# JDBC

Java DataBase Connectivity.

ה JDBC הוא ה API של ג'אווה לצורך גישות ל DB: שליפות של נתונים, עדכוני נתונים, שינויים של טבלאות וכו'.

הclasses יושבים בשני ה packages:

java.sql.

javax.sql.

**מושגים:**

**DB** – נתונים הנשמרים בצורה טבלאית.

**DBMS** – תוכניות המנהלות DB כמו: Oracle, DB2, SqlServer, Access.

**SQL** – שפה משותפת לכל סוגי ה DB, אשר נותנת אפשרות של: יצירה, עדכון ומחיקה של נתונים, ניהול טבלאות, שליפת נתונים.

**DDL**- Data Definition Language – פקודות לניהול טבלאות כמו: יצירת טבלה חדשה, עדכון טבלה קיימת, מחיקת טבלה, שינוי עמודות.

**DML** – Data Managing Language – פקודות לניהול נתונים: יצירה, עדכון, מחיקה.

Stored Procedure – שאילתה שמורה היכולה לקבל פרמטרים.

**Primary Key** (PK)– מפתח ראשי. המאפיין של מפתח ראשי זה שהוא לא יכול להיות null והוא unique.

**Foreign Key** (FK)– קשר בין רשומה בטבלה אחת לרשומה בטבלה אחרת, על ידי עמודות מקשרות. באחת מהטבלאות העמודה המקשרת צריכה להיות PK.

**Join** – שליפה מכמה טבלאות בשאילתא אחת.

**טרנזקציה** – סט של שאילתות הקשורות בינהן בקשר לוגי, והם תלויות אחת בשניה: או שכל השאילתות מבוצעות בסוף (commit), או שאף אחת לא מבוצעת – אם אחת נכשלה – כולן נכשלות (rollback) – אין עדכון חלקי, וכל עוד לא בוצעו כל העדכונים – לא רואים את העדכון של חלק מהנתונים.

בנוסף, אפשר לנעול את השורה / טבלה אם היא באמצע טרנזקציה.

**ResultSet** – התוצאה המתקבלת ממשפט Select. יכול להכיל 0 רשומות או יותר, וניתן לעבור על הרשומות.

**MetaData** – מידע על ה DB מבנה טבלאות, סוגי ערכים של עמודות ועוד.

**ODBC** – גשר בין כל DB שהוא לבין אפליקציה.

חוקים עבור Relational DB:

1. אסור שיהיו שורות כפולות.
2. יש משמעות לערך null (הוא לא נשמר בתור 0 – יש לו ערך מיוחד).
3. יש PrimaryKey – עמודה אחת או יותר המייצגות את המפתח הראשי.
4. ישנה תמיכה להרצת שאילתות SQL.

**JDBC:**

ה JDBC נותן FrameWork לעבודה מול DB:

הוא נותן את ה interfaces הבאים:

Driver

Connection

Statement

ResultSet

כאשר כל DBMS (oracle, sqlserver וכו') מממש את ההתנהגות של ה FrameWork מול ה DB שלו.

הרבה חברות המספקות שירותי DB מימשו את ה API של ה JDBC וכך אפשר לגשת ישירות אל ה DB – זה מאוד יעיל ונותן גמישות יתר.

כדי להשתמש במימוש אלו יש להתקין jar מיוחד, ולהשתמש במימושים ספציפיים של ה interfaces הנ"ל. כאשר מבחינת כתיבת קוד – כל ההבדל הוא רק ביצירה שונה של האוביקטים.

אם אין מימוש מיוחד אפשר תמיד להשתמש ב ODBC – שזה גשר המקשר בין כל סוגי ה DB לאפליקציה.

**יצירת Connection –**

פעולת קישור אל ה DB. זו הפעולה הראשונה בכל פניה אל ה DB.

ישנם 2 דרכים לעשות זאת:

1 DriverManager – חיבור ל DB על ידי String URL.

2. DataSource – חיבור ל DB על ידי שם לוגי, יותר גמיש, ניתן לשנות את שם הDB בלי לשנות את הקוד.

שימוש ב DriverManager:

ראשית, יש להטעין את ה Driver על ידי הפקודה:

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

כאשר שם הקלאס משתנה מ DB ל DB.

פתיחת Connection:

DriverManager.*getConnection*(url);

ב url – נרשום את שם ה DB כפי שמגדיר ה DB Vendor (מנהל ה DB).

לדוגמא: ב ODBC זה ייכתב בצורה הבאה:

con = DriverManager.*getConnection*("jdbc:odbc:myDBname");

אפשר להוסיף פרמטרים נוספים על ידי שימוש ב ;.

ישנו overloading ל Connection – אפשר לשלוח גם שם וסיממא:

con = DriverManager.*getConnection*("jdbc:odbc:myDBname", "enga13","1q2w3e");

**קבלת Statement מה Connection:**

Statement stmt = con.createStatement();

ע**ל ידי ה Statement נריץ את כל משפטי ה SQL שנרצה:**

הרצת פקודות DDL:

stmt.execute("create table ADRESS (id varchar(6) primary key, name varchar(50))");

הרצת פקודות DML (insert,update,delete):

stmt.executeUpdate("insert into ADRESS values('123', 'abc')");

הרצת שאילתות Select:

ResultSet rs = stmt.executeQuery("select \* from PERSON");

הפונקציה executeQuery – מחזירה אובייקט ResultSet המכיל את כל הרשומות שחזרו.

