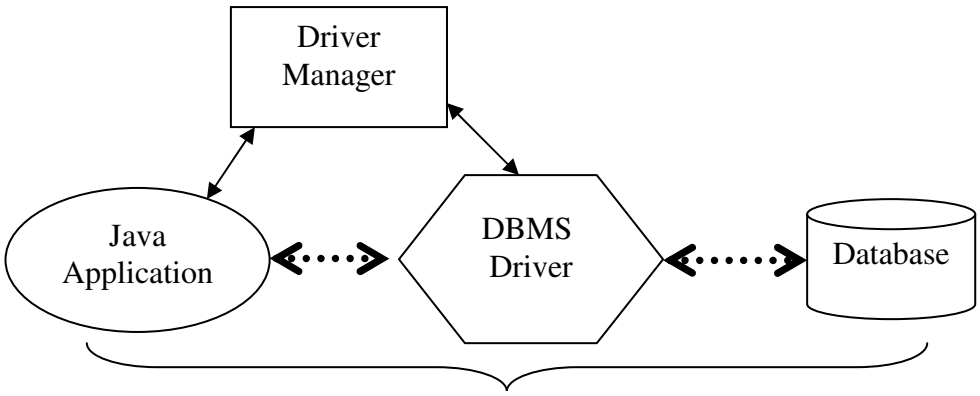
JDBC אקסס ו-mySQL

מבוא:
בתרגיל זה נלמד כיצד לבצע התחברות למסד הנתונים mySQL היושב ע"ג מחשב אחר דרך אפליקציה הכתובה ב-Java תוך שימוש ב-JDBC (Java DataBase Conectivity).

- **JDBC**:
באמצעות JDBC ניתן לכתוב קוד ב-Java המכיל בתוכו קוד המסוגל לפנות למסד הנתונים 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 צורך לייבא את חבילת java.sql .
באמצעות שימוש ב- JDBC ניתן לבצע מספר התחברויות בו זמני 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 חזרה ל- JDBC. בד"כ טוענים Driver-ים באופן דינאמי בזמן ריצה ע"י רישום אצל DriverManager שאחראי על ניהול כל ה-Driver-ים 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 בדפוס תכן של Object Pooling בכל הקשור לניהול אובייקטיי ההתחברות מאחר ויצירת התחברות כרוכה בהשקעת משאבי מחשוב רבים (למשל: פרוטוקול תקשורת,משאבי זיכרון...).

- **JDBC – ארכיטקטורה**:



- 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.

- "מסד חברת התעופה" זהו מסד אקסס שיושב אצלכם מקומית במחשב.
בתרגול נשתמש בסכמה הבאה:

Aircrafr(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

- "מסד ההוצאה לאור" זהו מסד נתונים של mySQL שיושב במחשב מרוחק.
בתרגול נשתמש גם בסכמה הבאה:

Authors (authID,authName)

- רישום ה Driver:

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

אנחנו נשתמש ב Driver שמגיע עם API של mySQL ולכן נרשום:

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

ניתן גם לרשום Driver ישירות דרך ה- DriverManager:

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

רישום דרייבר של אקסס מתבצע דרך המגשר ODBC דבר שמחייב בנוסף לרושמו במערכת ההפעלה.

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

- ביצוע התחברות למסד mySQL:

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

String username = "OrtStudent";
String password = "Braude";

try{

   Connection conn =
                  DriverManager.getConnection(url,username,password);

}catch(SQLException e){
  System.err.println("Connection to Database failed:"+e.getMessage());
}
```

חיבור למסד האקסס בתרגול זה דומה, אך שונה בכתובת ובסיסמה
הכתובת מפנה למעשה ל ODBC "jdbc:odbc:MSAccess"

```
this.con = DriverManager.getConnection(url, "Admin","");
```

- יצירת JBDC Statements:

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. על מנת ליצור אובייקטי Statement יש לפנות אל אובייקט ההתקשרות ולקבל ממנו התייחסות לאובייקט (reference).

דוגמא: 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 statment
stmt.executeUpdate(createTablegifts);
```

כל פעולות המבצעות יצירה/ביטול של טבלאות או מבטים הן פעולות DDL ויש לבצען תוך שימוש במתודה הנ"ל. כמו כן פעולות DML שמעדכנות נתונים (הוספה,עדכון,שינוי,מחיקה) גם הן מבוצעות תוך שימוש באותה מתודה.

