

Chapter 15

Java and the Database (JDBC)

Foundational Java
Key Elements and Practical Programming

Persistent Storage

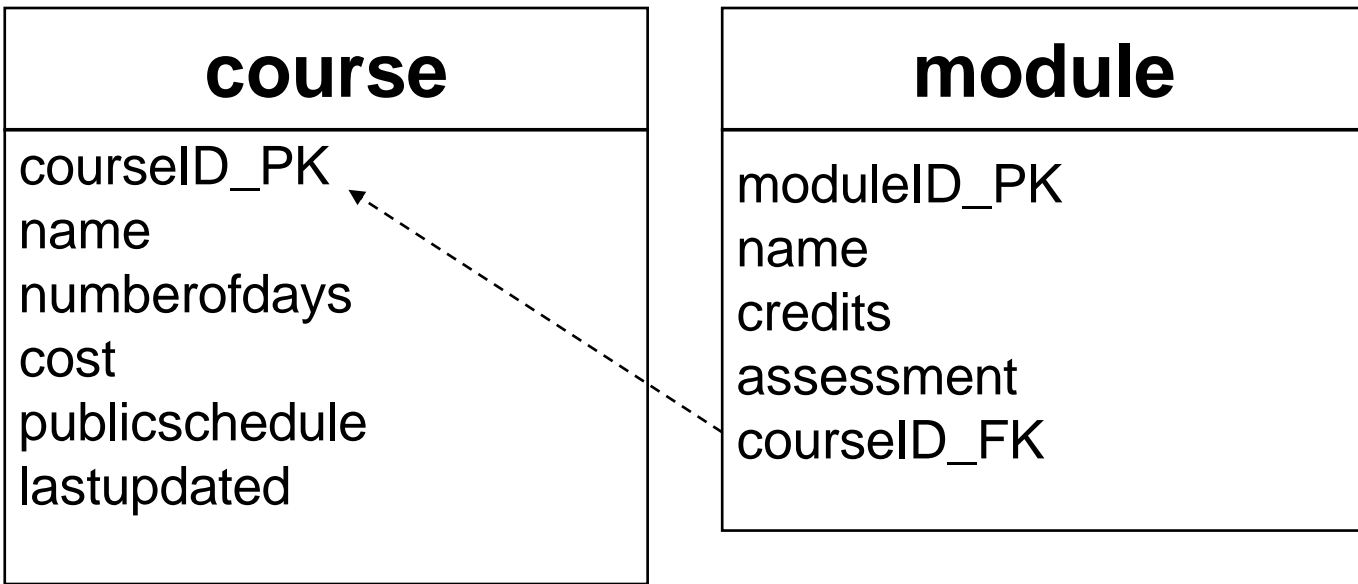
- The data in Java programs does not last beyond a single run of a program
- Most applications require some more persistent storage of state than this
 - In most cases we will use a database
 - Sequential files do not allow the querying, performance, availability and accessibility of database storage

JDBC

- Although there are several different types of database, most current commercial databases are relational
- JDBC (Java Database Connectivity) is a bridge between the table schemas of a relational database and Java code
- JDBC enables a Java application to connect to a relational database and execute SQL statements
 - Store and retrieve the state of any objects that need to be persistent outside of the run time of the application

