מעבדה מספר 4

MySQL-ו JDBC

במעבדה זו נלמד כיצד לבצע חיבור למסד הנתונים MySQL באמצעות אפליקציה הכתובה ב- Java תוך שימוש ב- JDBC (Java Database Connectivity).

קבצי המעבדה כוללים קובץ זה עם הסבר ומשימה לביצוע ובנוסף:

קבצי jdbc_project עם תכנית בסיסית ליצירת connection עם database, קובץ SQL בשם bakupFlights, קובץ בשם mysql_commands עם פקודות MySQL ואפליקציית JAVA בשם mysql-connector-java-3.1.11-bin .

1. **רקע**

JDBC

באמצעות JDBC ניתן לכתוב קוד ב-Java המכיל בתוכו קוד המסוגל לפנות למסד נתונים ולבצע פעולות שונות ומגוונות ב-SQL. הקוד נכתב כולו בשפת Java ובניגוד לשפות Embedded SQL אינו מצריך פעולה נוספת של תרגום מקוד המקור לשפת התכנות (לדוגמא C Embedded SQL). Java מספקת API (Application Programming Interface) שלם לצורך עבודה עם מסדי נתונים. לצורך שימוש ב- API יש ליבא את חבילת java.sql.

באמצעות שימוש ב- JDBC ניתן לבצע מספר התחברויות בו זמנית למספר מסדי נתונים שונים או מספר התחברויות לאותו מסד דרך אותה אפליקציה.

כל ההתקשרות בין האפליקציה למסד נתונים ספציפי מתבצעת דרך תוכנה מתווכת אשר נקראת DBMS Driver (או בקיצור Driver), כאשר ה-Driver הוא האחראי להעברת פקודות מ- JDBC אל המסד ומהמסד בחזרה ל-JDBC. בדרך כלל טוענים Drivers באופן דינאמי על ידי רישום בזמן ריצה אצל DriverManager שאחראי על ניהול כל ה-Drivers הרצים במחשב המארח.

לאחר שרושמים את ה-Driver שאיתו משתמשים, ניתן ליצור חיבור למסד הנתונים וליצור אובייקט התקשרות למסד (Connection). מעתה כל הפקודות למסד הנתונים מתבצעות דרך אובייקט ההתחברות עד לסגירתו.

Java משתמשת בתבנית תכן של Object Pooling בכל הקשור לניהול אובייקטי ההתחברות מאחר ויצירת התחברות כרוכה בהשקעת משאבי מחשוב רבים (למשל: פרוטוקול תקשורת או משאבי זכרון).

הסבר על פעולת ה-JDBC

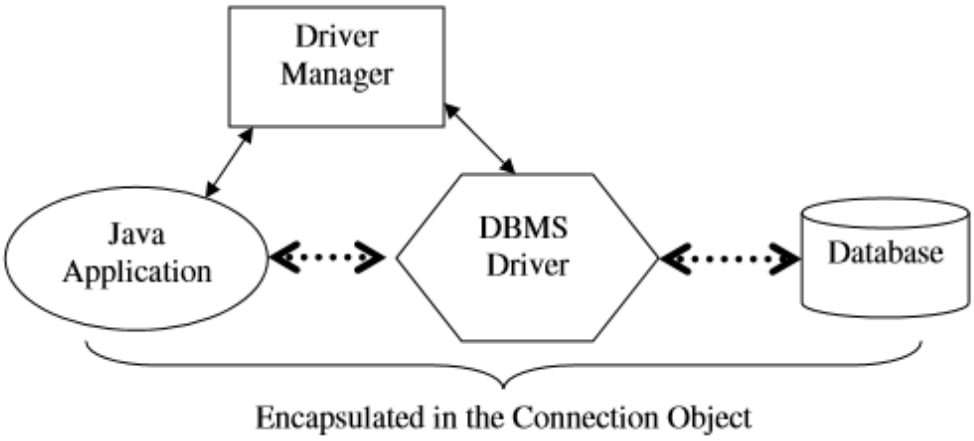
כדי להשתמש ב-JDBC יש ליצור **connection** ל-database, על ידי שימוש באוביקט מטיפוס Connection.אוביקטים מטיפוס Connection אופיינים לסוג ה-database, לדוגמא MySQL Connection או Oracle Connection.

באמצעות אובייקט Connection ניתן ליצור **Statement**. Statement הוא אוביקט פקודה שניתן לשימוש חוזר ובאמצעותו ניתן לבצע פקודות SQL ולקבל תשובות מה-database. ניתן לציין עבור attributes (כמו למשל כמה תוצאות הוא יכול להכיל). בנוסף, Connection יכול ליצור סוג אחר של Statement שנקרא PreparedStatement עם יכולת ביצוע טובה יותר ועם אבטחת מידע.

