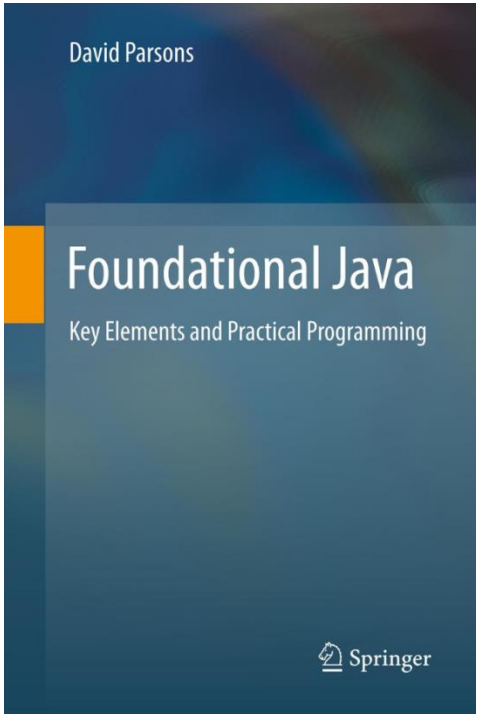


Chapter 15
Java and the Database (JDBC)

Foundational Java

Key Elements and Practical Programming

Persistent Storage

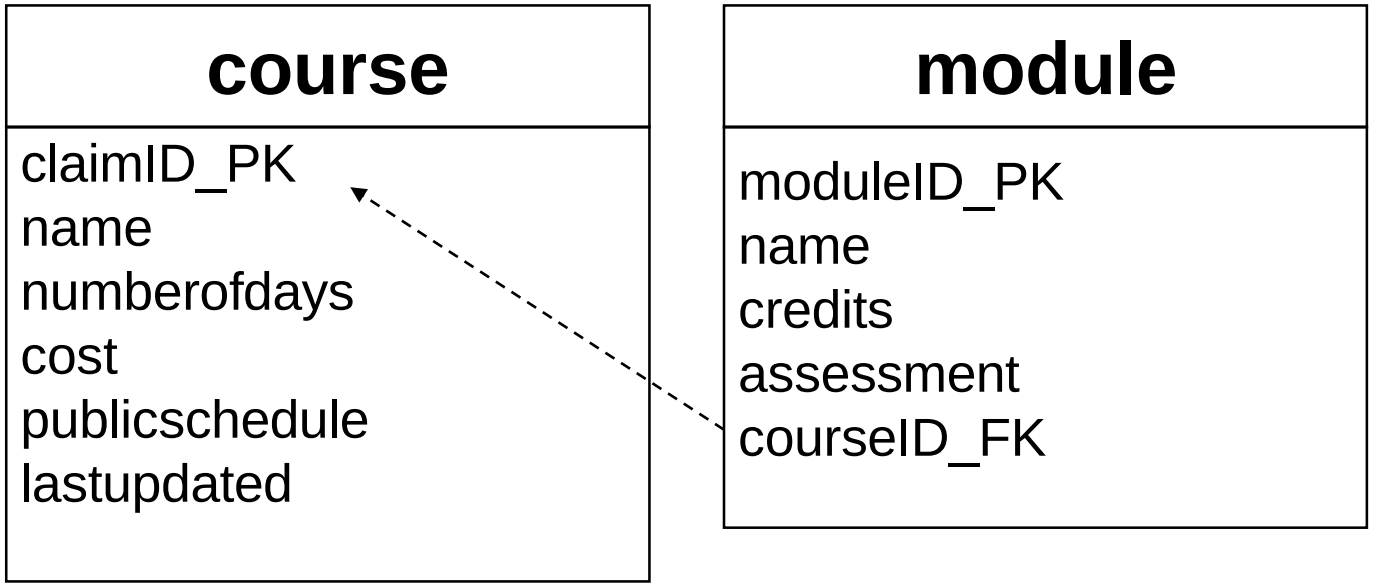
- The data in Java programs does not last beyond a single run of a program
- Most applications require some more persistent storage of state than this
  - In most cases we will use a database
  - Sequential files do not allow the querying, performance, availability and accessibility of database storage