An Example Database

- Each course may have multiple modules
- Each module belongs to only one course

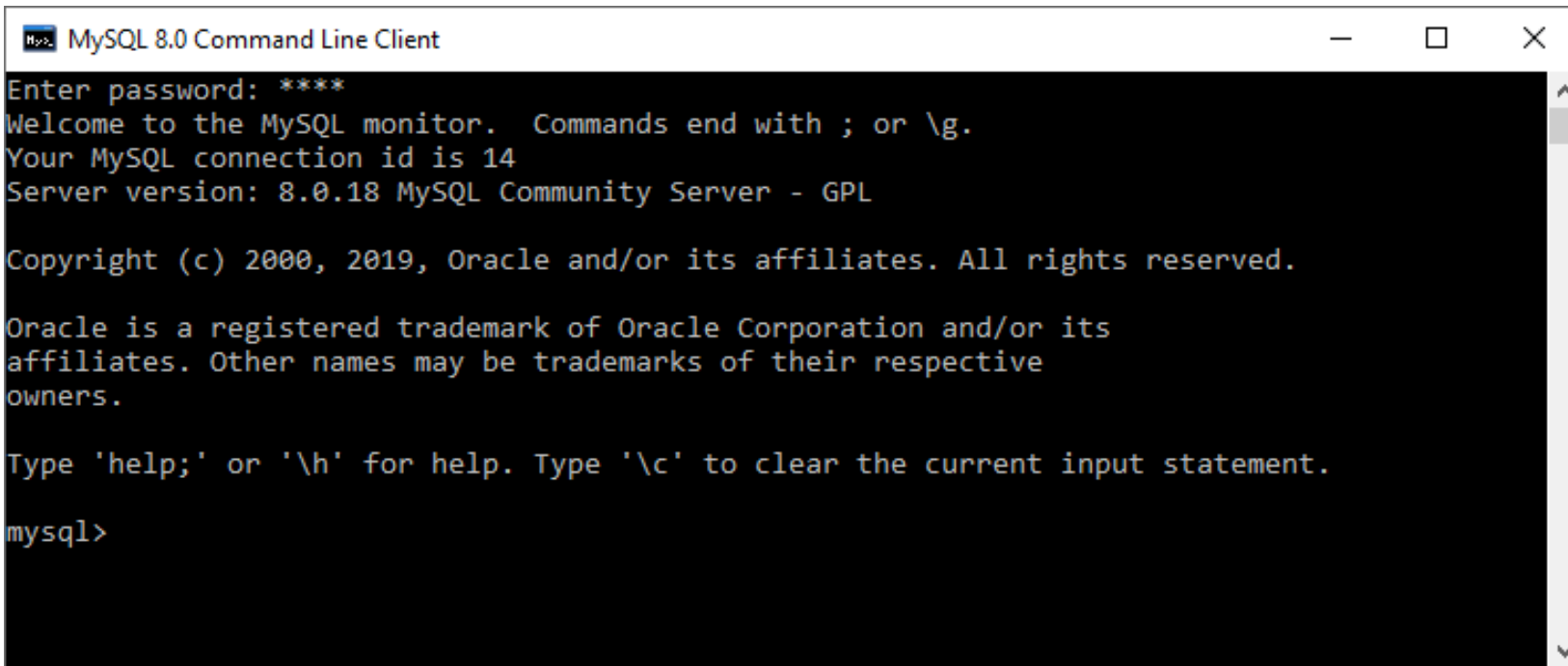


Using MySQL

- MySQL is a popular open-source database
 - It is a fully featured RDBMS
 - Includes a number of tools to help with configuration and management
- MySQL can be run as a Windows service
- There are versions of MySQL for several different operating systems
- Oracle own and sell the commercial versions of MySQL

MySQL Command Line Client

- MySQL includes the command line client as one of the tools
- It will ask for the password to login, configured when MySQL was installed
- If you log in successfully you will see a “mysql” prompt after the copyright notices



```
MySQL 8.0 Command Line Client
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.18 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Creating a New Database

- To create a new MySQL database, use the 'create database' command, i.e.

```
mysql>create database databasename;
```

- For example to create a new database called 'courses', we would enter the following command

```
mysql>create database courses;
```

- Commands are terminated by a semicolon

Showing Databases

- The “show databases” command will list all the databases that have been created

```
mysql>show databases;
```

- If you have created the “courses” database you should see it added to list of databases that come as part of the MySQL installation

```
+-----+
| Database |
+-----+
| courses  |
| information_schema |
| mysql    |
| performance_schema |
| sakila   |
| sys      |
| world    |
+-----+
```


Connecting to a Database

- We can connect to any available database with ‘use *databasename*’
- To connect to the ‘courses’ database, we would enter the following command:

```
mysql>use courses;
```

- This should result in the ‘Database changed’ message coming back from MySQL
 - At this point, however, the database has no schema, so there is not much to connect to

Structured Query Language (SQL)

- SQL is the standard language for accessing relational databases, and allows you to
 - Create tables (CREATE statements)
 - Insert data (INSERT statements)
 - Update data (UPDATE statements)
 - Execute queries to retrieve data (SELECT statements)
 - Remove rows from tables (DELETE statements)
 - Perform other operations connected with data and table management, such as dropping tables from the database (DROP statements)

Generating Primary Keys

- Add 'AUTO_INCREMENT' to the configuration of the primary key

