מסדי נתונים

עד עכשיו למדנו לעבוד עם עצמים בשפת ג'אבה. היום נלמד לעבוד עם עצמים בשפה שונה - שפת SQL - ונראה את הדומה והשונה. שפת SQL נועדה לעבודה עם **מסדי-נתונים-רלציוניים** (relational database) ולכן קודם נדבר בקצרה על מסדי-נתונים בכלל. ישנן כמה דרכים להסביר מה זה מסד נתונים (database):

- מסד-נתונים הוא אוסף של עצמים שנשמר לאורך זמן (persistent) וחי גם מעבר לאורך-החיים של התוכנה. זאת בניגוד לעצמים בתוכנה שנעלמים כשהתוכנה מסתיימת.
- מסד-נתונים אנחנו כבר מכירים מנגנון פשוט ל-persistence והוא - קבצים. למדנו איך לכתוב עצמים לקובץ בפורמטים שונים (json, xml, ObjectStream) ואיך לקרוא אותם בחזרה מהקובץ.הוא קובץ (או אוסף קבצים) שיש להם מבנה ומשמעות מוגדרים. זאת בניגוד לקובץ רגיל שהוא רק אוסף של בתים/אותיות בלי מבנה.

קובץ הוא סוג פשוט של מסד נתונים; סוגים משוכללים יותר של מסדי-נתונים נותנים לנו יתרונות נוספים, כגון:

- ריבוי חוטים - אפשר לקרוא/לכתוב למאגר-הנתונים מכמה חוטים שונים והוא יישאר תקין (בניגוד לקובץ).
- יעילות - במסדי-נתונים ישנם מפתחות המאפשרים לגשת לחלקים מהמאגר במהירות.
- סדר - מסד-נתונים אחד יכול לשמור מידע שהיינו צריכים לפזר בקבצים שונים.

ישנם שני סוגי מסדי-נתונים:

- מסד נתונים פשוט הוא בסה"כ אוסף של מפתחות ועצמים. זה כמו Map בשפת ג'אבה, רק שהוא חי מעבר לאורך-החיים של התוכנה (persistent). דוגמאות: MongoDB, Redis.
- מסד נתונים רלציוני הוא אוסף של טבלאות. לכל טבלה יש אוסף קבוע של עמודות מסוגים שונים, ומספר משתנה של שורות. דוגמאות: SQLite, MySQL, MS-SQL, Maria DB, Oracle DB, ...

מסד-נתונים רלציוני מאפשר ליצור טבלאות, לעדכן טבלאות ולשאול שאילתות בשפה מקובלת הנקראת SQL. השפה הזאת כל-כך מקובלת, עד שמסדי-נתונים שאינם רלציוניים נקראים "NoSQL".

ישנה הקבלה מסויימת בין עצמים במסד-נתונים רלציוני לבין עצמים בג'אבה:

מסד-נתונים רלציוני	ג'אבה
הגדרת טבלה	הגדרת מחלקה
שורה בטבלה	עצם
טבלה	רשימה של עצמים
עמודה בטבלה	שדה

אבל יש גם הבדלים:

- בג'אבה לכל עצם יש בנוסף לשדות גם **מתודות**; במסד-נתונים בדרך-כלל אין דבר כזה, הפעולות נפרדות מהמתודות.
- בג'אבה יש הבחנה בין שדות ציבוריים ושדות **פרטיים**; במסד-נתונים בדרך-כלל כל השדות ציבוריים.
- בג'אבה יש ירושה; במסד-נתונים רלציוני בדרך-כלל אין.

היתרון של מסדי-נתונים רלציוניים, בנוסף לשמירת מידע (persistence), הוא האפשרות לבצע בהם שאילתות מורכבות בקלות, כפי שנראה בהמשך.

מערכות מסדי-נתונים רלציוניים

אנחנו נלמד שתי מערכות לניהול מסדי-נתונים רלציוניים: סקלייט (SQLite) ומייסקל (MySQL).

- סקלייט היא פשוטה, ונועדה לשמירת נתונים של משתמש אחד על מחשב מקומי;
- מייסקל היא משוכללת יותר, ונועדה לשמירת נתונים של משתמשים רבים על שרת משותף.