אם רוצים להריץ שאילתת executeQuery על ה statement אפשר ביצירת ה Statement לשלוח גם פרמטרים שמגדיר איך יעבוד ה resultSet:

stmt = con.createStatement(ResultSet.*TYPE\_SCROLL\_INSENSITIVE*,ResultSet.*CONCUR\_UPDATABLE*,ResultSet.*FETCH\_REVERSE*);

**שימוש ב ResultSet:**

ה resultSet מכיל קבוצה של רשומות שמתקבלת כתוצאה מהשאילתא.

על ה resultSet ישנו מצביע – המצביע בכל פעם על רשומה אחת, ניתן להזיז את הרשומות קדימה, אחורה (אם הוגדר type מתאים), להתחלה לסוף.

בהתחלה ה RS מצביע לפני הרשומה הראשונה.

פונקציות:

next() – מזיז את המצביע רשומה אחת קדימה.

previous()– מזיז את המצביע רשומה אחת אחורה.

first()– מזיז את המצביע לרשומה הראשונה.

last()– מזיז את המצביע לרשומה האחרונה.

beforeFirst()– מזיז את המצביע ללפני הרשומה הראשונה.

afterLast()– מזיז את המצביע לאחרי הרשומה האחרונה.

relative(int rows) – מזיז את המצביע מהמקום שהוא נמצא.

absolute(int row) – מזיז את המצביע מההתחלה.

שליפת נתונים מה ResultSet:

בכל פעם ה RS מצביע על שורה אחת, כדי לשלוף את הנתונים של אותה שורה נשתמש במתודות getXXX – כאשר XXX מייצג את סוג השדה ששולפים (getInt, getString).

הפונקציות מקבלות או int – ה index של העמודה, או String – שם העמודה.

\* ה index מתחיל מ 1.

\* שמות העמודות הם case sensitive.

\* אם אותו שם עמודה מופיע פעמיים – יחזור הראשון שיימצא.

דוגמא:

stmt = con.createStatement();

rs = stmt.executeQuery("select \* from PERSON");

while(rs.next()) {

String id = rs.getString("ID");

String name = rs.getString("NAME");

System.*out*.println("id: " + id + " name: " + name);

}

עדכון על ידי RS:

ניתן לבצע גם עדכונים על ה ResultSet על ידי הפקודות של updateXXX(columnName, value) – שמעדכן את העמודה המצויינת ב columnName עם ה value. ברשומה בא המצביע נמצא כעת.

העדכון עוד לא נשמר ב DB – כדי לבצע עדכון ב DB יש להוסיף:

updateRow בסוף.

דוגמא:

stmt = con.createStatement(ResultSet.*TYPE\_FORWARD\_ONLY*,ResultSet.*CONCUR\_UPDATABLE*);

ResultSet rs = stmt.executeQuery("select \* from PERSON");

rs.next();

rs.updateString("NAME", "woo");

rs.updateRow();

יש לזכור לבצע סגירה בסוף כל פעולה:

} **catch** (SQLException e) {

System.*out*.println("an unexcepted exception been accoured");

e.printStackTrace();

**throw** e;

} **finally** {

**if** (rs != **null**) {

rs.close();

}

**if** (stmt != **null**) {

stmt.close();

}

**if** (con != **null**) {

con.close();

}

}

**PreparedStatements:**

preparedStatement היא אפשרות נוספת להרצת שאילתות. היא יותר יעילה בזמן, במיוחד כאשר מריצים את אותה שאילתה הרבה פעמים – כיון שהשאילתא מקומפלת פעם אחת, ואפשר כל פעם רק לשנות לה את הפרטמרים.

**דוגמא:**

Connection con = DriverManager.*getConnection*("jdbc:odbc:kupatHolim");

//create the PreparedStatement

PreparedStatement updatePerson = con.prepareStatement("UPDATE PERSON SET NAME = ? WHERE ID LIKE ?");

//send paramters to the PreparedStatement

//the first paramter is the index of the paramter in the statement (? number...)

updatePerson.setString(1, "david");

updatePerson.setString(2, "123");

//run the sql

updatePerson.executeUpdate();

**StoreProcedure:**

שאילתות המאוחסנות ב DB וניתנות להרצה על ידי JDBC עם פרמטרים שונים. היתרון של StoreProcedure הוא שהם מאוד מהירות.

לא נלמד על StoreProcedure במסגרת.

**Transactions:**

טרנזקציות הם חלק חשוב מעדכוני DB. פעמים רבות, על מנת לשמור על נכונות הנתונים, נרצה לעדכן קבוצה של שאילתות יחד כטרנזקציה, ואז או שהכל מצליח, או שהכל נכשל.

con.setAutoCommit(false); - פונקציית autoCommit היא פונקציה הנמצאת על Connection כאשר היא במצב של true – commit יתבצע על כל פקודה ופקודה בנפרד (זה ה default עבור ה connection), על מנת לתמוך בטרנזקציות נשנה א ת ה autoCommit ל false.

בסיום של טרנזקציה נבצע או:

con.commit() – לאישור כל הטרנזקציה, כל הפעולות יבוצעו.

con.rollback() – לביטול כל הטרנזקציה, כל הפעולות שבוצעו בטרנזקציה זו יבוטלו.