```
CREATE TABLE course (  
  courseID_PK INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT
```

- If we do not provide a key value MySQL will generate a new integer key every time a new record is inserted
- Different databases have different ways of generating keys

Using a DDL Script

- Entering SQL to set up and populate tables manually can be tedious and error prone
- Better to write the SQL statements in a DDL (Database Definition Language) file
- To execute a DDL file use the 'source' command, followed by the path and filename

```
mysql>source C:/ddlfiles/courses.ddl;
```

- The separator character between subfolders in the path must be a forward slash

Example DDL script

- Written in an external text file that can be called from MySQL

```
USE courses;
DROP TABLE course;
CREATE TABLE course (
  courseID_PK INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(35),
  numberofdays INTEGER,
  cost FLOAT,
  publicschedule BOOLEAN,
  lastupdated DATE
);
INSERT INTO course (name, numberofdays, costperperson, publicschedule, lastupdated) VALUES
('Introduction to Java',3,1000.00,true,'2020-08-11');
INSERT INTO course (name, numberofdays, costperperson, publicschedule, lastupdated) VALUES
('Agile Programming Techniques',2,650.00,false,'2020-12-01');
etc..
```

Viewing Table Names

- In MySQL, the 'show tables' command lists all the tables in the database to which you are connected

```
+-----+  
| Tables_in_courses |  
+-----+  
| course            |  
| module           |  
+-----+
```

Viewing Table Schema

- To see the schema of a table, we can use the 'describe' command, which shows the schema of the named table

```
mysql>describe course;
```

```
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| courseID_PK   | int(11)       | NO   | PRI | NULL     | auto_increment |
| name          | varchar(35)   | YES  |     | NULL     |                |
| numberofdays | int(11)       | YES  |     | NULL     |                |
| cost          | float         | YES  |     | NULL     |                |
| publicschedule | tinyint(1)    | YES  |     | NULL     |                |
| lastupdated   | date          | YES  |     | NULL     |                |
+-----+-----+-----+-----+-----+-----+

```

Creating an Authorised MySQL User

- We need a user with the necessary access privileges to the database to connect to MySQL from Java
- Log onto MySQL as the root user and grant access to a user identified by a username and a password.
- Create the user with name, database URL and password
 - Username 'javaclient'
 - Database URL server 'localhost'
 - Password 'foundjava'

```
create user 'javaclient'@'localhost' identified by 'foundjava';
```


Giving an Authorised MySQL User Access Privileges

- Once a user has been added, they can be granted various privileges on selected databases
- The following “grant” command provides a general set of privileges, including creating, updating and deleting records, across all tables in the “courses” database.

