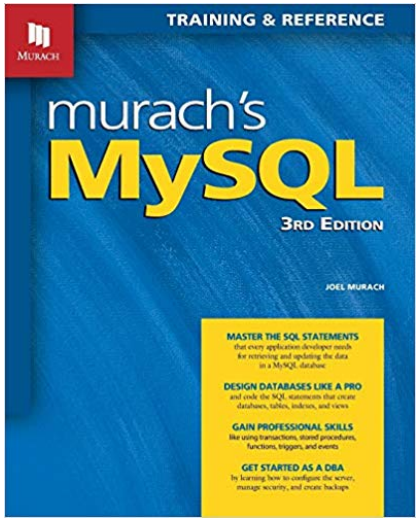
Building host programs Java

Connecting to a MySQL database
Topic 4 Lesson 7

Adapted from Chapter 1



https://dev.mysql.com/doc/connector-j/8.0/en/

https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html

https://pymysql.readthedocs.io/en/latest/

Embedding SQL

SQL commands can be called from within a host language (e.g., C++ or Java) program. SQL statements can refer to host variables (including special variables used to return status).

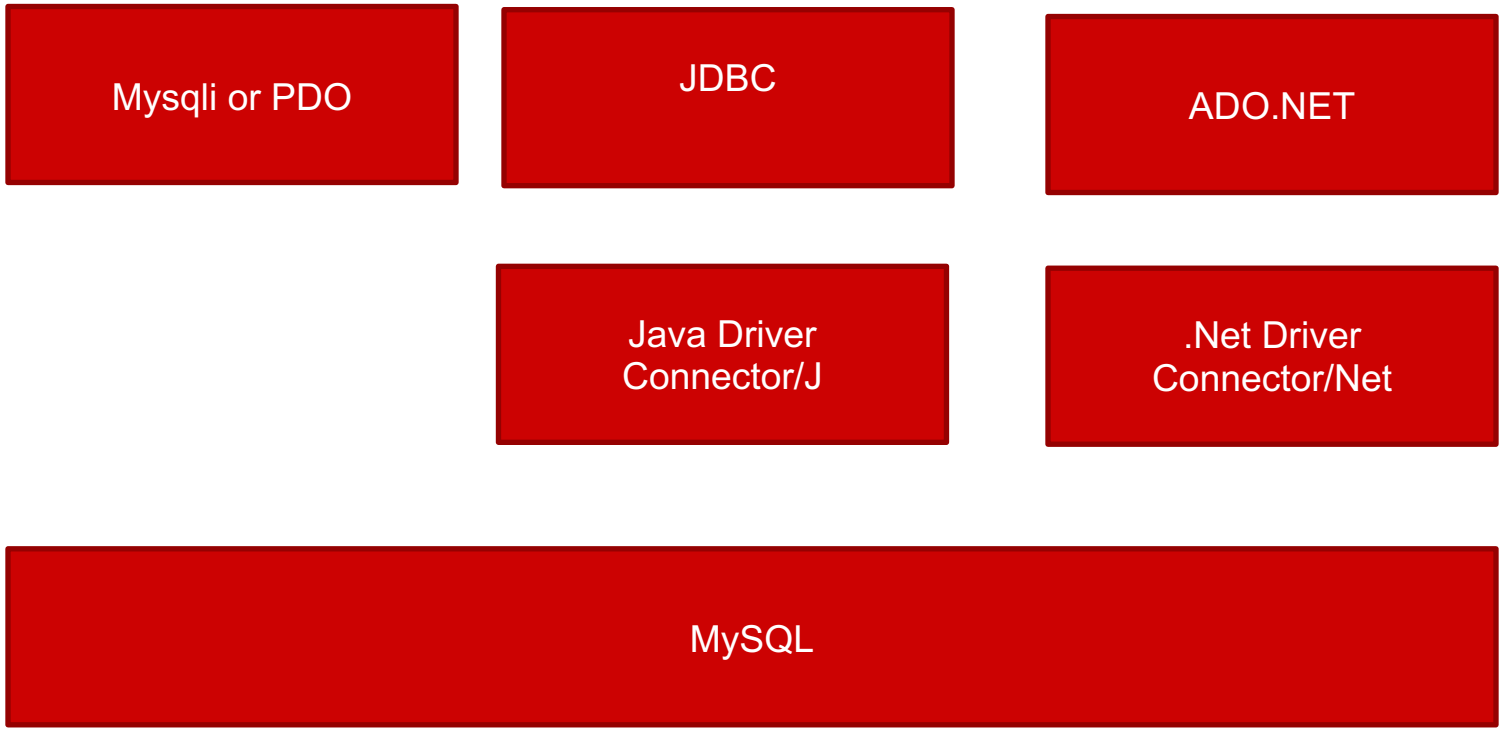
Two main integration approaches:

- Embed SQL in the host language (Embedded SQL, SQLJ). A Preprocessor converts SQL code to host language calls. The output from the preprocessor is then compiled by the host compiler
- Create special API to call SQL commands

JDBC Java Database Connectivity API (for JAVA)
http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/

ODBC Standard database connectivity API Pep 249 – **Python** Database Application specification https://www.python.org/dev/peps/pep-0249/

Embedded SQL



Database (API)s

Add a library with database calls (API)

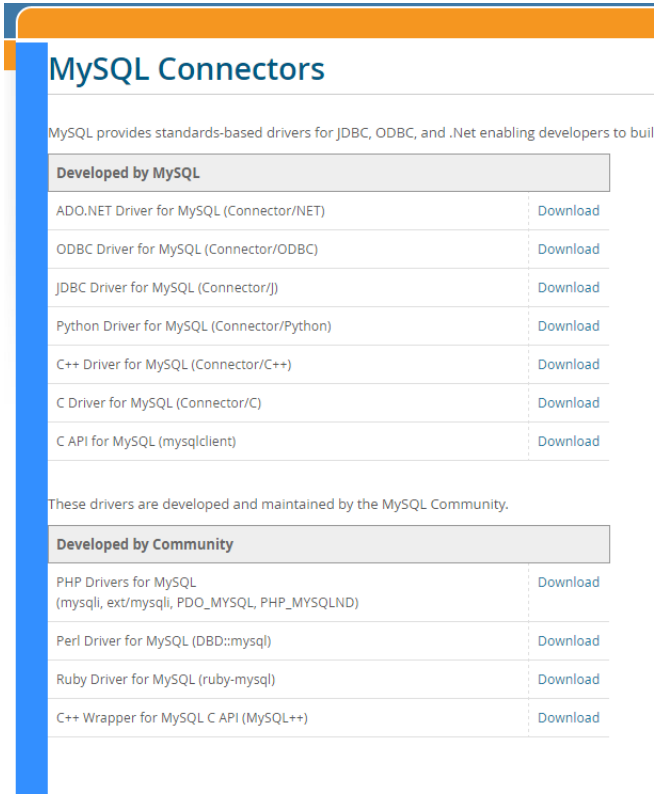
- Special standardized interface: procedures/objects
  - Pass SQL strings from host language, presents result sets in a host language-friendly way

A “driver” traps the calls and translates them into DBMS specific code (Oracle, MySQL, SQL Server etc.)

- database can be across a network

GOAL: applications are independent of database systems and operating systems

Download the desired driver



MySQL Connectors

MySQL provides standards-based drivers for JDBC, ODBC, and .Net enabling developers to buil

Developed by MySQL	
ADO.NET Driver for MySQL (Connector/NET)	Download
ODBC Driver for MySQL (Connector/ODBC)	Download
JDBC Driver for MySQL (Connector/J)	Download
Python Driver for MySQL (Connector/Python)	Download
C++ Driver for MySQL (Connector/C++)	Download
C Driver for MySQL (Connector/C)	Download
C API for MySQL (mysqlclient)	Download

These drivers are developed and maintained by the MySQL Community.

Developed by Community	
PHP Drivers for MySQL (mysqli, ext/mysqli, PDO_MYSQL, PHP_MYSQLND)	Download
Perl Driver for MySQL (DBD::mysql)	Download
Ruby Driver for MySQL (ruby-mysql)	Download
C++ Wrapper for MySQL C API (MySQL++)	Download

GO TO:
https://www.mysql.com/products/connector/

MySQL Drivers

- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API.
- Connector/Net enables developers to create .NET applications that connect to MySQL. Connector/Net implements a fully functional ADO.NET interface and provides support for use with ADO.NET
- Connector/J provides driver support for connecting to MySQL from **Java** applications using the standard Java Database Connectivity (JDBC) API.
- Connector/Python provides driver support for connecting to MySQL from **Python** applications using an API that is compliant with the Python DB API version 2.0. http://dev.mysql.com/doc/connector-python/en/
- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/C is a standalone replacement for the MySQL Client Library (libmysqlclient), to be used for C applications.

JDBC Processing (Java)

Steps to submit a database query:

Load the JDBC driver
Connect to the data source
Execute SQL statements

JDBC Architecture

Application or the client (initiates and terminates connections, submits SQL statements)
Driver manager (loads the JDBC driver)
Driver (connects to data source, transmits requests and returns/translates results and error codes)
Data source (processes SQL statements)

JDBC Driver Class

All drivers are managed by the Java DriverManager class

To load a JDBC driver in Java host code:

Class.forName(“oracle/jdbc.driver.Oracledriver”); /Oracle

Class.forName("com.mysql.jdbc.Driver"); /MySQL

When starting the Java application:

-Djdbc.drivers=oracle/jdbc.driver

Or provide the driver in the CLASSPATH directory

For a description of the flags that can be passed to driver:

https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-configuration-properties.html