השאילתות "SELECT" SQL מחזירות תוצאות באובייקט מטיפוס **ResultSet**, המאפשר למתכנת לעבור של התוצאות בצורת איטרטיבית.

.

JDBC – ארכיטקטורה:



driver (בעברית **מנהל התקן** (Driver) הוא תוכנית מחשב המאפשרת לתוכנית מחשב אחרת, לתקשר עם חומרה כלשהי או עם תוכנה אחרת הפועלת בפורמט שונה באמצעות מימוש הממשק שלה ומתן API לעבודה מולו. להלן סיווג ה-drivers הקיימים לעבודה עם JDBC.

סיווג Driver-ים:

את כל ה Driver-ים הקיימים ניתן לסווג ל-4 טיפוסים:

1. Bridge – מתרגם קריאות של פונקציות ב-JDBC לקריאות ב-API אחר שגם הוא לא טבעי למסד (כלומר מבצע תרגום לשפה טבעית או פונה ל-API אחר גם). (למשל: ODBC-JDBC bridge).
2. Direct Translation to the Native API via none Java Driver- הקריאות של JDBC מתורגמות ישירות לקריאות של API בשפה טבעית למסד.
3. Network Bridge – סוג זה פונה לשכבת ביניים (middleware) שאחראית על התרגום (ע"ג רשת) ומעביר לה רק את הצהרות SQL ללא תרגום. התרגום נעשה בשכבת הביניים.
4. Direct Translation to the Native API via Java Driver- כאן התקשורת בין ה-Driver למסד הנתונים נעשית ישירות ע"י Java Sockets. סוג זה נכתב ישירות לשימוש במסד ספציפי. מאחר וגם מסד הנתונים כתוב ב- Java בד"כ שילוב זה מספק ביצועים די טובים.

2. <u>המעבדה מבוססת על מסד הנתונים הבא:</u>

"מסד חברת התעופה"

זהו מסד אקסס שיושב אצלכם מקומית במחשב.

בתרגול נשתמש בסכמה הבאה:

Aircraft(aid,aname, crusingrange)

Certifies(edi,aid)

Employees(eid,ename,salary)

Flights(flno,from.to,distance,depart,arrives,price)

לסכימה זו תוסיפו טבלה שמרכזת את מחירי המוצרים הנמכרים בדיוטיפרי.

ולהלן מופע של טבלה במסד:

Aircraft

aid	aname	crusingrange
1	Boeing 747-400	8430
2	Boeing 737-800	3383
3	Airbus A340-300	7120
4	British Aerospace Jetstream 41	1502
5	Embraer ERJ-145	1530
6	SAAB 340	2128
7	Piper Archer III	520
8	Tupolev 154	4103
9	Lockheed L1011	6900
10	Boeing 757-300	4010
11	Boeing 777-300	6441
12	Boeing 767-400ER	6475
13	Airbus A320	2605
14	Airbus A319	1805
15	Boeing 727	1504
16	Schwitzer 2-33	30

3. רישום ה-Driver:

```
Class.forName(<driver class path>);
```

אנחנו נשתמש ב-Driver שמגעי עם API של MySQL ולכן נרשום:

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

ניתן גם לרשום Driver ישירות דרך ה-DriverManager:

```
DriverManager.registerDriver(new mySQLDriver());
```

נתן גם להתחבר למבני נתונים אחרים, למשל Access של Microsoft. רישום דרייבר של אקסס מתבצע דרך המגרש ODBC דבר שמחייב בנוסף לרושמו במערכת ההפעלה.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

<u>דוגמא</u>: בצעו התחברות למסד MySQL:

```
String url = "jdbc:mysql://localhost/test";

String username = "Student";
String password = "University";

try {
      Connection conn = DriverManager.getConnection(url, username, password);
} catch(SQLException e) {
      System.err.println("Connection to Database failed: " + e.getMessage());
}
```

4. <u>יצירת JDBC Statement</u>

אובייקט Statement הוא זה ששולח את הצהרת SQL של משתמש אל המסד. ע"מ לבצע הצהרת SQL כל מה שצריך הוא ליצור אובייקט חדש ולהשתמש בפונ' Statement.executeXXX() המתאימה (כלומר יש להחליף את XXX בפעולה הנדרשת: Update או Query) על מנת ליצור אובייקט Statement יש לפנות אל אובייקט ההתקשרות ולקבל ממנו התייחסות לאובייקט(reference).