```
grant all privileges on courses.* to 'javaclient';
```

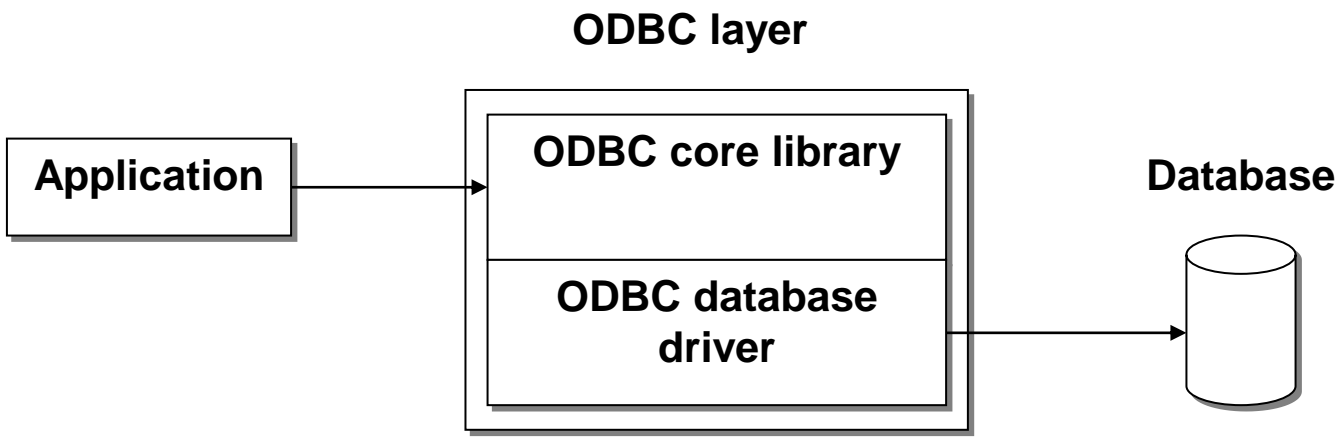
- It will then be possible to connect to the “courses” database from Java programs with the username “javaclient” and the password “foundjava”.

MySQL Commands

MySQL Command	Meaning
<code>create database <i>dbname</i></code>	Create a new database
<code>use <i>dbname</i></code>	Connect to an existing database
<code>show databases</code>	Show the names of all databases
<code>show tables</code>	Show the names of all tables in the current database
<code>exit / quit</code>	Both exit from MySQL
<code>source <i>path/filename</i></code>	Execute a SQL script file
<code>describe <i>tablename</i></code>	Show the schema of a table
<code>create user '<i>username</i>'@'<i>url</i>' identified by '<i>password</i>';</code>	Create a new user with the given username and password
<code>grant <i>privilege type</i> on <i>dbname.tablename</i> to '<i>username</i>'</code>	Grant privileges to resources to this user

Java Database Access with JDBC

- JDBC based on concepts from from ODBC (Open DataBase Connectivity)
- ODBC consists of a core library, that the application uses, and a supporting database driver, that links the standard library to specific databases

