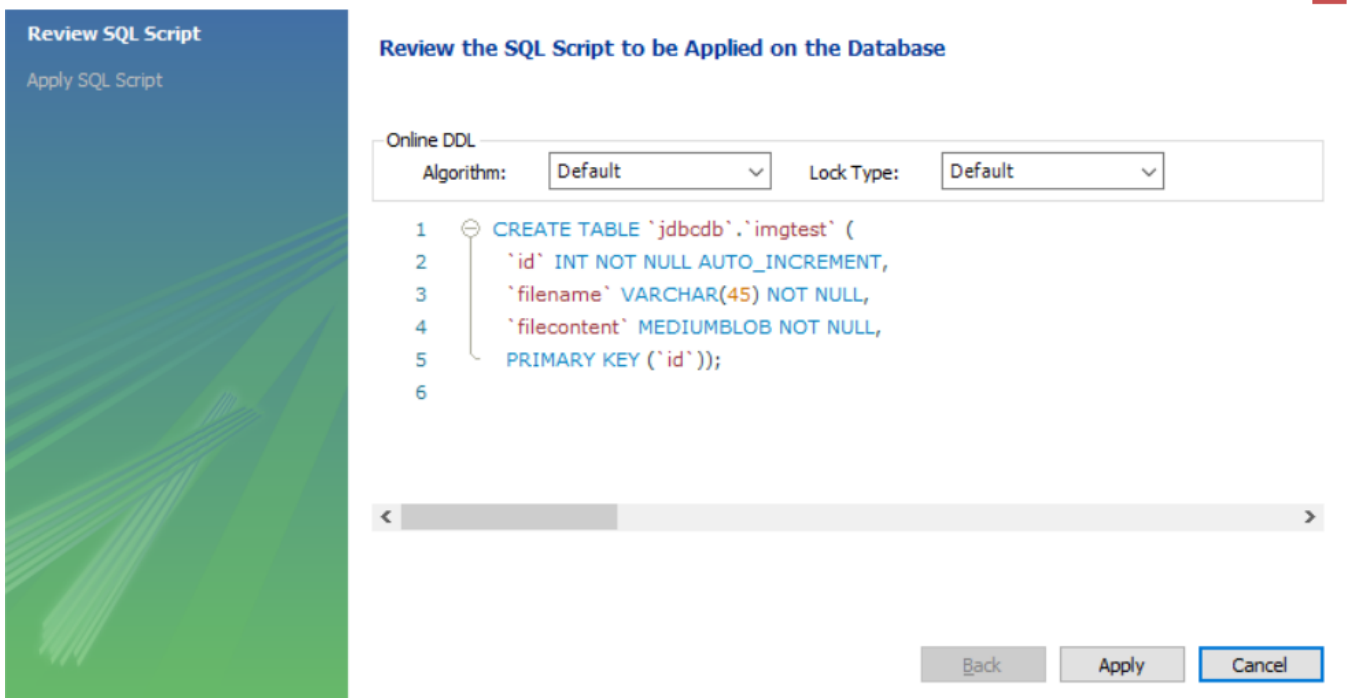
Algorithm: Default

Lock Type: Default

```
1 CREATE TABLE `jdbcdb`.`visitor` (  
2   `name` VARCHAR(12) NOT NULL,  
3   `writedate` DATETIME NOT NULL,  
4   `memo` VARCHAR(100) NOT NULL);  
5
```



Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Applying SQL script to the database

The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.

☒ Execute SQL Statements

SQL script was successfully applied to the database.

Show Logs

Back

Finish

Cancel