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

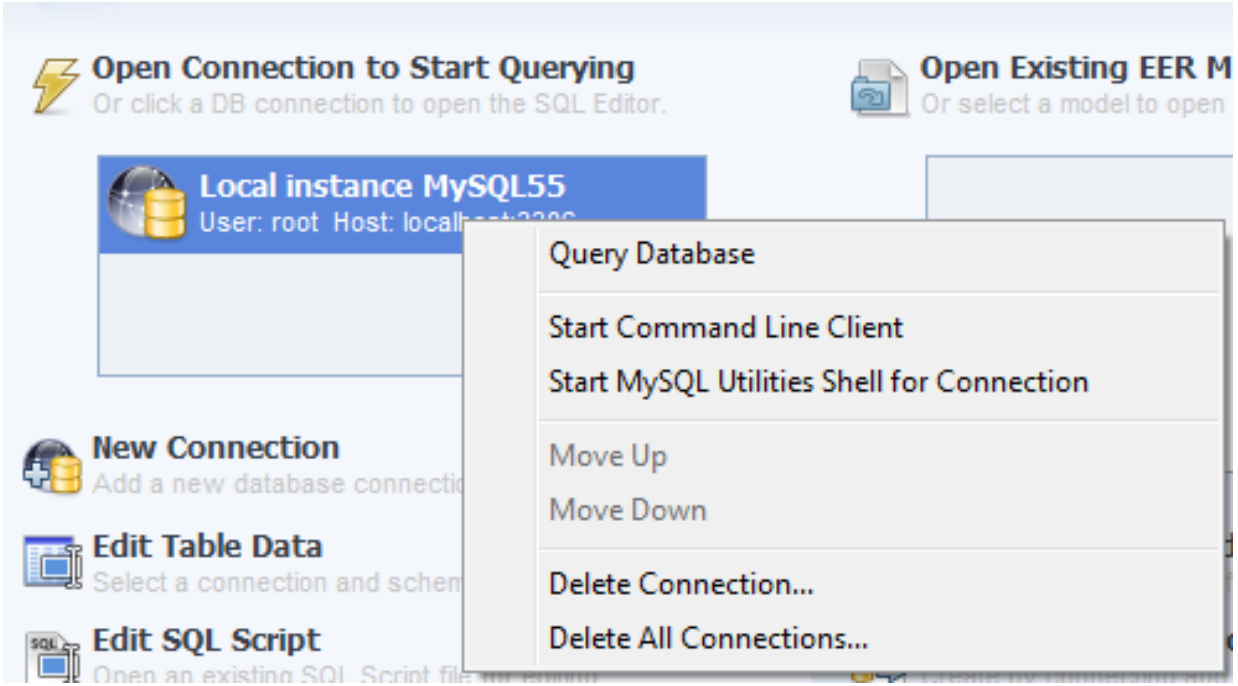


Using MySQL

- 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

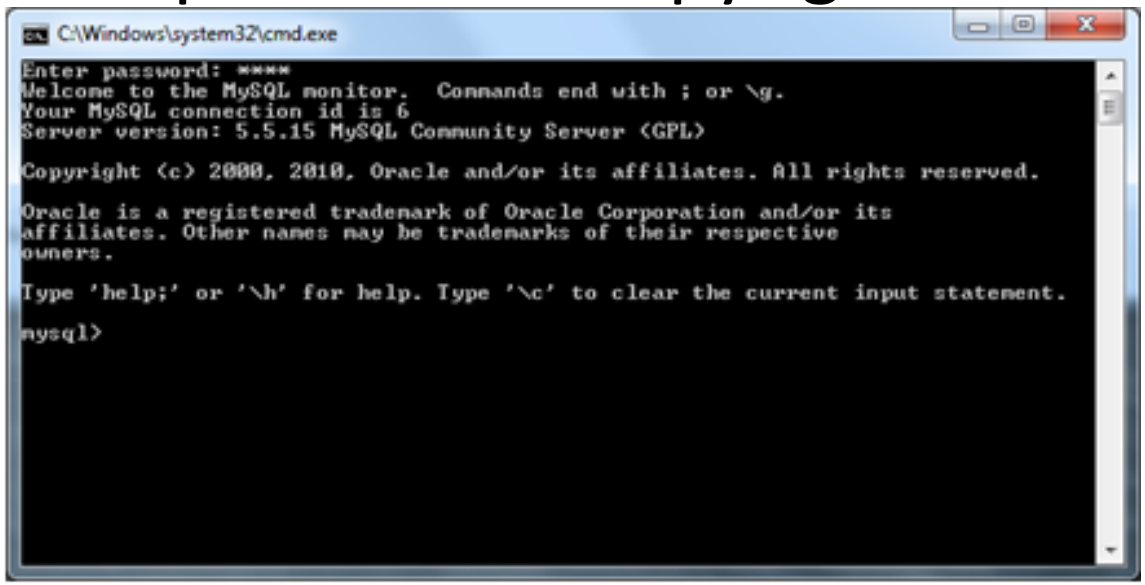
MySQL Workbench

- When MySQL is installed it includes the MySQL workbench
  - One of the options is to start the Command Line Client



Command Line Client

- The command line client will ask you for the password
- If you log in successfully you will see a ‘mysql’ prompt after the copyright notices



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           |
+--------------------+
| information_schema |
| courses            |
| mysql              |
| performance_schema |
| sakila             |
| test               |
| 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 (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

```
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
);
DROP TABLE module;
CREATE TABLE module (
 moduleID_PK INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
 name VARCHAR(35),
 credits INTEGER,
 assessment VARCHAR(20),
 courseID_FK INTEGER
);
INSERT INTO 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

- Grant access privileges to a user identified by a username and a password
  - Create, update and delete across all tables in the 'courses' database
    - Username 'javaclient'
    - Password 'introjava'
    - Database URL server 'localhost'