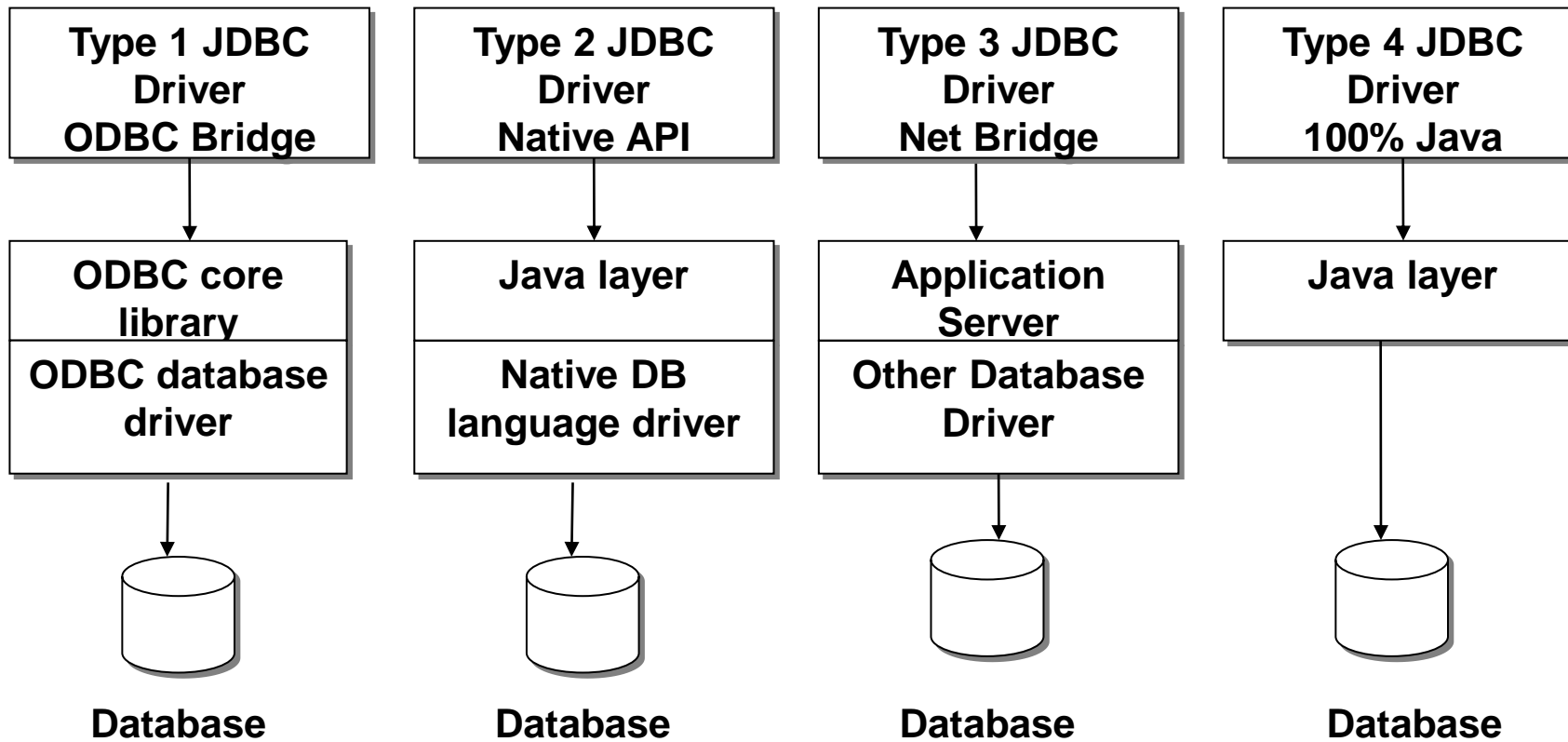


JDBC API

- A ‘thin’ Application Programming Interface (API) that wraps Java code around SQL queries and results
- Implementing classes are supplied by different vendors
- Requires a JDBC driver to communicate with the database and translate between database types and Java types
 - e.g. String (Java) = VARCHAR (database)

JDBC Drivers

- There are four different types of JDBC driver with different characteristics

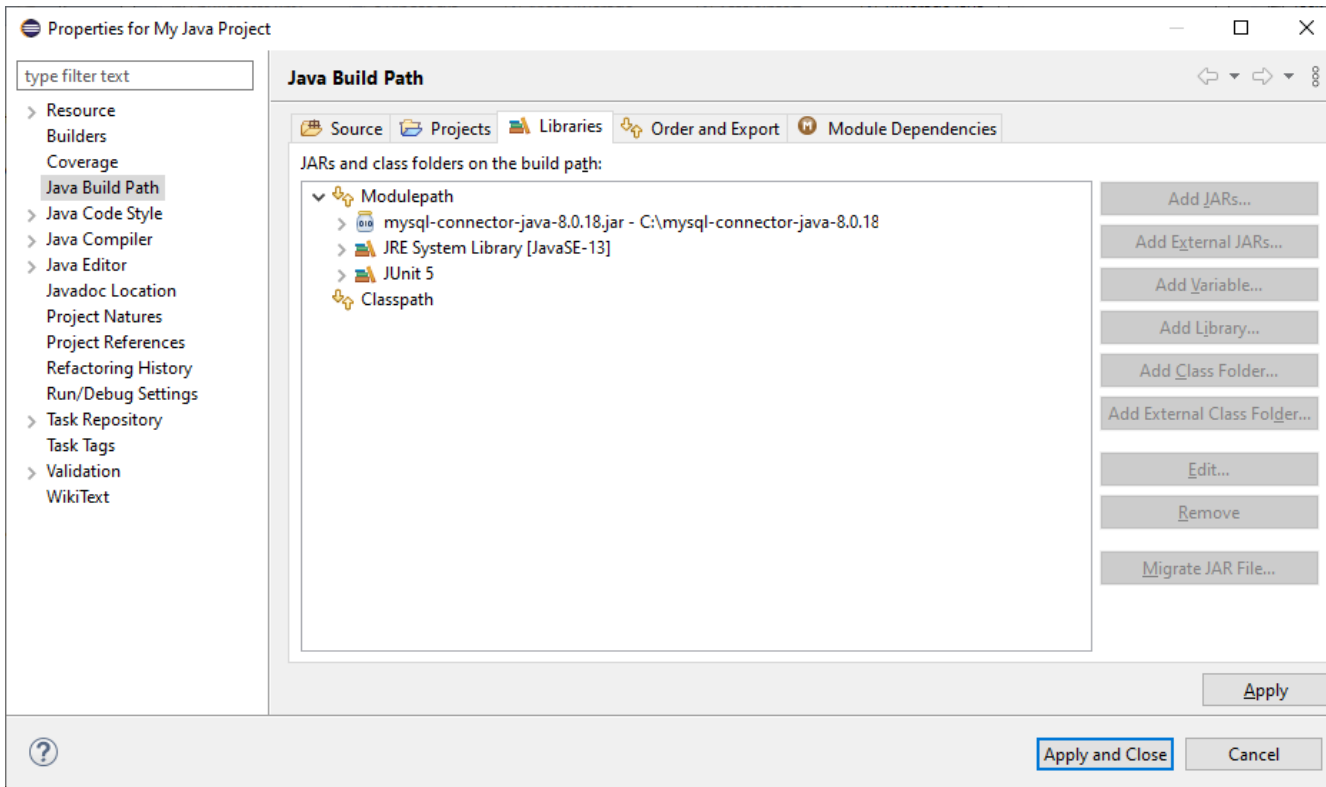


MySQL Connector Driver

- The driver we will be using in these examples is the type 4 MySQL Connector/J driver
 - Can be freely downloaded as an archive from the MySQL web site
 - Needs to be unzipped into a suitable location on your computer
- Inside the main folder of the unzipped archive there will be a JAR file containing the driver
 - “mysql-connector-java-8.0.18-bin.jar” (or something similar depending on the version)

Adding the Driver to Eclipse

- 'Project' -> 'Properties'
- 'Java Build Path', 'Libraries' tab



Press 'Add External JARS...' and browse for the jar file

Running Outside of Eclipse

- If you want to run database access code from the command line, outside of Eclipse, you will need to set the classpath on the command line, e.g.