דוגמא: עתה נניח שברצוננו להכניס רשומה של קפה לטבלת GIFTS:

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

- שליפת נתונים מתוך טבלה:

בשימוש באובייקט Statement ניתן גם להריץ שאילתות למסד ולקבל תוצאת השאילתא.

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

<u>דוגמא</u>: נניח נרצה להריץ שאילתא המוצאת את מוצרי הקפה שמחירם קטן או שווה ל- 8.99 ₪:

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

- <u>שליפת נתונים מתוך ResultSet</u>

כפי שנאמר האובייקט `ResultSet` מכיל את הרשומות של פלט שאילתת SQL שבוצעה דרך JDBC. לאובייקט זה קיימות מתודות שימושיות למעבר על הרשומות וכמו כן לגישה לכל אחד מהשדות ברשומות. שימו לב: כאשר אובייקט זה נוצר לראשונה הוא ממוקם (לוגית) מעל הרשומה הראשונה.

<u>דוגמא</u>: נוציא ל - Standard Output את תוצאת השאילתא הקודמת:

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

- <u>מיפוי בין טיפוסי נתונים של SQL לאלה של JDBC:</u>

על מנת לשלוף 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()

טבלה: מעבר מטיפוסי SQL ל- JDBC

- שימוש ב- Prepared Statements:

לפעמים יותר נוח וגם יותר יעיל להשתמש באובייקט מסוג PreparedStatement ע"מ לשלוח הצהרות SQL למסד הנתונים. אובייקט זה יורש ישירות מאובייקט Statement שכבר הכרנו בתרגול זה. למשל בד"כ רצוי להשתמש באובייקט זה כאשר תדירות השימוש בהצהרה כלשהי באפליקציה היא גובהה (למשל הכנסת נתוני טרנזאקציות כספיות של בנק , כאשר ההצהרה היא תמיד זהה אך קלטים שונים) היות ועפ"י שיטה קודמת יש ליצור אובייקט Statement עבורו יש לשכתב כל פעם מחדש מחרוזת עבור הרצת ההצהרה.

דוגמא: נניח שהאפליקציה שלנו מקבלת נתונים מכירות של קפה ויש לעדכן אותן במסד (נניח כי נתוני המכירות מוזנים דרך ממשק גראפי כלשהו או נקראים מקובץ או משורת הפעלה וכד'...)

```
PreparedStatment updateSales = con.prepareStatement("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 העידכון...

- עדכון נתונים דרך ResultSets:

ניתן ליצור ResultSet כפלט של שאילתת SQL שלאחר מכן ניתן להשתמש באותו אובייקט על מנת לבצע פעולות עדכון על תוצאת השאילתא.

דוגמא:

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

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

//now 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",10.99);
uprs.updateRow();

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

- הכנסה ומחיקת רשומות חדשות באופן מתוכנת:

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

דוגמא:

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

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

//use the result set Object to insert a new row for Kona Coffee
```

```
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.inserRow();
```

- <u>תפיסת חריגות וטיפול בהתראות ממסד הנתונים</u>:

בד"כ כל פעולות המתבצעות על אובייקטים בחבילת java.sql יכולות לזרוק חריגות SQLException
שיש להצהיר עבורן על בלוק של try….catch על מנת לתפוס אותן.
בעת רישום של Driver בצורה הבאה יתכן ותזרק חריגה נוספת:

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

התראות (SQLWarning) היא חריגה שאינה נזרקת , אלא יש לברר אם ארעתה:

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