Coding Bootcamp

Σύνδεση στην βάση δεδομένων μέσω JDBC API

Τι είναι το JDBC;

Το **JDBC** (**J**ava **D**ata**b**ase **C**onnectivity) είναι μια διεπαφή προγραμματισμού εφαρμογών Java (**Java API**) μέσω της οποίας ο χρήστης-προγραμματιστής μπορεί να συνδεθεί από το την Java Εφαρμογή του (πχ Java Application, Servlets, JSP, Applets κτλ) σε ένα μεγάλο σύνολο βάσεων δεδομένων (πχ MySQL Server, SQL Server, Oracle, MS Access κα) και να εκτελέσει SQL Statements.

Ανάλογα με τον Database Server που βρίσκεται η Βάση σας θα πρέπει να κατεβάσετε και την αντίστοιχη έκδοση του JDBC. Στην υπάρχουσα υποδομή του εργαστηρίου η Βάση σας βρίσκεται στον MySQL Server, για τον λόγο αυτό θα πρέπει να κατεβάσετε το JDBC για MySQL.

Τι είναι το JDBC; (2)

Το **JDBC** αποτελείται από δυο τμήματα:

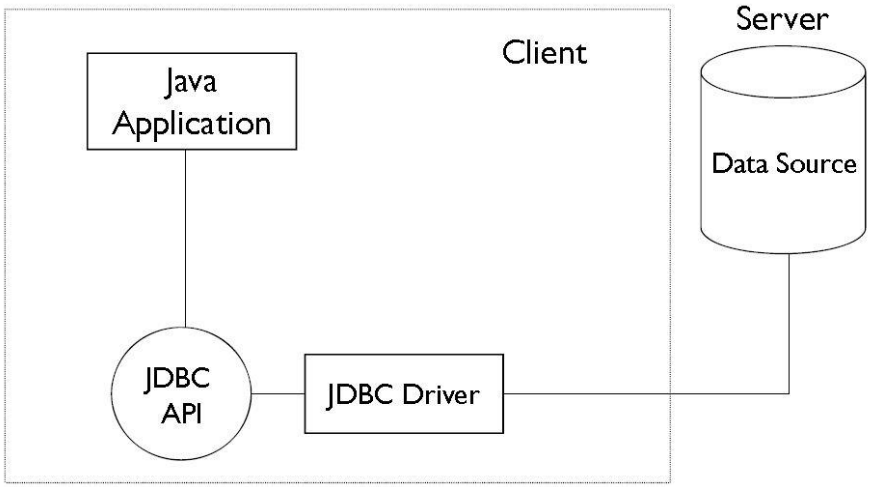
• Το **JDBC API** το οποίο είναι καθαρά ένα java-based API (**A**pplication **P**rogramming **I**nterface) και περιλαμβάνει τα κατάλληλα συστατικά και τις μεθόδους για:

- σύνδεση/αποσύνδεση με βάση δεδομένων
- δημιουργία και εκτέλεση SQL Statements
- συλλογή και επεξεργασία αποτελεσμάτων (query results)

• Τον **JDBC Driver Manager** ο οποίος «μεταφράζει» (μετατρέπει) τις κλήσεις από το JDBC API στην κατάλληλη «μορφή» (πρωτόκολλο) έτσι ώστε να είναι «κατανοητές» από τον Database Server (στην περίπτωση μας τον MySQL Server).