```
grant all privileges on courses.* to 'javaclient'@'localhost' identified by 'introjava';
```

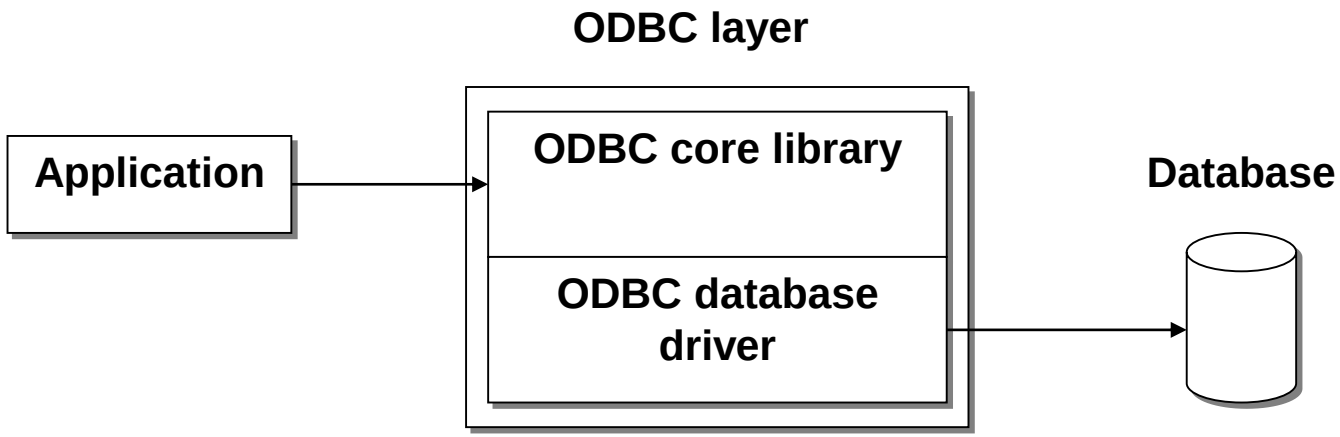
  - Can now connect to 'courses' as username 'javaclient' with password 'introjava'

MySQL Commands

MySQL Command	Meaning
`create database dbname`	Create a new database
`use dbname`	Connect to an existing database
`show databases`	Show the names of all databases
`show tables`	Show the names of all tables in the current database
`exit / quit`	Both exit from MySQL
