Databases – Advanced SQL

Jaeyong Choi
Dept. of Software, Gachon University

SQL Access in a Program

- API (Application Program Interface)
  - A program to interact with a database server
  - Application makes calls to
    - Connect with the database server
    - Send SQL commands to the database server
    - Fetch tuples of result one-by-one into program variables
- Various tools
  - JDBC (Java Database Connectivity) works with Java
  - ODBC (Open Database Connectivity) works with C, C++, C#, Visual Basic, etc.
  - Embedded SQL

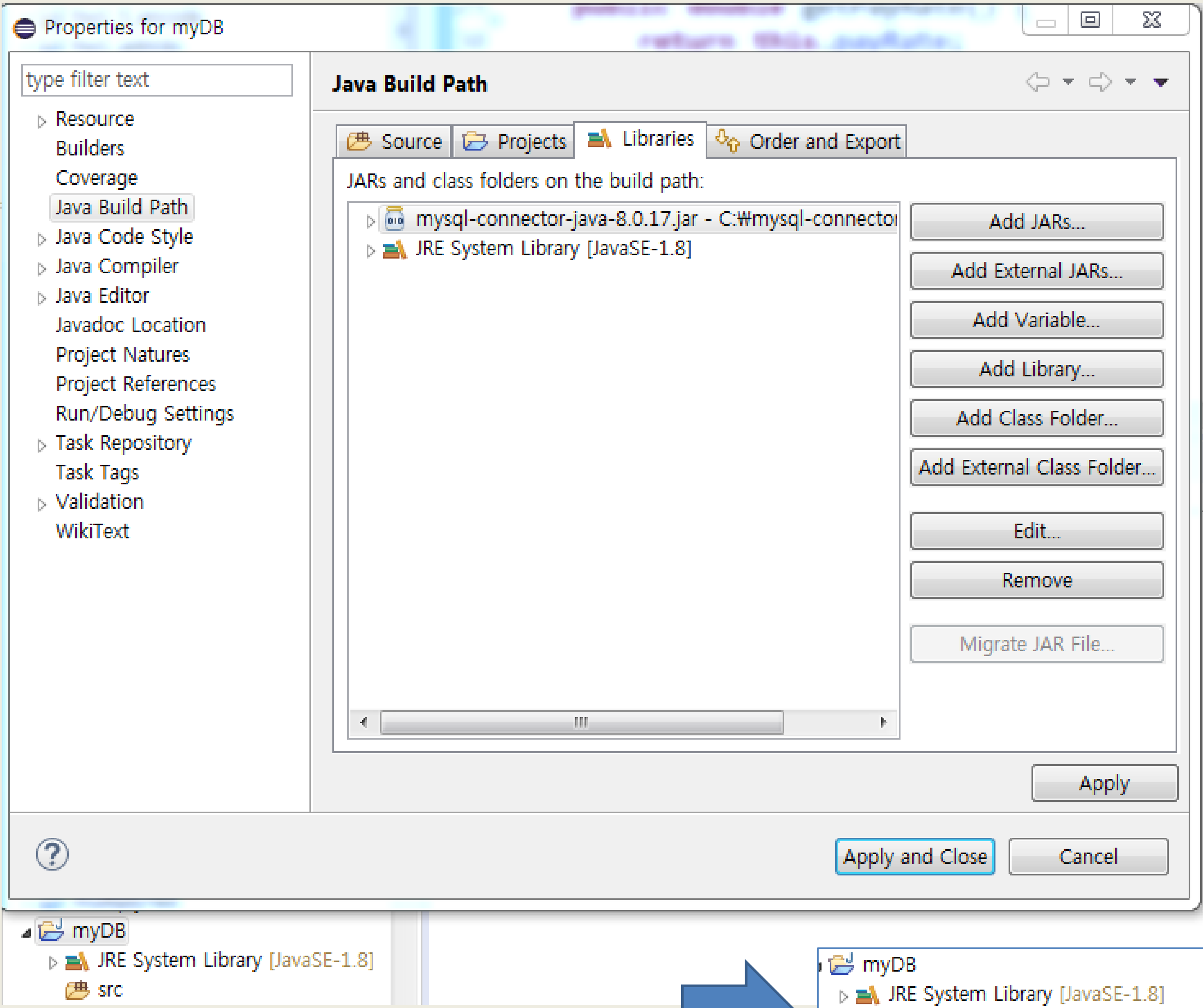
- JDBC
  - Java API for communicating with database systems supporting SQL
  - Supports a variety of features for querying and updating data, and for retrieving query results
  - Also supports metadata retrieval: querying about relations in the database and the names and types of relation attributes
- Model for communicating with the database
  - Open a connection
  - Create a “statement” object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