JDBC (Pure Java Driver) - Αρχιτεκτονική

Type 4 Driver - Database-Protocol Driver (Pure Java Driver)



Πλεονεκτήματα

- είναι η πιο αποτελεσματική μέθοδος για πρόσβαση στη βάση δεδομένων, τόσο από πλευράς απόδοσης-ταχύτητας (performance) όσο και από πλευράς ανάπτυξης (development).
- δεν απαιτείται εγκατάσταση κάποιου άλλου ειδικού λογισμικού στον client ή στον server.
- ιδανικός για διαδικτυακές εφαρμογές (web applications)

Μειονεκτήματα

- απαιτεί διαφορετικό driver για διαφορετική βάση (άλλος driver για MySQL, άλλος για MS Access κτλ)

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (1)

Τα **βήματα** που ακολουθούμε για να μπορέσουμε από ένα Java πρόγραμμα να συνδεθούμε στην βάση μας (MySQL), να εκτελέσουμε SQL Statements (INSERT, DELETE, UPDATE, SELECT κτλ) και να λάβουμε/επεξεργαστούμε τα αποτελέσματα (query results) συνοψίζονται στα παρακάτω:

Βήμα 1ο *(import JDBC API)*:

Στο πρόγραμμά μας κάνουμε **import** το **JDBC API**, αυτό γίνεται τις πιο πολλές φορές με την προσθήκη του παρακάτω κώδικα:

```
import  java.sql.*;
```

Βήμα 2ο *(Load the JDBC driver)*:

«Φορτώνουμε» τον JDBC driver στην μνήμη, μέσω του driver θα ανοίξουμε δίαυλο επικοινωνίας (communication channel) με την βάση δεδομένων.

```
try {
   Class.forName("com.mysql.jdbc.Driver").newInstance();
 } catch (Exception e) {
   …//handle the exception or/and print message
}
```

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (2)

Βήμα 3ο *(Register driver and establish the connection)*:
Εδώ απαιτείται η χρήση της μεθόδου **DriverManager.getConnection()**, η μέθοδος αυτή δέχεται τρία ορίσματα (ή άλλα ? γιατί είναι overloaded μέθοδος) όλα τύπου String και επιστέφει ένα αντικείμενο τύπου **Connection** το οποίο «αντιπροσωπεύει» την συνεδρία (Session) της επικοινωνίας με την βάση δεδομένων. Μέσω των μεθόδων του αντικειμένου **Connection** θα εκτελούμε τα SQL Statements στην βάση μας (βλέπε Βήμα 4ο). Η μέθοδος **DriverManager.getConnection()** παράγει **SQLException** για αυτό θα πρέπει να την συμπεριλάβουμε σε try – catch block.

Παράδειγμα:

```
Connection con;
try {
   con = DriverManager.getConnection(
   "jdbc:mysql://195.251.249.131:3306/dbname", "username", "password");
 } catch (SQLException e) {
   …//handle the exception or/and print message
}
```

Διεύθυνση του Database Server (πχ localhost)

Θύρα επικοινωνίας

Όνομα βάσης

Χρήστης

Κωδικός χρήστη

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (3)

Βήμα 4ο *(Create and execute SQL Statement(s))*:
Το **Connection** είναι μια διεπαφή (Interface) και αντιπροσωπεύει την συνεδρία της σύνδεσης με την βάση δεδομένων. Μέσω των μεθόδων του Connection μπορούμε να εκτελούμε τα SQL Statements στην βάση μας και να κλείνουμε την συνεδρία. Στην συνέχεια θα χρησιμοποιούμε τρεις από τις μεθόδους του Connection:

• η μεθόδου **createStatement() «**δεν δέχεται ορίσματα» (προσοχή overloaded) και επιστρέφει το αντικείμενο **Statement** μέσω του οποίου μπορούμε να εκτελέσουμε SQL Statements (πχ SELECT, INSERT, DELETE, UPDATE κτλ) στην βάση δεδομένων.

• η μέθοδος **prepareStatement()** δέχεται όρισμα String (προσοχή overloaded) το οποίο περιέχει το SQL Statement που θέλουμε να εκτελέσουμε (σε παραμετρική μορφή) και επιστρέφει το αντικείμενο **PrepareStatement**. Στη συνέχεια ορίζουμε τις παραμέτρους στο **PrepareStatement** και το εκτελούμε στην βάση δεδομένων.