במערכת סקלייט, כל מסד-נתונים נמצא בקובץ אחד (כמו בתוכנת Microsoft Access, זה גם די דומה לאקסל).
כדי לעבוד עם סקלייט, כל מה שצריך הוא להתקין תוכנה פשוטה וחינמית שנקראת sqlite browser. כשאנחנו יוצרים מאגר חדש, נוצר אוטומטית קובץ עבורו. אפשר להגדיר טבלאות ולהכניס לתוכן נתונים דרך הממשק הגרפי, אבל אנחנו נלמד גם איך לעשות זאת בעזרת פעולות סקל.

במערכת מייסקל, מסד-נתונים נמצא על שרת כלשהו. אפשר להפעיל שרת על המחשב שלנו, אבל אפשר גם להתחבר לשרת קיים; נראה בהמשך איך עושים את זה.

שתי המערכות מבינות את שפת סקל (SQL). בשלושת הפרקים הבאים נלמד את שלושת הסוגים העיקריים של פקודות שיש בשפה: **הגדרת נתונים**, **שינוי נתונים**, **ושאילתות נתונים**.

א. שפת הגדרת נתונים

מסד נתונים רלציוני בנוי מ**טבלאות**. בכל טבלה יש **עמודות**. כדי לעבוד עם טבלאות צריך קודם-כל להגדיר אותן. תת-השפה של סקל האחראית להגדרת טבלאות נקראת "שפת הגדרת נתונים" - Data Definition Language. למשל, הפקודה:

```
CREATE TABLE employees
(id         INT PRIMARY KEY     NOT NULL,
 name       TEXT     NOT NULL,
 age        INT      NOT NULL,
 address    CHAR(50),
 salary     REAL);
```

מגדירה טבלה בשם employees עם חמש עמודות. ליד כל עמודה כתוב הטיפוס שלה וכן פרטים נוספים שנראה בהמשך.

כדי למחוק טבלה משתמשים בפעולת SQL:

```
DROP TABLE employees;
```

ב. שפת שינוי נתונים

אחרי שיצרנו טבלה, אנחנו יכולים להכניס ולהוציא ולעדכן בה שורות. תת-השפה של SQL האחראית לשינוי נתונים בטבלאות נקראת "שפת שינוי נתונים" - Data Manipulation Language.
הפעולות הרלבנטיות הן: INSERT, DELETE, UPDATE. למשל:

```
INSERT INTO employees (id,name,age,address,salary)
       VALUES (1, 'Paul', 32, 'California', 20000.00);

DELETE FROM employees WHERE id IN (1,2,3,4);

UPDATE employees SET salary=25000 WHERE id=1 OR id=3;
```

ג. שפת שאילתת נתונים

כדי לקרוא נתונים יש לבצע **שאילתות**. לשם כך משתמשים בתת-שפה של SQL המיועדת לשאילתות ונקראת Data Query Language. הפעולה העיקרית בשפה זו היא **SELECT**, הנה כמה דוגמאות:

```
SELECT * FROM employees;
SELECT id,name,age FROM employees WHERE id=1;
SELECT name,age FROM employees WHERE salary>=20000;
SELECT MAX(age) FROM employees WHERE salary>=20000;
SELECT age, COUNT(*), AVG(salary) FROM employees GROUP BY age;
```

שפת השאילתות של SQL היא עשירה ומשוכללת מאד ואפשר לבצע בה הרבה שאילתות מורכבות בשפה כמעט טבעית; ראו דוגמאות בקובץ DataQueryLanguage.sql.

צירוף טבלאות

אחת האפשרויות השימושיות ביותר בשפת סקל היא האפשרות לצרף טבלאות שונות. מכאן נובע התואר "רלציוני" שהשתמשנו בו כמה פעמים. relation = יחס; מסד-נתונים רלציוני מאפשר להגדיר יחסים בין נתונים בטבלאות שונות.

לדוגמה, נניח שאנחנו רוצים להגדיר צוותים - לכל צוות יש ראש-צוות וכן חבר-צוות אחד או יותר. בשפת ג'אבה יכולנו פשוט להוסיף לכל עצם מסוג Employee, רשימה של עובדים שנמצאים בצוות שלו (List<Employee>). אבל במסד-נתונים רלציוני אין שדה מסוג "אוסף" ובפרט לא מסוג "אוסף של עצמים". מה עושים? -מגדירים טבלה נוספת:

```
CREATE TABLE IF NOT EXISTS teams
(manager   INT     NOT NULL,
 member    INT     NOT NULL
```

```
);
```

הטבלה מגדירה קשר בין מנהל-צוות לבין חבר-צוות. למשל, אפשר למלא אותה כך:

```
INSERT OR IGNORE INTO teams(manager,member)
       VALUES (1,1), (1,2), (1,3), (3,3), (3,4);
```

פקודה זו מגדירה שני צוותים - צוות בראשות עובד מס' 1 ובו שלושה חברים, וצוות בראשות עובד מס' 3 ובו שני חברים.

