JDBC

Agenda

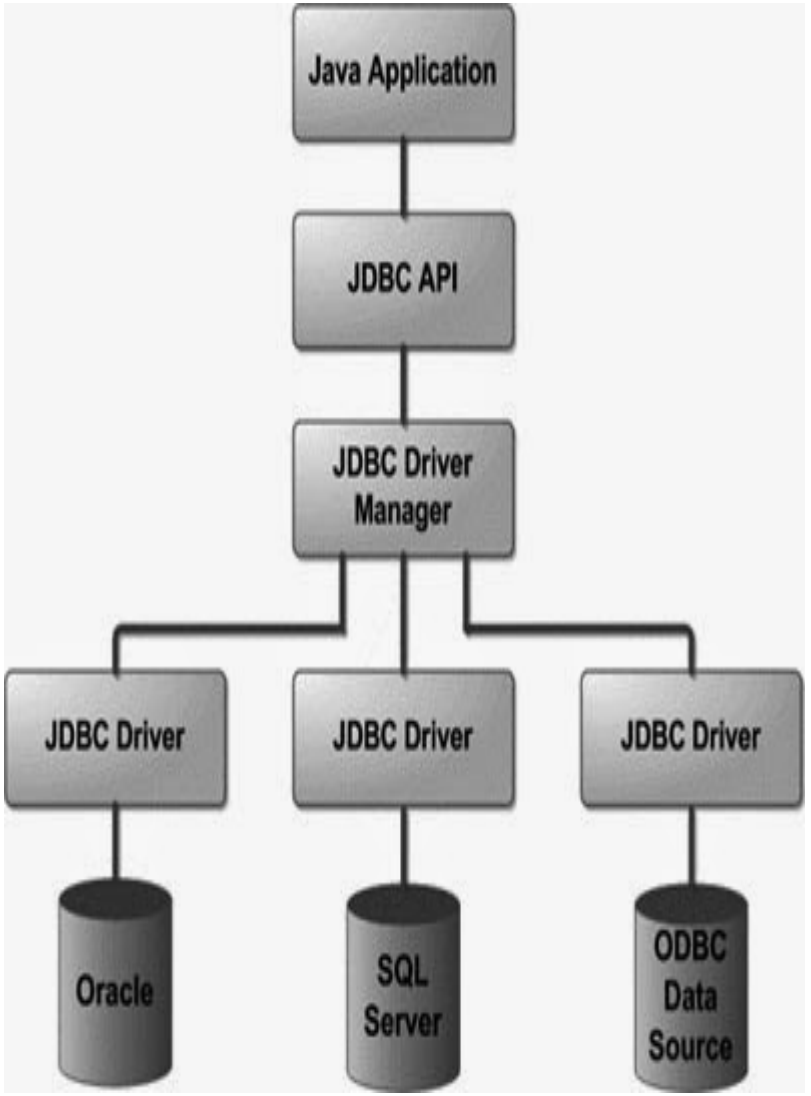
- What is JDBC
- JDBC components
- JDBC interface (API)
- Main classes of java.sql package:
  - Driver
  - Connection
  - Statement
  - ResultSet
  - PreparedStatement
- Briefly about SQL
- SQL injections
- Transactions

JDBC

- JDBC (Java Database Connectivity) is a Java database connectivity technology
- It provides API for querying and updating data in a relational database.
- JDBC supports ANSI SQL-2 databases, but can use any database, which has JDBC driver

JDBC components

- JDBC consists of following components:
  - **JDBC API** — provides programmatic access from Java o database
  - **JDBC Driver Manager** — defines objects which can connect Java applications to a JDBC driver.
  - **JDBC Test Suite** — helps to determine that JDBC drivers will run your program.
  - **JDBC-ODBC Bridge** — provides JDBC access via ODBC (ODBC (Open Database Connectivity developed by Microsoft) drivers