- ODBC
  - Standard for application program to communicate with a database server
  - Application program interface (API)
    - Open a connection with a database
    - Send queries and updates
    - Get back results
  - Used by applications such as GUI, spreadsheets, etc.

- Embedded SQL
  - The SQL standard defines embeddings of SQL in a variety of programming languages such as C, C++, Java, etc.
  - A language to which SQL queries are embedded is referred to as a *host language*
- **EXEC SQL** statement
  - Used to identify embedded SQL request to the preprocessor
    - **EXEC SQL** <embedded SQL statement >;
    - Note: this varies by language
  - Java embedding uses: **#SQL** { .... };

MySQL JDBC

- MySQL Connector/J
  - https://dev.mysql.com/downloads/connector/j/
  - Operating System: **Platform Independent**
  - Extract *mysql-connector-java-x.x.xx-bin.jar* file from the downloaded zip file (current: 8.0.21)
- Eclipse Java Project
  - Right click project name (e.g., MyDB) and choose **Build Path→Configure Build Path** menu
  - Choose **Libraries** tab and click **Add JARs** button to add MySQL connector jar file



Properties for myDB

type filter text

- Resource
- Builders
- Coverage
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project Natures
- Project References
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

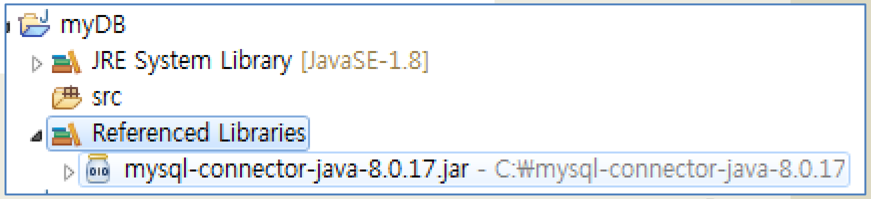
Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

- mysql-connector-java-8.0.17.jar - C:₩mysql-connector
- JRE System Library [JavaSE-1.8]

- myDB
  - JRE System Library [JavaSE-1.8]
  - src



- myDB
  - JRE System Library [JavaSE-1.8]
  - src
  - Referenced Libraries
    - mysql-connector-java-8.0.17.jar - C:₩mysql-connector-java-8.0.17

JDBC Programming

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

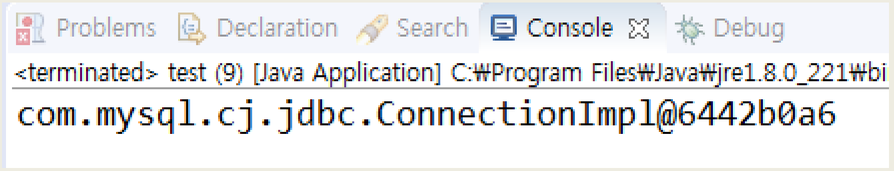
- Connecting to database

```java
public static void main(String[] args) {
    Connection con = null;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String url = "jdbc:mysql://localhost/mydb";
        String user = "root", passwd = "12345";
        con = DriverManager.getConnection(url, user, passwd);
        System.out.println(con);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    // database operations ...

    try {
        if (con != null && !con.isClosed()) con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- If you get error message
  - The timestamp format of the MySQL timezone has changed so that the connector does not recognize it.