עכשיו, נניח שאנחנו רוצים לדעת, עבור כל ראש-צוות, את שמותיהם ומשכורותיהם של חברי-הצוות שלו. איך עושים זאת? בעזרת פעולת **JOIN** - צירוף:

```
SELECT * FROM employees
INNER JOIN teams ON(employees.id=teams.member);
```

הפעולה JOIN יוצרת מכפלה קרטזית של כל שורה בטבלה employees עם כל שורה בטבלה teams, ואז מצמצמת אותה לפי התנאי שבתוך ה-ON. במקרה זה, נקבל עבור כל שורה בטבלה employees, את השורה של המנהל בטבלה teams.

תרגילי כיתה:

1. כיתבו שאילתה המחשבת, עבור כל ראש-צוות, את מספר חברי-הצוות שלו ואת הגיל הממוצע שלהם.
2. כיתבו שאילתה המציגה טבלה עם שתי עמודות: שם ראש-הצוות ושם חבר-הצוות (כמו הטבלה teams רק עם שמות במקום מספרים).

ירושה

במסד-נתונים רלציוני אין ירושה. זה מעורר את השאלה איך לשמור היררכיה של מחלקות בג'אבה במסד-נתונים כזה? ישנן כמה דרכים:

- שימוש בטבלה אחת גדולה - הכוללת את כל השדות בהיררכיה. שדות שאין בהם שימוש יהיו null.
- שימוש בטבלה נפרדת עבור כל מחלקה, הכוללת את כל השדות של אותה מחלקה (כולל השדות שהתקבלו בירושה).
- שימוש בטבלה נפרדת עבור כל מחלקה, הכוללת רק את השדות החדשים של אותה מחלקה (ללא השדות שהתקבלו בירושה). בשיטה זו צריך מזהה מספרי מיוחד שיקשר בין הטבלאות.

הרחבה בנושא זה היא מעבר להיקף של הקורס הנוכחי.

הגדרת פונקציות

CREATE FUNCTION - המשך יבוא(?)

עבודה עם מייסקל

עד עכשיו עבדנו עם סקלייט על המחשב שלנו. מייסקל מאפשר לנו לעבוד עם מאגרים הנמצאים על שרת מרוחק.

כלי מקובל לעבודה עם מייסקל הוא mysql workbench.

לצורך השיעור פתחתי חשבון חינמי באתר https://www.freemysqlhosting.net/, וקיבלתי שם מסד נתונים בחינם. כתובת השרת היא: sql11.freemysqlhosting.net ושם המאגר הוא: sql11213398.

במערכת מייסקל, בניגוד לסקלייט, צריך שם-משתמש וסיסמה. אני קיבלתי מהם שם-משתמש וסיסמה עם הרשאות לעבוד עם מאגר-הנתונים שפתחו לי. הכנסתי את הנתונים ל-mysql workbench והתחברתי.

כל הפקודות בשפת סקל שראינו עד עכשיו, הן זהות לחלוטין בסקלייט ובמייסקל. כך שאפשר להריץ אותן דרך mysql workbench ולקבל בדיוק אותן תוצאות, רק שהפעם, מסד-הנתונים שלנו נגיש דרך הרשת מכל מקום בעולם.

אחד היתרונות של מייסקל הוא האפשרות להגדיר משתמשים שונים עם הרשאות שונות. למשל, בועז הגדיר מאגר-נתונים שבו יש כמה משתמשים שונים: לחלק מהם יש הרשאות מלאות, לחלק מהם יש רק הרשאות לכתיבה וקריאה אבל לא למחיקת טבלאות, ולחלק מהם יש הרשאות לקריאה בלבד. פרטים מלאים על הגדרת משתמשים ומתן הרשאות תלמדו, אולי, בקורס מסדי נתונים.