• η μέθοδος **close()** κλείνει την συνεδρία (σύνδεση) με την βάση δεδομένων απελευθερώνοντας ταυτόχρονα και τους πόρους που έχει δεσμεύσει το σύστημα. Πρέπει να καλείται στο τέλος.

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (4)

Χρησιμοποιούμε το αντικείμενο **Statement** ή το **PreparedStatement** για να υποβάλλουμε-εκτελέσουμε το SQL Statement που επιθυμούμε στην βάση. Πρώτα βάζουμε το SQL Statement που θέλουμε να εκτελέσουμε σε ένα String και στην συνέχεια χρησιμοποιούμε τις μεθόδους **executeUpdate**() ή **executeQuery**() του αντικειμένου **Statement** ή **PreparedStatement** ως εξής:

- χρησιμοποιούμε την μέθοδο **executeUpdate**() όταν θέλουμε να εκτελέσουμε SQL Statement τύπου **INSERT**, **DELETE** ή **UPDATE**.

- χρησιμοποιούμε την μέθοδο **executeQuery**() όταν θέλουμε να εκτελέσουμε SQL Statement τύπου **SELECT** (query).

Παράδειγμα με Statement:

```
Statement stmt = null;
String sql1 = "DELETE FROM student where am= '8020067';";
try {
   stmt = con.createStatement(); //initialize stmt Statement
   stmt.executeUpdate(sql1); //execute (SQL Statement) sql1
   stmt.close();   //close Statement stmt
 } catch (SQLException e) {
   …//handle the exception or/and print message
}
```

Παράδειγμα με PreparedStatement:

```
PreparedStatement stmt1 = null;
String sql2 = "DELETE FROM student where am= ?;";
try {
      stmt1 = con.prepareStatement(sql2);  //initialize stmt1
      stmt1.setString(1, "8020067"); //setting parameter
      stmt1.executeUpdate();       //execute (SQL Statement) sql2
      stmt1.close();                    //close Statement stmt1
 } catch (SQLException e) {
   …//handle the exception or/and print message
}
```

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (5)

Σε περίπτωση που θέλουμε να εκτελέσουμε **SELECT** (query) και όχι INSERT ή DELETE ή UPDATE, τότε (όπως αναφέραμε) χρησιμοποιούμε την μέθοδο **executeQuery**() (είτε του Statement ή του PreparedStatement). Η μέθοδος **executeQuery**() επιστρέφει το αντικείμενο **ResultSet,** μέσω των μεθόδων του Resultset συλλέγουμε και επεξεργαζόμαστε τα αποτελέσματα του query (βλέπε Βήμα 5).

Παράδειγμα SELECT με Statement

```
Statement stmt = null;
ResultSet rs = null;
String sql3 = "SELECT * FROM student where am='8020067';";
try {
   stmt = con.createStatement(); //initialize stmt Statement
   rs = stmt.executeQuery(sql3); //execute (SQL Statement) sql3
  …. //process the results (Βλέπε 5ο βήμα)
 stmt.close();   //close Statement stmt
 } catch (SQLException e) {
   …//handle the exception or/and print message
}
```

Παράδειγμα SELECT με PreparedStatement

```
PreparedStatement stmt4 = null;
 ResultSet rs = null;
String sql4 = "SELECT * FROM student where am= ?;";
try {
      stmt4 = con.prepareStatement(sql4);  //initialize stmt4
      stmt4.setString(1, "8020067"); //setting parameter
      rs = stmt4.executeQuerry();  //execute (SQL Statement) sql4
      … //process the results (Βλέπε 5ο βήμα)
      stmt4.close();                          //close Statement stmt4
 } catch (SQLException e) {
   …//handle the exception or/and print message
}
```

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (6)

Βήμα 5ο *(Extracting data from ResultSet object)*:

Σημείωση: Εκτελούμε το βήμα αυτό εάν στο Βήμα 4 έχουμε εκτελέσει SELECT, διαφορετικά παρακάμπτουμε το Βήμα 5 και προχωράμε κατευθείαν στο Βήμα 6.