```
set classpath=%CLASSPATH%;path/databaselibrary.jar;
```


MySQL Database URLs

- The tiers in a two-tier connection are a Java program and a database, connected via a JDBC Driver.
- The JDBC driver connects to the database and can send SQL commands to it using classes and interfaces from the “java.sql” package.
- To connect to a database we need to know the database URL
 - MySQL database URLs have the following format:
`jdbc:mysql://hostname:portnumber/databasename`
 - The default port is usually 3306, and if we are connecting to a database on the same machine then the host can be “localhost”

Making a Two Tier Connection

- Database connections are made using the static “getConnection” method of the DriverManager, which returns a Connection object

```
Connection connection = DriverManager.getConnection  
    ("jdbc:mysql://localhost/courses","javaclient","introjava");
```

- This code can be put into a “try-with-resources” construct, which ensures that the resources are closed if an exception occurs

```
try(Connection connection = DriverManager.getConnection  
    ("jdbc:mysql://localhost/courses","javaclient", "foundjava"))
```

- The com.mysql.cj.jdbc.Driver is the type 4 driver that the DriverManager will use in these examples, based on the jar file added to the modulepath

SQL Exceptions

- Most of the methods in the java.sql classes can throw java.sql.SQLException
- We must use try...catch blocks around our database access code

```
try(Connection....)
{
    // JDBC code
}
catch(SQLException e)
{
    //...handle exception
}
```

Creating Statements

- Statements are created from the `createStatement` method of the connection

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

- With this `Statement` object we can execute SQL commands such as `SELECT`, `INSERT`, `UPDATE` and `DELETE` against the database, wrapped in Java code

Reading With 'executeQuery'

- We can read data from the database by using the 'executeQuery' method of the statement
- This is passed an SQL query as a parameter, and returns a ResultSet object that contains the result of the query
 - A SELECT statement to retrieve all the rows from the 'course' table:

```
ResultSet results = statement.executeQuery("SELECT * FROM course");
```

ResultSets

- A ResultSet contains the data that has been read using the SELECT query
- We can get the data from it by iterating over it with a 'while' loop
 - Each iteration gives us a row from the database query

```
while(results.next())  
{  
    // process the next row returned by the query  
}
```

ResultSet Methods

- ResultSets have methods to retrieve different types of data
- They begin with 'get', followed by a data type
 - getString, getInt, etc
 - The parameter to these methods is either the column number or the column name

```
String courseName = results.getString(2);
```

- Using the column name is more reliable as the schema may change