JDBC interface

- JDBC (Java Database Connectivity) interface (API) provides:
  - Connection to the database
  - Creation of SQL statements
  - Invocation of SQL statements, actual execution is performed by database
  - Retrieval of returned results

java.sql package

- Java.sql package contains following main classes
  - java.sql.Driver
  - java.sql.Connection
  - java.sql.Statement
  - java.sql.PreparedStatement
  - java.sql.CallableStatement
  - java.sql.ResultSet
  - java.sql.ResultSetMetaData
  - java.sql.DatabaseMetaData

JDBC example

```
import java.sql.*;

public class JDBCSample {
  public static void main( String args[]) {
    Connection con = null;
    try {
      Class.forName("com.mysql.jdbc.Driver"); // Load the driver class.


       con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/test","user", "password");  //Create connection

      Statement stmt = con.createStatement(); //Create a Statement


      //Execute the statement and store results in ResultSet object
      ResultSet rs = stmt.executeQuery("select moviename, releasedate from movies");


      while (rs.next())
        System.out.println("Name= " + rs.getString("moviename")); // Iterate through ResultSet

    } catch (Exception e) {
      System.err.println(e);
    } finally {
      con.close(); // Close the connection
    }
  }
}
```

Creation of JDBC driver instance

- Before connection to database it is necessary to load appropriate driver
- It is static class which is loaded on first invocation, passing its name as string:

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

- There is no need to assign returned value to variable

Creation of JDBC connection

- After JDBC driver class is loaded, JDBC connection can be established

```
Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/test","user",
"password");
```

- All other database management is done using returned Connection object
- To get rid of warning:

```
WARN: Establishing SSL connection without server's
identity verification is not recommended... You need
either to explicitly disable SSL by setting useSSL=false…
```

add **../?autoReconnect=true&useSSL=false** in URL

SQL statement

- Database operations are performed using Statement object:

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(
"select * from mytable");
```

- As Java is strongly typed language, it has three methods with different names and returned types:
  - **boolean execute()** — **true** if the first result is a ResultSet object; **false** if it is an update count or there are no results
  - **ResultSet executeQuery()** — **ResultSet** of Select operation
  - **int executeUpdate()** — **row count** for insterted, updated or deleted records, or **0** if statement return nothing

Select statement

- Select statement is used to retrieve records frome from one or more tables
- General syntax:

  SELECT <column(s) > FROM <table(s)> [WHERE <condition>] [ORDER BY <column(s) [ASC|DESC]>

- To execute Select statement **executeQuery()** command should be used
- Example:

```
ResultSet rs = st.executeQuery("select * from mytable");
```

Insert statement

- INSERT is used to insert new rows into an existing table.
- To execute Insert statement, **executeUpdate()** should be used
- General syntax:

  INSERT INTO <table> [(<column(s)>)] VALUES (<value(s)>)

- Example:

```
int res = st.executeUpdate("insert into
mytable values (1,'aa')");
```

Update statement

- UPDATE statement is used to update values of columns in existing rows of the table
- To execute Update statement executeUpdate() should be used
- General syntax:

  UPDATE <table> SET <column> = <value> [, <column> = <value>, …] [WHERE <condition>]

- Example:

```
int res = st.executeUpdate("update
mytable set mycolumn='aa' where id=1");
```

Delete statement

- Delete statement is used to delete records from the table
- To execute Delete statement executeUpdate() method should be used
- General syntax:

  DELETE FROM <table> [WHERE <condition>] ;

- Example:

```
int res = st.executeUpdate("delete from mytable where id=1");
```

Result handling

- Results which return more than 1 object are stored in ResultSet object
- ResultSet has internal iterator, which points to non-existing record before first record
- **ResultSet.next()** move iterator to next record, and if there is no any, ir returns false. Otherwise it points to first record
- Different **getXxx()** methods allow to get values from fields of one record
- Example:

```
while (rs.next()) {
  System.out.println("Customer: " +
rs.getString(2));
  System.out.println("Id: "  + rs.getString(1));
  System.out.println("");
}
```

getXxx() methods of ResultSet

Method	Returned type
getBinaryStream()	java.io.InputStream
getBoolean()	boolean
getByte()	byte
getBytes()	byte[]
getDate()	java.sql.Date
getDouble()	double
getFloat()	float
getInt()	int
getLong()	long
getShort()	short
getString()	java.lang.String
getTime()	java.sql.Time

SQL injections

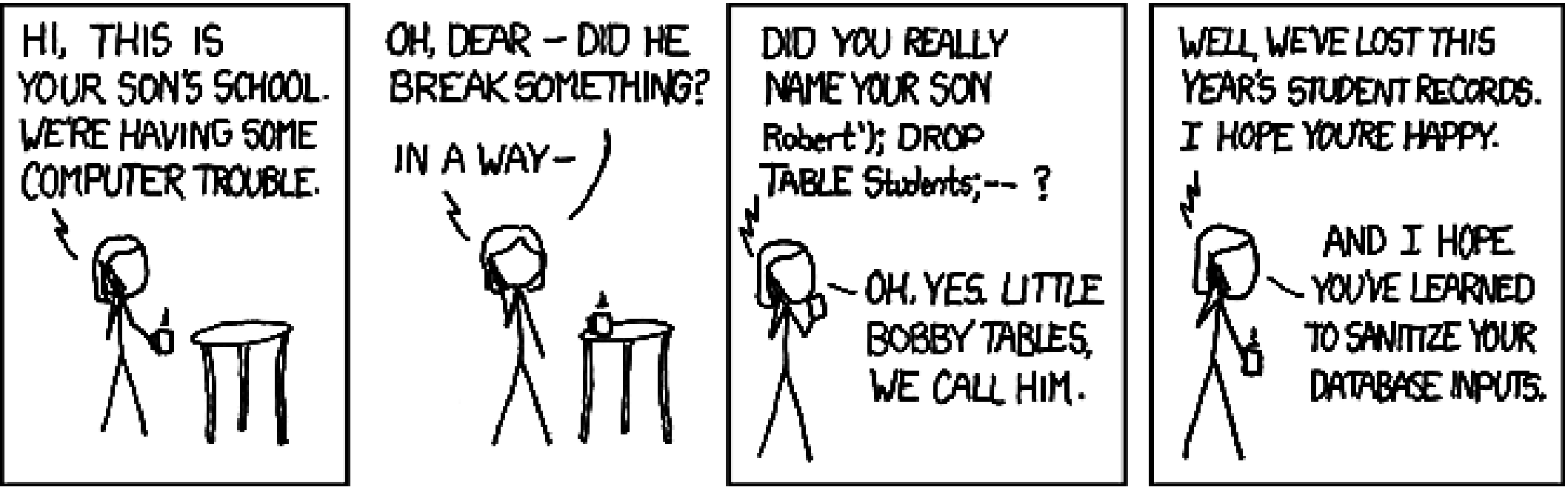
- If SQL statements are prepared using naive string concatenation

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("st =
"SELECT * FROM users WHERE name = '" +
userName + " and password = '" +
password + "';"");
```

- user can submit (inject) malicious data which performs unwanted statements

```
' or '1'='1
' or '1'='1';/*'
'; SELECT * FROM passwords;'
1;DROP database;
```

Exploit of a mom



https://xkcd.com/327/

Prepared statements

- Prepared statements are used to:
  - Improve performance by converting string to pre-processed bytecode form
  - Avoid SQL injections by allowing to pass only valid type of parameters

```
String sql = "update people set username=? where id=?";
PreparedStatement preparedStatement =
        connection.prepareStatement(sql);

preparedStatement.setString(1, "Gary");
preparedStatement.setLong  (2, 123);

int rowsAffected = preparedStatement.executeUpdate();
```