<u>דוגמא</u>: נניח שטבלת GIFTS לא קיימת במסד ונרצה ליצור אותה (כמובן בהנחה נוספת שיש לנו הרשאה במסד ליצירת טבלאות):

```
//create new statement using Connection Factory Method
Statement stmt = conn.createStatement();

String createTableGifts = "CREATE TABLE GIFTS " +
                          "(GIFT_NAME VARCHAR(32), SUP_ID INTEGER," +
                          "PRICE FLOAT, SALES INTEGER, TOTAL INTEGER";
//execute create statement
stmt.executeUpdate(createTableGifts);
```

פעולות על מבני נתונים מתחלקות לשני סוגים:

DDL – פעולות יצירה וביטול של טבלאות. (Data Definition Language)

DML – פעולות לעדכון נתונים בתוך הטבלאות (מחיקה, הוספה, עדכון) (Data Manipulation Language)

שני הסוגים של פעולות מבוצעות על ידי אותה מתודה שבקוד הנ"ל.

דוגמא להכנסת נתונים לטבלה:

```
Statement stmt = conn.createStatement();
stmt.executeUpdate("INSERT INTO GIFTS VALUES " +
                   "('Diamond', 101, 2000, 0, 0)");
```

5. **שליפת נתונים מתוך טבלה**

בשימוש באובייקט מטיפוס Statement ניתן גם להריץ שאילתות למסד ולקבל את תוצאותיהן.

היות ותוצאת שאילתא ב-SQL היא אוסף של רשומות, התוצאה מוחזרת כאובייקט מסוג ResultSet. אובייקט זה מאפשר בצורה נוחה לגשת לכל רשומה בפלט השאילתא, כמו כן ע"י שימוש ב-Curser ניתן גם לגשת באופן יחסי, אבסולוטי לכל שדה וכו'.

דוגמא: נניח נרצה להריץ שאילתא המוצאת את מוצרי הקפה שמחירם קטן שווה ל-8.99 ₪:

```
Statement stmt = conn.createStatement();
//the ResultSet will hold the query result which we can manipulate
ResultSet rs = stmt.executeQuery("SELECT GIFT_NAME, PRICE FROM " +
                   "GIFTS WHERE PRICE <= 5000");
```

6. **שליפת נתונים מתוך ResultSet**

כפי שנאמר האובייקט ResultSet מכיל את הרשומות של פלט שאילתת SQL שבוצע דרך JDBC. לאובייקט זה קיימות מתודות שימושיות למעבר על הרשומות וכמו כן לגישה לכל אחד מהשדות ברשומות. שימו לב: אובייקט זה מחזיק מצביע לרשומת המידע הנוכחית. כאשר הוא נוצר לראשונה הוא המצביע מאותחל לערך ראשוני, לפני הרשומה הראשונה.

דוגמא: נוציא ל-Standart Output את תוצאת השאילתא מדוגמה קודמת:

```
System.out.println("This is the query result");
while(rs.next()){
      System.out.println(rs.getString(1)+" "+rs.getFloat(2));
}
```

7. **מיפוי ביו טיפוסי נתונים של SQL לאלה של JDBC**

על מנת לשוף שדה מסוג yyy של SQL כשדה של JDBC מסוג xxx נשתמש במתודה ResultSet.getxxx(int index) עפ"י טבלת המעבר הבאה:

SQL Type	Java Method
BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()
LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

8. **שימוש ב-Prepared Statements**

מסיבות אבטחה **חייבים להשתמש ב-PreparedStatement ע"מ לשלוח הצהרות SQL** למסד הנתונים. הנה סרטון קצר שמסביר את פריצה האבטחה: https://youtu.be/ciNHn38EyRc

לא להשתמש במחרוזות אשר נבנות "on the go".

דוגמא:

```
PreparedStatement updateSales = con.preaparedStatement("UPDATE GIFTS "+
                                "SET SALES = ? WHERE GIFT_NAME = ?");
```

ועתה נניח שיש לעדכן למתנה Diamond את סך המכירות להיום ל-30(יח' מכירה)

```
updateSales.setInt(1, 30);
updateSales.setString(2, "Diamond");
updateSales.executeUpdate();
```

ואם שוב פעם יש להזין נתוני מכירות פשוט נזין ערכים חדשים ונריץ העדכון.

9. עדכון נתונים דרך ResultsSets

ניתן ליצור ResultSet של שאילתת SQL שלאחר מכן ניתן להשתמש באתו אובייקט על מנת לבצע פעולות עדכון על תוצאות השאילתא.