```
String courseName = results.getString("name");
```

ResultSet Example

```
ResultSet results = statement.executeQuery("SELECT * FROM course");
...
while(results.next())
{
    id = results.getInt("courseID_PK");
    name = results.getString("name");
    numberOfDays = results.getInt("numberofdays");
    cost = results.getDouble("costperperson");
    publicSchedule = results.getBoolean("publicschedule");
    lastUpdated = results.getDate("lastupdated");
    System.out.println(id + "\t" + name + "\t" + numberOfDays + "\t" + costperperson + "\t" +
        publicSchedule + "\t" + lastUpdated);
}
```


Closing Objects

- ResultSets, Statements and Connections should all be closed when they are finished with
- Although the “try-with-resources” construct closes the connections to resources automatically, if you want to close them manually to free up resources they should be closed in the reverse order that they were opened
 - ResultSet first
 - then Statement
 - then Connection

```
results.close();  
statement.close();  
connection.close();
```

Exercise 15.1

- Create the 'courses' database in MySQL and populate it using the 'courses.ddl' file
- Write a Java class with a 'main' method that connects to your 'courses' data-base
- Execute a query to read all the rows in the 'module' table into a ResultSet
- Iterate through the ResultSet and write the data to standard output

Exercise 15.2

- Write a Java class with a 'main' method that connects to your 'courses' database
- Execute a query to read all the rows in the 'course' table where the cost is less than 1000 into a ResultSet
- Iterate through the ResultSet and write the data to standard output

Updating Records

- Update queries, which change the data by inserting, updating or deleting records, are supported by the 'executeUpdate' method of the 'Statement' class
- They do not return a ResultSet
- The return value (an int) contains the number of rows affected

```
rowsUpdated = statement.executeUpdate  
    ("INSERT INTO course (name, numberofdays, cost, publicschedule, lastupdated)  
    VALUES ('Python for Snake Charmers',4,1500.00,true,'2012-12-11')");
```

Updating and Deleting

- We can also update and delete using the executeUpdate method
- UPDATE

```
rowsUpdated = statement.executeUpdate  
("UPDATE course SET publicschedule=true WHERE courseID_PK=2");
```

- DELETE

```
rowsUpdated = statement.executeUpdate  
("DELETE FROM course WHERE name='Introduction to Java'");
```

Exercise 15.3

- Write a Java class with a 'main' method that connects to your 'courses' database
- Execute an update that adds a new course to the database with one new module
- Ensure that the foreign key from the module to the course is set correctly

Prepared Statements

- Prepared Statements are useful (and efficient) where similar queries are to be executed with different data
- The SQL string used as the parameter to the `prepareStatement` method has one or more placeholders where data can be provided
- These are indicated by question marks

```
PreparedStatement preparedStatement = connection.prepareStatement  
("UPDATE course SET lastupdated = ? WHERE courseID_PK = ?");
```

Using Prepared Statements

- To use a prepared statement, each placeholder is populated using 'set' methods based on the data type of the column

```
LocalDate localDate = LocalDate.of(2021, 4, 25);  
java.sql.Date sqlDate = java.sql.Date.valueOf(localDate);  
preparedStatement.setDate(1, sqlDate);  
preparedStatement.setInt(2, 2);  
preparedStatement.executeUpdate();
```

- The same prepared statement can be used multiple times

```
LocalDate localDate = LocalDate.of(2022, 3, 5);  
java.sql.Date sqlDate = java.sql.Date.valueOf(localDate);  
preparedStatement.setDate(1, sqlDate);  
preparedStatement.setInt(2, 254);  
preparedStatement.executeUpdate();
```


PreparedStatement Reuse

- This reuse not only makes our Java code more elegant, it actually makes it much more efficient
- The underlying SQL code that a prepared statement uses only needs to be generated once, rather than each time as it would be for individual Statement objects.

Exercise 15.4

- The PreparedStatement class has an 'executeQuery' method for executing SELECT statements
- Modify your code to use a PreparedStatement to query individual courses based on their name

Summary

- Creating and populating databases using MySQL
- Providing database access rights to an authorized MySQL user
- Using a JDBC driver for Java code to connect to and interact with a relational database using a standard API
- Using try-with-resources when making a database connection
- Executing queries to create a ResultSet
- Executing database updates (insert, update, delete)
- Creating PreparedStatement for common types of interaction with the database