Το αντικείμενο ResultSet είναι ένας «πίνακας» με δεδομένα ο οποίος αντιπροσωπεύει τα αποτελέσματα (Database result set) του query που εκτελέσαμε. Ο πίνακας έχει την παρακάτω δομή:

- κάθε στήλη του πίνακα περιέχει τις τιμές του πεδίου που αναφέραμε στο SELECT (query)
- στις γραμμές βρίσκονται οι τιμές των αντίστοιχων πεδίων.
- το ResultSet διαθέτει επίσης και έναν κέρσορα (cursor) ο οποίος μας δείχνει την τρέχουσα εγγραφή που διατρέχουμε. Στην αρχική του κατάσταση ο κέρσορας αυτός βρίσκεται πριν την πρώτη γραμμή του πίνακα.

Παράδειγμα 1

Πίνακας Product

pid	name	price
1	spaghetti	1.5
2	Cheese	7.2
3	Milk	1.8

```
...
ResultSet rs = stmt.executeQuery("SELECT pid, name FROM product where price < 5;");
```

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (7)

Παρακάτω παραθέτουμε μερικές από τις «βασικές» μεθόδους του αντικειμένου **ResultSet** που χρησιμοποιούμε για να προσπελάσουμε και να λάβουμε τα δεδομένα του.

- boolean **next**()

Είναι η «βασική» μέθοδος που χρησιμοποιούμε για να προσπελάσουμε τις εγγραφές (μια-μια). Μετακινεί τον κέρσορα στην επόμενη γραμμή, ενώ ταυτόχρονα επιστρέφει **true** εάν στην επόμενη γραμμή υπάρχει εγγραφή, διαφορετικά (δλδ βρίσκεται στο τέλος του πίνακα) επιστρέφει **false**. Η χρήση της γίνεται συνήθως σε συνδυασμό με while.

Παράδειγμα:|

```
while( rs.next() ) {
    …
}
```

- boolean **previous**()
- void **beforeFirst**()
- void **afterLast**()
- String **getString**(**String** columnLabel)
- int **getInt**(**String** columnLabel)
- double **getDouble**(**String** columnLabel)
- Date **getDate**(**String** columnLabel)
- void **close**()

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (8)

Παράδειγμα 2:

```
…
Statement stmt = null;
ResultSet rs = null;
String sql = "SELECT pid, name FROM product where price < 5;";
try {
   stmt = con.createStatement(); //initialize stmt Statement
   rs = stmt.executeQuery(sql); //execute query and store results to rs (ResultSet)
   while(rs.next()) {
       int productID = rs.getInt("pid");
       String productName = rs.getString("name");
       out.println("Product ID: " + productID + "<br>");
       out.println("Product Name: " + productName);
   }
   rs.close(); //close ResultSet object
   stmt.close();   //close Statement stmt
   …
 } catch (SQLException e) {
   …//handle the exception or/and print message
 }
```

Χρήση του JDBC - «Προγραμματιστική» διαδικασία (9)

Βήμα 6ο *(Clean-up environment and finally close connection with database)*:

Σε αυτό το σημείο κλείνουμε όλα τα **Statement** (ή/και **PreparedStatement**) και **ResultSet** (εάν δεν το έχουμε κάνει ήδη) που χρησιμοποιήσαμε απελευθερώνοντας έτσι τους πόρους του συστήματος (Tomcat Web Server).

Τέλος, κλείνουμε την σύνδεση με την Βάση δεδομένων (αντικείμενο Connection) αυτό γίνεται καλώντας την μέθοδο **close()** του αντικειμένου **Connection** (πχ con.close()).