```
java.sql.SQLException: The server time zone value '????α? ????' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the serverTimezone configuration property) to use a more specifc time zone value if you want to utilize time zone support.
```

- Modify here
  - String url = "jdbc:mysql://localhost/mydb ?serverTimezone=UTC&useSSL=false ";



```
<terminated> test (9) [Java Application] C:₩Program Files₩Java₩jre1.8.0_221₩bi
com.mysql.cj.jdbc.ConnectionImpl@6442b0a6
```

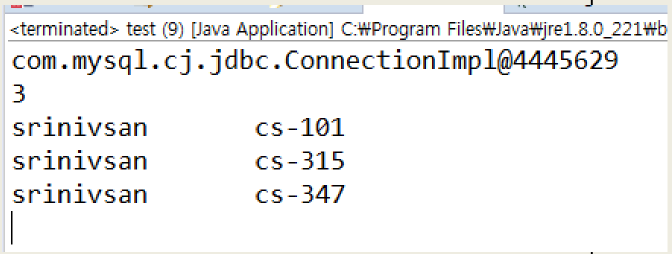
❑ Shipping SQL statements to database system

```
Statement stmt = null;                         import java.sql.Statement;
try {
    stmt = con.createStatement();
    String update = "update instructor set salary = salary * 1.1 where 
dept_name=comp. sci'";
    int count = stmt.executeUpdate(update);
    System.out.println(count);
} catch (SQLException e1) {
    e1.printStackTrace();
}

try {
    if (stmt != null && !stmt.isClosed()) stmt.close();
} catch (SQLException e1) {
    e1.printStackTrace();
}
```

❑ Retrieving result of a query

```
import java.sql.ResultSet;
```

```
Statement stmt = null;
ResultSet rs = null;
try {
    stmt = con.createStatement();
    String sql = "select name, course_id from instructor natural join teaches";
    rs = stmt.executeQuery(sql);

    while (rs.next()) {
        String name = rs.getString(1);
        if (rs.wasNull()) name = "null";
        String course_id = rs.getString(2);
        if (rs.wasNull()) course_id = "null";
        System.out.println(name + "\t" + course_id);
    }
} catch (SQLException e1) {
    e1.printStackTrace();
}

try {
    if (stmt != null && !stmt.isClosed()) stmt.close();
    if (rs != null && !rs.isClosed()) rs.close();
} catch (SQLException e1) {
    e1.printStackTrace();
}
```



```
<terminated> test (9) [Java Application] C:₩Program Files₩Java₩jre1.8.0_221₩b
com.mysql.cj.jdbc.ConnectionImpl@4445629
3
srinivsan       cs-101
srinivsan       cs-315
srinivsan       cs-347
```

❑ Prepared statements

```
import java.sql.PreparedStatement;
```

```
PreparedStatement pstmt = null;
try {
    String psql = "insert into instructor value (?, ?, ?, ?)";
    pstmt = con.prepareStatement(psql);

    String id = "12345", name = "Neumann", dept_name = "Biology";
    int salary = 98000;
    pstmt.setString(1, id);
    pstmt.setString(2, name);
    pstmt.setString(3, dept_name);
    pstmt.setInt(4, salary);

    int count = pstmt.executeUpdate();
    System.out.println(count);
} catch (SQLException e1) {
    e1.printStackTrace();
}

try {
    if (pstmt != null && !pstmt.isClosed()) pstmt.close();
} catch (SQLException e1) {
    e1.printStackTrace();
}
```

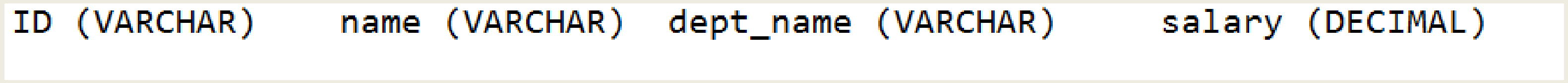
❑ Metadata features