מסדי נתונים בג'אבה

בפרק זה נראה איך לעבוד עם מסדי-נתונים מתוך ג'אבה, על-מנת לשלב אותם בפרוייקטי ג'אבה גדולים יותר שאנחנו בונים.

א. רכיב קישור

השלב הראשון בעבודה עם מערכת מסדי-נתונים היא הורדת רכיב המקשר בין המערכת שבחרנו לבין ג'אבה. רכיב-קישור בין ג'אבה לבין מערכת מסדי-נתונים נקרא JDBC. למשל, כדי לעבוד עם מערכת סקלייט צריך לחפש "sqlite jdbc". אפשר להוריד jar ולחבר אותו לפרוייקט כמו שעשינו עד לפני שבועיים, או שאפשר להוסיף תלות בקובץ גריידל כפי שלמדנו לפני שבוע:

```
repositories {     mavenCentral()    }
dependencies {     implementation 'org.xerial:sqlite-jdbc:+'    }
```

שימו לב - התלות היא רק ב-implementation ולא ב-compile, כי אין צורך ברכיב-הקישור לצורך קומפילציה! הקומפילציה נעשית עם הספריה הכללית של ג'אבה ל-sql ובלי תלות במערכת מסויימת.

ב. התחברות

השלב השני הוא ליצור חיבור (Connection) מתוך התוכנה שלנו בג'אבה למסד נתונים מסויים. למשל, כדי להתחבר למאגר "company.db" של מערכת סקלייט נשתמש בפקודה הבאה:

```
Connection connection = DriverManager.getConnection(
    "jdbc:sqlite:company.db")
```

במערכת סקלייט, כל מסד נתונים הוא בסה"כ קובץ על המחשב. הפעולה הנ"ל מחפשת את הקובץ company.db בתיקייה של הפרוייקט, ואם הוא לא נמצא שם, היא יוצרת אותו.

בחיבור למאגר במערכת מייסקל, המחרוזת שמעבירים ל-getConnection היא מעט מורכבת יותר. כל מה שצריך לעשות הוא לשנות את פקודת ההתחברות ל:

```
Connection conn = DriverManager.getConnection(
     "jdbc:mysql://sql11.freemysqlhosting.net:3306/sql11213398",
     "<username>",        "<password>");
```

המחרוזת הראשונה מכילה את כתובת השרת, השלוחה (port) ושם מאגר-הנתונים. המחרוזת השניה היא שם-המשתמש שלי, והשלישית היא הסיסמה.

פקודת-ההתחברות היא ההבדל היחיד בין מערכות שונות; מרגע שיש לנו Connection, כל שאר הפעולות מתבצעות בשפת SQL בלי להתחשב בסוג המערכת שהתחברנו אליה. הדבר מקל מאד כשרוצים להחליף מערכת לניהול מסדי-נתונים.

שימו לב 1: הקומפיילר של ג'אבה דורש מכם לייבא את הספרייה הכללית לטיפול ב-sql:

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

אבל הוא **לא דורש** מכם לייבא שום ספריה ספציפית הקשורה לסקלייט. המשמעות היא, שאם שכחתם להתקין את סקלייט, לא תקבלו שגיאת קומפילציה אלא שגיאת ריצה כשתנסו להתחבר למאגר: "

```
java.sql.SQLException: No suitable driver found for
"jdbc:sqlite:test.db".
```

שימו לב 2: כדאי מאד לשים את פקודת החיבור למאגר בתוך בלוק של try, כדי שהחיבור ייסגר אוטומטית כשיוצאים מהבלוק. לפרטים ראו בדוגמאות הקוד.

ג. פקודות הגדרת נתונים ועדכון נתונים

כדי להעביר פקודת SQL מתוך ג'אבה למסד נתונים, צריך ליצור עצם מסוג "פקודה" (Statement):

```
Statement stmt = connection.createStatement()
```

גם אותו כדאי מאד ליצור בתוך בלוק try. במבנה זה משתמשים כדי לשלוח פקודות למאגר-הנתונים, למשל:

```
stmt.executeUpdate(
     "CREATE TABLE employees\n" +
     "(id        INT PRIMARY KEY     NOT NULL,\n" +
     " name      TEXT    NOT NULL,\n" +
     " age       INT     NOT NULL,\n" +
     " address   CHAR(50),\n" +
     " salary    REAL);\n" +
     ""
);
```

הערה: מומלץ מאד ליצור פקודות SQL בקובץ טקסט נפרד, ואז להעתיק אותן לתוך גרשיים בתוכנת ג'אבה. אם עובדים באקליפס, ההפרדה לשורות עם סימני + תיווצר אוטומטית.