Παράδειγμα:

```
try {
   …
  rs.close(); //κλείσιμο του ResultSet
  stmt.close(); //κλείσιμο του Statement (ή PreparedStatement)
  con.close(); //κλείσιμο συνεδρίας σύνδεσης με την βάση
} catch (SQLException e) {
   …//handle the exception or/and print message
}
```

Σημείωση: Αυστηρά στο ΤΕΛΟΣ κλείνουμε ΠΑΝΤΑ την σύνδεση με την βάση μας, διαφορετικά η συνεδρία της σύνδεση μένει ανοικτή και δεσμεύει (αδικαιολόγητα) πόρους του συστήματος μέχρι να τους απελευθερώσει ο JVM's garbage collector.

Statement vs PreparedStatement

Οι περισσότερες σχεσιακές βάσεις δεδομένων (όπως η MySQL) χειρίζονται τα SQL queries που εκτελούμε ακολουθώντας τα παρακάτω βήματα (διαδικασίες):

1. **Parse the incoming SQL query and compile it**: αναλύουν το query ως προς το συντακτικό (δλδ έχει σωστή σύνταξη;) και εξετάζουν εάν έχουμε την άδεια (permission) για να το εκτελέσουμε κτλ και το κάνουν compile.
2. **Plan/optimize the data acquisition path**: αξιολογούν-εξετάζουν (με βάση τα στατιστικά που κρατάνε) και βρίσκουν τον καλύτερο τρόπο από όλους τους δυνατούς (πχ εάν θα χρησιμοποιήσουν κάποιο ευρετήριο (index) ή θα κάνουν full table scan, ποιον αλγόριθμο θα χρησιμοποιήσουν κτλ).
3. **Execute query and return data**: εκτελούν το query και επιστρέφουν τα αποτελέσματα.

• Τα Statements κάθε φορά ακολουθούν τα παραπάνω βήματα 1-3, ενώ τα PreparedStatements (precompiled) προ-εκτελούν τα βήματα 1-2, με αυτόν τον τρόπο μειώνουν τον φόρτο στην βάση και εκτελούνται πιο γρήγορα.

• Τα queries στα Statements αποτελούνται (κατασκευάζονται) από concatenated Strings, για αυτό το λόγο είναι ευάλωτα στο SQL injection. Αντιθέτως, τα PreparedStatements (παραμετρικά) δεν είναι ευάλωτα στο SQL injection.

• Τα σφάλματα στα queries μέσω Statements γίνονται πιο εύκολα ανιχνεύσιμα (debugging - γιατί μπορούμε να τα τυπώσουμε) σε αντίθεση με τα queries στα (παραμετρικά) PreparedStatements.

Mapping Java Types to SQL Types

Ο JDBC driver «μεταφράζει» τα αντικείμενα και τύπους της Java (πχ int, double, float, String, Date κτλ) στους αντίστοιχους τύπους της SQL, για παράδειγμα ο ακέραιος (int) της Java θα μεταφραστεί σε SQL INTEGER. Στον παρακάτω πίνακας φαίνεται η αντιστοιχία των (πιο «συχνών») αντικειμένων και τύπων της Java και πως αυτοί μεταφράζονται από τον JDBC driver:

- σε τύπους της SQL
- όταν καλούμε την μέθοδο setXXX() (πχ *stmt.setInt(1, 16)*) του PreparedStatement για να περάσουμε τους παραμέτρους
- όταν καλούμε την μέθοδο getXXX() (πχ *rs.getString("surname")*) του ResultSet για να πάρουμε την τιμή του πεδίου (του πίνακα της βάσης).

JAVA	**PreparedStatement - setXXX(θέση, τιμή)**	**ResultSet - getXXX(**"όνομα στήλης"**)**	**SQL**
int	setInt	getInt	INTEGER
double	setDouble	getDouble	DOUBLE
String	setString	getString	VARCHAR, CHAR
float	setFloat	getFloat	FLOAT
java.sql.Date	setDate	getDate	DATE, DATETIME