```
import java.sql.ResultSetMetaData;
```

```
ResultSetMetaData rsmd = rs.getMetaData();
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
    System.out.print(rsmd.getColumnName(i) + " (");
    System.out.print(rsmd.getColumnTypeName(i) + ")\t");
}
System.out.println();
```

Metadata 확인을 위한 코드
– try catch 문으로 exception 처리 하거나 throws 통해서 처리 필요

public static void main(String[] args) throws SQLException



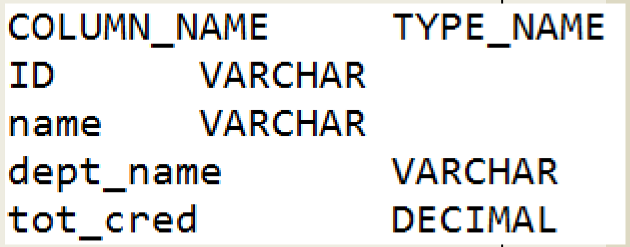
```
ID (VARCHAR)    name (VARCHAR)  dept_name (VARCHAR)     salary (DECIMAL)
```

Metadata features *cont'd*

```java
import java.sql.DatabaseMetaData;
```

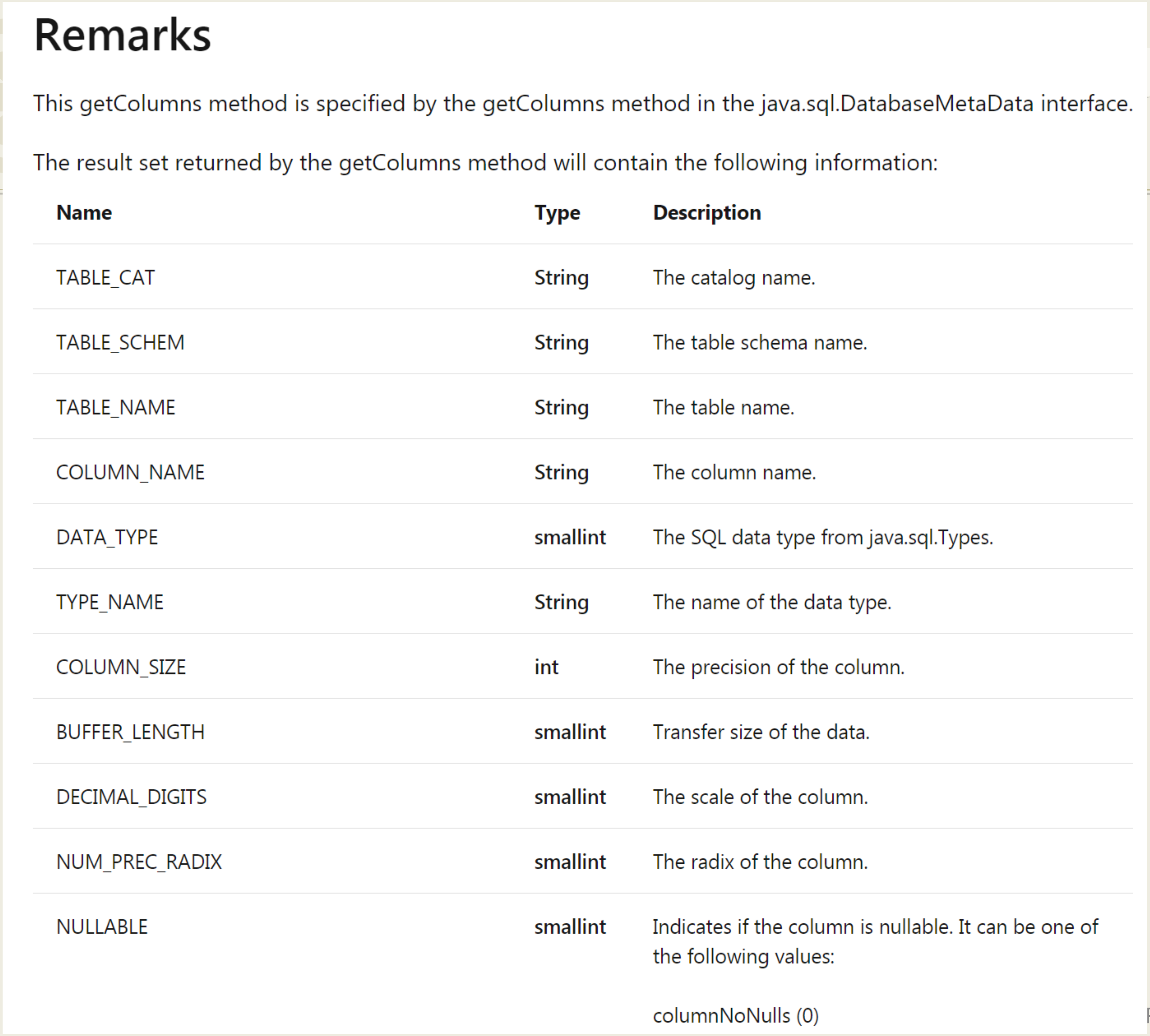
public java.sql.ResultSet getColumns(java.lang.String **catalog**, java.lang.String **schema**, java.lang.String **table**, java.lang.String **col**)