כשמריצים stmt.executeUpdate עם פעולת שינוי נתונים (כגון INSERT, UPDATE, DELETE), הערך המוחזר הוא מספר השורות שהשתנו (נכנסו / נמחקו / התעדכנו). לפרטים ראו בדוגמה DataManipulationDemo.

ד. שאילתות

כדי לבצע שאילתה מתוך ג'אבה, משתמשים ב-executeQuery:

```
ResultSet rs = stmt.executeQuery("SELECT MAX(age) FROM employees
WHERE salary>=20000");
```

שוב, מומלץ לשים את הפקודה בתוך בלוק try. תוצאת הפעולה היא עצם מסוג ResultSet. אפשר לעבור עליו בלולאת while, למשל כך:

```
while (rs.next()) {
     System.out.println(
          "id=" + rs.getInt("id")+" "+
          "name=" + rs.getString("name")+" "+
          "age=" + rs.getInt("age")+" "+
          "address=" + rs.getString("id")+" "+
          "salary=" + rs.getFloat("salary"));
}
```

לדוגמאות נוספות ראו בקובץ DataQueryDemo.

ה. הכנת פקודות וחיטוי פקודות

עד עכשיו השתמשנו בפקודות קבועות מראש, אבל בדרך-כלל הפקודות שלנו יהיו תלויות בפרמטרים שמגיעים מהמשתמש. למשל, במערכת-מידע של עובדים בחברה, ראש מדור שכר רוצה לברר איזה משכורת מקבל עובד מסויים. לשם כך הוא צריך להקליד את שם העובד. איך משלבים פרמטרים כאלה בשאילתות?

אפשרות אחת היא פשוט להשתמש בשירשור מחרוזות, למשל: query="SELECT * FROM employees WHERE name='"+input+"'". הבעיה היא שזה מאד לא בטיחותי - ראש המדור עלול להכניס, בטעות או בכוונה, שם שגוי שיגרום למחיקת כל הטבלאות במאגר! (לדוגמה ראו https://xkcd.com/327).

כדי שהשאילתות לא יגרמו נזק, צריך "לחטא" אותן (sanitize). בפרט, יש לוודא שהן לא מכילות שום קלט מהמשתמש באופן ישיר, אלא רק דרך פקודות שהוכנו מראש. לשם כך משתמשים במחלקה PreparedStatement. למשל, כדי להכין מחלקה שבוחרת עובד לפי שם, אפשר להשתמש ב:

```
PreparedStatement stmt = connection.prepareStatement("SELECT *
FROM employees WHERE name=?")
```

במקום כל פרמטר שאמור להגיע מהמשתמש, נשים סימן שאלה (?). כדי להזין את הפרמטר מהמשתמש, נשתמש בפעולות set, למשל:

```
stmt.setString(1, "Paul");
```

המספר "1" הוא האינדקס של סימן-השאלה; **האינדקסים מתחילים מ-1**. הפעולה setString תכניס את המחרוזת במקום סימן-השאלה, עם גרשיים ועם "חיטוי" מתאים. אחר-כך נבצע ResultSet rs = stmt.executeQuery() כדי לבצע את השאילתה בפועל.

גם אם המשתמש ינסה לגרום נזק ע"י העברת פרמטרים שגויים, הוא לא יצליח. למשל, הפעולה:

```
stmt.setString(1, "Robert'; DROP TABLE employees");
```

פשוט תחזיר 0 שורות, ולא תגרום שום נזק.

(**הערה**: במייסקל אפשר להדפיס את stmt.toString ולראות איך המערכת משנה את הפקודה כך שלא תזיק).

מקורות

- JDBC Create database example, Tutorials Point: https://www.tutorialspoint.com/jdbc/jdbc-create-database.htm
- SQLite Java, Tutorials Point: https://www.tutorialspoint.com/sqlite/sqlite_java.htm
- Java Database Connection: JDBC and MySQL, Udemy course: https://www.udemy.com/how-to-connect-java-jdbc-to-mysql/learn/v4/overview
- How to do Inheritance Modeling in Relational Databases, StackOverflow, https://stackoverflow.com/q/1567935/827927

סיכם: אראל סגל-הלוי.