`source path/filename`	Execute a SQL script file
`describe tablename`	Show the schema of a table
`GRANT privilege_type ON dbname.tablename TO 'username'@'server' IDENTIFIED BY 'password';`	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

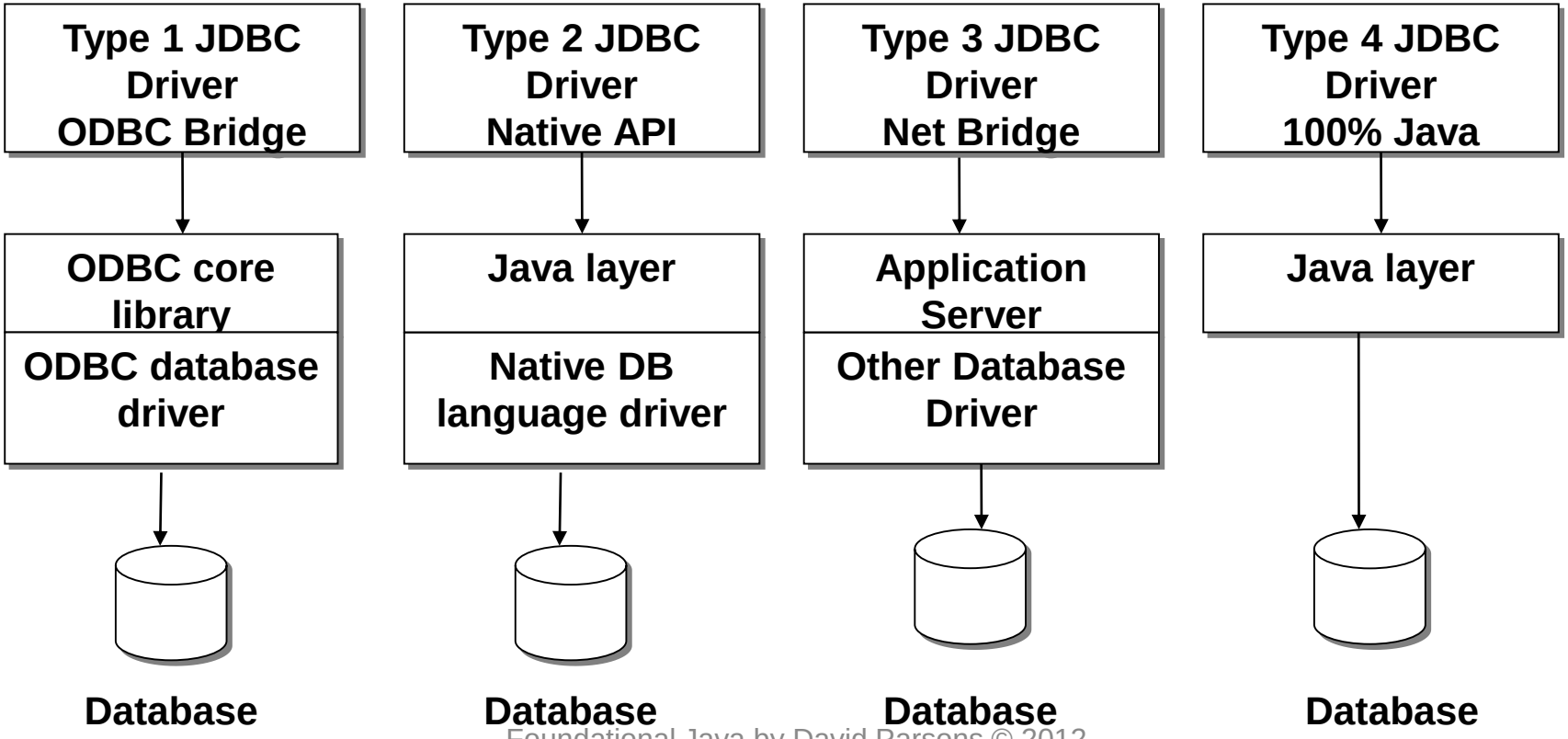


JDBC API

- 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

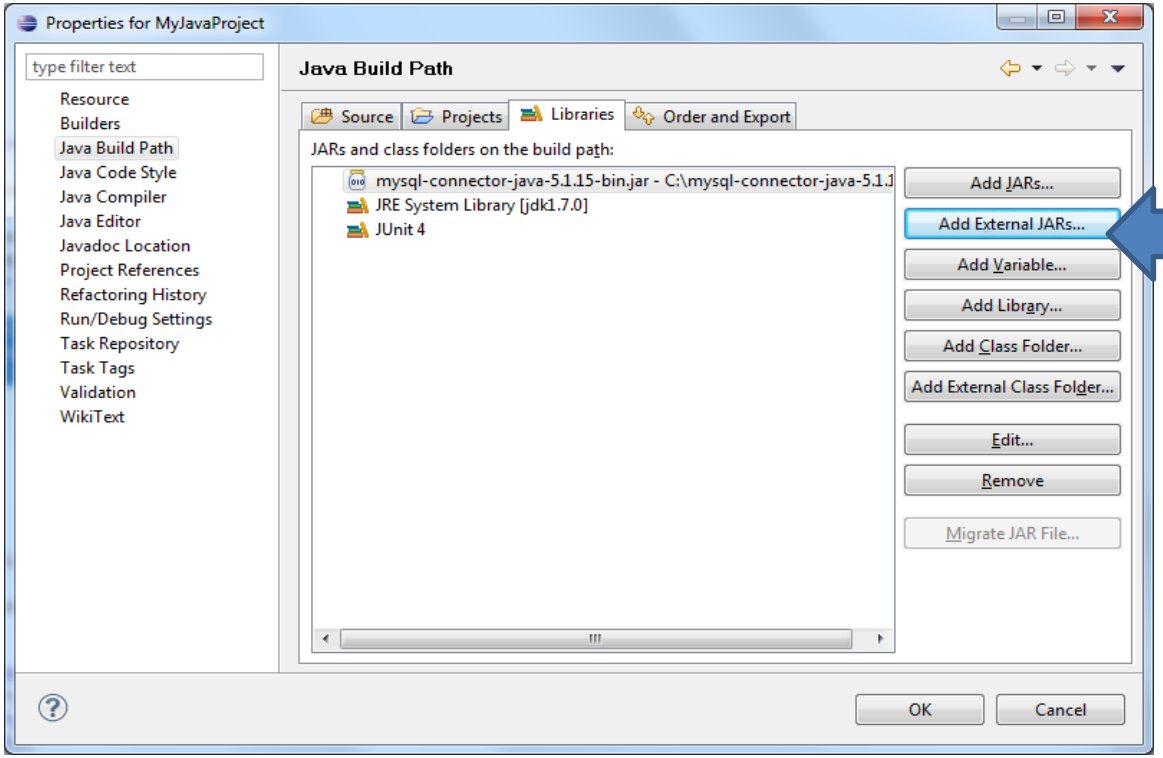


MySQL Connector Driver

- 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-5.1.15-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;.
```