Connecting to a DB via JDBC

Interact with a data source through sessions. Each connection identifies a logical session.
JDBC URL: jdbc:<subprotocol>:<otherParameters>

Example:

```
//Define URL of database server for
 // database named mysql on the localhost
 // with the default port number 3306.
 String url =
    "jdbc:mysql://localhost:3306/mysql";

//Get a connection to the database for a user named root with a xxxx password.
Connection con = DriverManager.getConnection( url,"root", “xxxx");

//Display URL and connection information
 System.out.println("URL: " + url);
 System.out.println("Connection: " + con);
```

Connection class interface

public int **getTransactionIsolation()** and
void **setTransactionIsolation**(int level)
Sets isolation level for the current connection.

public boolean **getReadOnly()** and void **setReadOnly(boolean b)**
Specifies whether transactions in this connection are readonly

public boolean **getAutoCommit()**
and void **setAutoCommit(boolean b)**
If autocommit is set, then each SQL statement is considered its own transaction. Otherwise, a transaction is committed using commit(), or aborted using rollback().

public boolean **isClosed()**
Checks whether connection is still open.

Executing SQL statements

Three different methods to execute SQL statements:

- **Statement** (both static and dynamic SQL statements)
- **PreparedStatement** (semi-static SQL statements)
- **CallableStatment** (stored procedures)

PreparedStatement class: Precompiled, parameterized SQL statements:

- Structure of the SQL statement is fixed
- Values of parameters are determined at run-time

Prepared stmt: pass and define arguments

String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=**con.prepareStatement**(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);
// No return rows use executeUpdate()
int numRows = **pstmt.executeUpdate();**

Parameters are positional

Result set (cursor)

PreparedStatement.executeUpdate only returns the number of affected records

PreparedStatement.executeQuery returns data, encapsulated in a ResultSet object (a cursor)

```
ResultSet rs=pstmt.executeQuery(sql);
// rs is now a cursor
While (rs.next()) {
// process the data  }
```

ResultSet: Cursor with seek functionality

A ResultSet is a very powerful cursor:

previous(): moves one row back
absolute(int num): moves to the row with the specified number
relative (int num): moves forward or backward
first() and **last**()

Functionality not available for MySQL cursors

Java to SQL types and get methods

SQL Type	Java class	Result Set get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	Java.sql.Date	getDate()
TIME	Java.sql.Time	getTime()
TIMESTAMP	Java.sql.Timestamp	getTimestamp()

JDBC: Processing errors and exceptions

Most of java.sql can throw an error and set SQLException when an error occurs

An SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause.

SQLWarning is a subclass of SQLException

- Not as severe as an error
- They are not thrown
- Code has to explicitly test for a warning

Example of try and catch for error handling

```
try {
            stmt=con.createStatement();
            warning=con.getWarnings();
            while(warning != null) {
                        // handle SQLWarnings;
                        warning = warning.getNextWarning():
             }
      con.clearWarnings();
     stmt.executeUpdate(queryString);
      warning = con.getWarnings();
                …
   }    //end try
   catch( SQLException SQLe) {
    // handle the exception
     System.out.println( SQLe.getMessage());}
```

Examining meta data for the DB

DatabaseMetaData object gives information about the database system catalog.

```
DatabaseMetaData md = con.getMetaData();
// print information about the driver:
System.out.println(
        “Name:” + md.getDriverName() +
         “version: ” + md.getDriverVersion());
```

Metadata- print out tables and fields

```
DatabaseMetaData md=con.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
String tableName;
While(trs.next()) {
        tableName = trs.getString("TABLE_NAME");
        System.out.println("Table: " + tableName);
        //print all attributes
        ResultSet crs = md.getColumns(null,null,tableName, null);
        while (crs.next()) {
                System.out.println(crs.getString("COLUMN_NAME" + ", ");
         }

   }
```

http://docs.oracle.com/javase/10/docs/api/java/sql/DatabaseMetaData.html

Connect, Process, check for errors

```
Connection con = // connect
      DriverManager.getConnection(url, ”login", ”pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
        // loop through result tuples
         while (rs.next()) {
               String s = rs.getString(“name");
                Int n = rs.getFloat(“rating");
                System.out.println(s + " " + n);
           }
    } catch(SQLException ex) {
        System.out.println(ex.getMessage () +
          ex.getSQLState () + ex.getErrorCode ());
}
```

Connect

Get multiset

Process with cursor

Catch Errors

Java documentation

For documentation refer to:
https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-examples.html

Java Summary

APIs such as JDBC introduce a layer of abstraction between application and DBMS

Embedded SQL allows execution of parameterized static queries within a host language

Dynamic SQL allows execution of completely ad hoc queries within a host language

Cursor mechanism allows retrieval of one record at a time and bridges impedance mismatch between host language and SQL