```java
DatabaseMetaData dbmd = null;
ResultSet mdrs = null;
try {
    dbmd = con.getMetaData();
    mdrs = dbmd.getColumns("mydb", null, "student", null);
    String[] column = { "COLUMN_NAME", "TYPE_NAME" };
    System.out.println(column[0] + "\t" + column[1]);

    while (mdrs.next()) {
        String column_name = mdrs.getString(column[0]);
        String type_name = mdrs.getString(column[1]);
        System.out.println(column_name + "\t" + type_name);
    }
} catch (SQLException e1) {
    e1.printStackTrace();
}

try {
    if (mdrs != null && !mdrs.isClosed()) mdrs.close();
} catch (SQLException e1) {
    e1.printStackTrace();
}
```



```
COLUMN_NAME      TYPE_NAME
ID      VARCHAR
name    VARCHAR
dept_name        VARCHAR
tot_cred         DECIMAL
```

https://docs.microsoft.com/en-us/sql/connect/jdbc/reference/getcolumns-method-sqlserverdatabasemetadata?view=sql-server-ver15



Remarks

This getColumns method is specified by the getColumns method in the java.sql.DatabaseMetaData interface.

The result set returned by the getColumns method will contain the following information:

Name	Type	Description
TABLE_CAT	**String**	The catalog name.
TABLE_SCHEM	**String**	The table schema name.
TABLE_NAME	**String**	The table name.
COLUMN_NAME	**String**	The column name.
DATA_TYPE	**smallint**	The SQL data type from java.sql.Types.
TYPE_NAME	**String**	The name of the data type.
COLUMN_SIZE	**int**	The precision of the column.
BUFFER_LENGTH	**smallint**	Transfer size of the data.
DECIMAL_DIGITS	**smallint**	The scale of the column.
NUM_PREC_RADIX	**smallint**	The radix of the column.
NULLABLE	**smallint**	Indicates if the column is nullable. It can be one of the following values: columnNoNulls (0)

- Transaction control
  - By default, each SQL statement is treated as a separate transaction that is committed automatically
    - Bad idea for transactions with multiple updates
  - Can turn off/on automatic commit on a connection
    - con.setAutoCommit(false);
    - con.setAutoCommit(true);
  - Transactions are committed or rolled back explicitly
    - con.commit();
    - con.rollback();