Making a Two Tier Connection

- Dynamically load the driver

```
Class.forName("drivername");
```

  - e.g. to load the MySQL driver

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

- May throw a 'ClassNotFoundException'

```
try
{
    Class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException e)
{…}
```

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

Connecting To A Database

- Need to know the database URL
  - MySQL database URLs have the following format

```
jdbc:mysql://hostname:portnumber/databasename
```

  - Default port is 3306
  - A host on the same machine is ‘localhost’
- Use a static getConnection methods of the DriverManager class
  - This returns a ‘Connection’ object

```
Connection con = DriverManager.getConnection("jdbc:mysql://localhost/courses");
```

Authorized User

- A username and password to can be passed as additional parameters to the ‘getConnection’ method

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

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("cost");
  publicSchedule = results.getBoolean("publicschedule");
  lastUpdated = results.getDate("lastupdated");
  System.out.println(id + "\t" + name + "\t" + numberOfDays +
   "\t" + cost + "\t" + publicSchedule + "\t" + lastUpdated);
}
```

Closing Objects

- ResultSets, Statements and Connections should all be closed when they are finished with
  - close these objects in the reverse order that they were opened

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

- Closure of these resources is automatic
  - Explicit closure can free resources more quickly

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’ data-base
- 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 prepstatement = 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

```
long currentDateTime = Calendar.getInstance().getTimeInMillis();
prepstatement.setDate(1, new java.sql.Date(currentDateTime));
prepstatement.setInt(2, 1);
prepstatement.executeUpdate();
```

- The same prepared statement can be used multiple times

```
long currentDateTime = Calendar.getInstance().getTimeInMillis();
prepstatement.setDate(1, new java.sql.Date(currentDateTime));
prepstatement.setInt(2, 254);
prepstatement.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
- Using a JDBC driver for Java code to connect to and interact with a relational database using a standard API
- Executing queries to create a ResultSet
- Execution of updates
- PreparedStatements for common types of interaction with the database