דוגמא:

```
Statement stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                    ResultSet.CONCUR_UPDATABLE);

ResultSet uprs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM GIFTS");

//noy you may update each one of the rows in the result set
uprs.last(); //the curser in the result set moves to last row
uprs.updateFloat("PRICE", (float) 10.99);
uprs.updateRow();

//case you want to cancel the row updates and change the price again
uprs.cancelRowUpdates();
uprs.updateFloat("PRICE", (float) 11.99);
uprs.updateRow();
```

10. הכנסה ומחיקת רשומות חדשות באופן מתוכנת

באופן דומה ניתן הכניס רשומות חדשות לטבלה או למחוק רשומות מטבלה.

```
ResultSet uprs = stmt.executeQuery("SELECT * FROM GIFTS");

// use the ResultSet Object to insert new row for "Barby" for 10.99
//                                                  units of some currency
uprs.moveToInsertRow();
uprs.updateString("GIFT_NAME", "Barby");
uprs.updateInt("SUP_ID", 150);
uprs.updateFloat("PRICE", 10.99);
uprs.updateInt("SALES", 0);
uprs.updateInt("TOTAL", 0);
uprs.insertRow();
```

11. תפיסת חריגות וטיפול בהתראות ממסד הנתונים

על פי רוב פעולות המתבצעות על אובייקט בחבילת java.sql יכולות לזרוק חריגות SQLException שיש להצהיר עבורן בלוק של try…catch על מנת לתפוס אותן.

למשל בעת רישום Driver בצורה הבאה יתכן ותיזרק חריגה:

```
try{
	Class.forName(<driver class path>);
}catch(java.lang.ClassNotFoundException e){
	System.err.println("no class for Driver:" + e.getMessage ());
}
```

התראה (SQL Warning) היא חריגה שאינה נזרקת, אלא יש לברר אם ארעה:

```
rs.executeQuery("SELECT_");
SQLWarning war = rs.getWarnings();
if (warn != null)
	System.err.println("warning: "+warn.getMessage());
```

12. תרגיל מעבדה 4

1. תעשה clone לפרוייקט מ-Clasroom GitHub כפי שנלמד במעבדה 4:
https://classroom.github.com/g/ZloyfvaO את הפרוייקט כדאי לייבא בתור פרוייקט Maven אם הIDE שלכם שואל.
2. תצרו מסד נתונים חינמי ב-https://remotemysql.com
3. כנסו ל-https://remotemysql.com/phpmyadmin/
   a. יש משמעות לפרטי ההתחברות, שימו לב שאתם מעדכנים את הקוד בתוך סקריפט SQL וקובץ JAVA בהתאם.
4. שחזרו database בשם flights מהקובץ backupFlights.sql מכיל את הטבלה *Flights(num, origin, destination, distance, prices)* עם רשימה של מחירי הטיסות. את הקובץ אפשר למצוא בתקית הפרוייקט שהורדתם. בשביל לשחזר את הטבלה:
   a. תלחצו על מקש SQL בחלק עליו של דף האינטרנט של PHP Admin והדביקו בחלון שמתחת את תוכן הקובץ backupFlights.sql.
   b. תעדכנו את השם של מבנה נתונים לפי מה שקבלתם (צריך לעדכן את שורה הראשונה והשניה).
   c. לאחר מכן הקליקו על GO שמתחת. שרת MySQL כרגע יריץ את הפקודה שנתתם. והטבלה תתמלא בנתונים.
5. כל המשימות הבאות צריכות להתבצע באמצעות אפליקציית Java (ולא ישירות באמצעות MySQL):
   a. הכניסו ל-database מחיר חדש לטיסה מספר 387.
   b. שלפו את הערך החדש והדפיסו את מחיר הטיסה המעודכן.
   c. עדכנו את מחירי הטיסות ב-database. העלו את מחיר כל טיסה במרחק הגדול מ-1000 ק"מ ב-50$. הראו את התוצאות. עדכנו את ה- databaseבעזרת ResultSet
   d. תחזרו על אותה פעולה בעזרת PreparedStatement.

הוראות הגשה:

1. ב-GitHub:
   קוד הפרוייקט מעודכן ב-master
2. במודל קובץ זיפ המכיל:
   a. את כל קבציה הפרוייקט
   b. מסמך עם 3 צילומי מסך של שלושה סעיפים אחרונים (3, 4, 5)
   c. במקום ברור במסמך שיהיה רשום בגדול שם של הקבוצה שתחתיו הגשתם את העבודה בגיטהב.

בהצלחה לכולם,

ועבודה נעימה