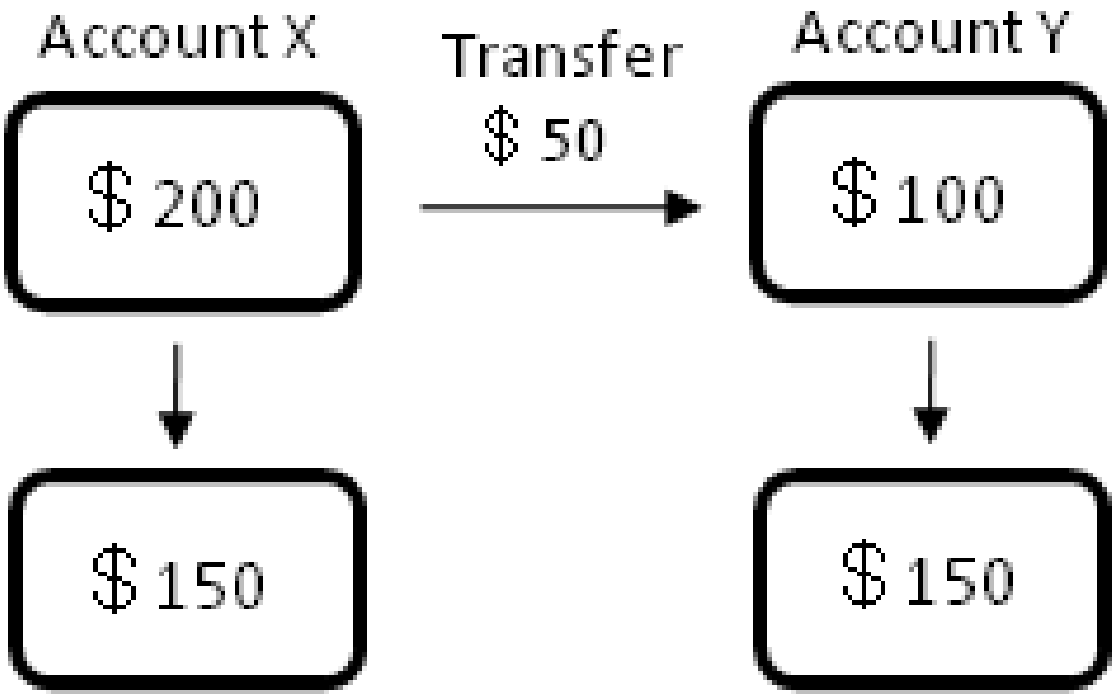
- If "'; DROP DATABASE;" parameter value is passed, it is properly escaped that username is updated to: "'; DROP DATABASE;"

setXxx() methods of PreparedStatement

Method	Converted SQL type
setBinaryStream()	LONGVARBINARY
setBoolean()	BIT
setByte()	TINYINT
setBytes()	VARBINARY vai LONGVARBINARY
setDate()	DATE
setDouble()	DOUBLE
setFloat()	FLOAT
setInt()	INTEGER
setLong()	BIGINT
setShort()	SMALLINT
setString()	VARCHAR vai LONGVARCHAR
setTIME()	TIME

Transaction

- A **transaction** is a set of one or more SQL statements that make up a logical unit of work
- It is ether fully executed or not executed at all
- How you "not execute" partially executed task?



Transaction example

```
try {
    con.setAutoCommit(false);
    updateSales =
con.prepareStatement(statementString);
    con.commit();  // commit successful
} catch (SQLException e ) {
    con.rollback(); // rollback
} finally {
    con.setAutoCommit(true);
}
```

ACID operations

- **A**tomicity
  transaction is series of database operations that either all occur, or nothing occurs
- **C**onsistency
  transactions change affected data only in allowed ways, database constraints are never violated
- **I**solation
  changes made by one transaction become visible to other users only when fully done
- **D**urability
  committed transactions change data permanently and allow data restoration in case of crash