

מסדי נתונים :**שיעור 1 :****הקדמה:**

מסדי נתונים – מקום שמאחסן מידע, נתונים, אך אצלנו בקורס הנתונים מאוחסנים בצורה אלקטרונית, בנוסף זה גם דרך לנהל את הנתונים (DBMS).

כל חברה מחזיקה מאגר מידע כדי לשמור נתונים וכדי לנהל אותם.

סוגי מסדי נתונים –

1. רלאציוני – מבוסס על קשרי יחס (אלגברה רלאציונית).

בנוי מטבלאות עם מאפיינים (עמודות) וכניסות (רשומה).

עמודות בטבלה אחת יכולה להיות קשורה לעמודה בטבלה אחרת באותו database - יש קשרים בין הטבלאות.

הפעולות מתבצעות בעזרת transaction - תנועה – פעולה לוגית שאני עושה על הנתונים כדי לשנות אותם (לדוגמא מעבר כסף בין חשבון לחשבון), תנועות אלה חייבות להתקיים בבת אחת כלומר או שכולם מתקיימות או שהכל מתבטל.

ACID – סט של תכונות שהdatabase חייב לעמוד בהם :

Atomicity – מתבצע בשלמות או לא מתבצע בכלל.

Consistency – עקביות – אסור לפעולה להשאיר את database במצב לא חוקי, למשל להזין ציון לתלמיד שלא קיים.

Isolation – בידוד, תנועות שונות יכולות להתרחש בו זמנית רק בתנאי שזה יהיה שקול לפעולה סדרתית.

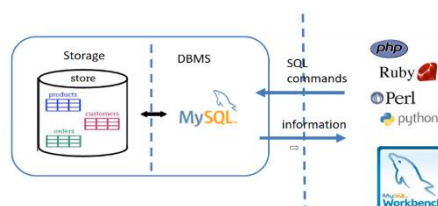
Durability – עמידות – כל בקשה שנשלח לdatabase חייבת להתבצע, כלומר גם במקרה של נפילה ה database חייב להבטיח שהוא ידע לשחזר את הפעולה ובנוסף גם להחזיק שירותי גיבוי.

2. ליניארי.

SQL :

מבנה הDBMS – האחסון הפיזי עצמו והשכבת ניהול בעצמו הם ה database אך שכבת הניהול מתקשרת עם העולם החיצוני שרוצה שירות מdatabase או להפך.

DBMS Architecture



SQL – שפה סטנדרטית לאחסון ואיחזור נתונים databases (שפת שאילתות מובנת).

שפה הצהרתית – כל database רלאציוני מבין אותה.

פקודת SELECT – שליפה מתוך הטבלה, פעולת קריאה בלבד, הפעולה * תחזיר את כל הטבלה.

Distinct – ייחודי, SELECT DISTINCT... יחזיר רק את הצירוף הייחודי.

WHERE – שליפה שמקיימת תנאי כולשהו.

LIKE '%' – כמו, כאשר ב% יהיה לנו איזה שהוא תו או כמות תווים להשוואה.

תנאים – BETWEEN, <, >, <=, >=, <>, IN, NOT, OR, AND.

שאילתות מקוננות – כל מה שחוזר מה SELECT הוא טבלה ולכן ניתן להשתמש גם ב NOT IN שזה יחזיר לך את מה שלא נמצא בטבלה ובעצם זה ימקד אותי יותר.

פקודת DEMO union – איחוד, במצב שיש נתון שמופיע בשתי טבלאות לדוגמא טבלה של עובדים וטבלה של סטודנטים וישנו סטודנט שהוא גם עובד, אזי הוא מופיע בשתי הטבלאות.

UNION – תאחד לי בבקשה בין שתי הטבלאות ובגלל פקודת SELECT תחזור לי טבלה.

לדוגמא : UNION SELECT id,...

INTERSECT – חיתוך בין טבלאות, כדי לבצע חיתוך נשתמש בתנאי IN.

EXCEPT – הפרש סימטרי בין טבלאות כדי לבצע משלים נשתמש ב NOT IN.

NULL – ריק, שליפה של רשומות בהם לא הוזנו נתונים.

טבלאות אמת של null : 3 Value Logic Truth Tables (filled)

| AND | True | False | Unknown |
|---------|---------|-------|---------|
| True | True | False | Unknown |
| False | False | False | False |
| Unknown | Unknown | False | Unknown |

| OR | True | False | Unknown |
|---------|------|---------|---------|
| True | True | True | True |
| False | True | False | Unknown |
| Unknown | True | Unknown | Unknown |

| NOT | True | False |
|---------|---------|---------|
| True | False | True |
| False | True | False |
| Unknown | Unknown | Unknown |

פקודת COALESCE – מחזירה את הערך הראשון שאינו NULL ברשימת הערכים.

לדוגמא : **SELECT id, COALESCE(lastName, firstName, 'אורח')**
FROM students



פקודה זו באה למנוע לנו שגיאה כזו –

פקודת INSERT INTO - הכנסה לרשומות, באמצעות המילה השמורה VALUES.

לדוגמא : **INSERT INTO courses**

(id,name,lecturer,year,semester) **VALUES** (66,
'databases', null, 2025, 1);

כאשר ההשמה מתבצעת בהתאמה, כלומר בשאילתה נכתוב את רשימת העמודות ואז לאחר המילה השמורה VALUES נכניס ערכים בהתאמה.

פקודת ORDER BY – החזרה של העמודה על בסיס מיון מסויים.

לדוגמא : **SELECT id,firstName FROM students ORDER BY
lastName**

כאן נשלוף את העמודות id, firstName על בסיס המיון של lastName (כאשר המיון הדיפולטיבי הוא מהקטן לגדול).

כדי להפוך את הסדר ולמייין המגדול לקטן נשתמש במילה DESC.

לדוגמא : **SELECT gender,age,lastName FROM students ORDER BY gender ASC, age
DESC**

בדוגמא זו אנו משלבים שתי מיונים, מיון על פי מגדר בסדר עולה ומיון על פי גיל בסדר יורד.

הפקודה LIMIT – יחזיר את כמות העמודות שתגיד – LIMIT 2 יחזירו 2 רשומות מהטבלה.

נבחר רנדומלית ולכן בצירוף פקודת ORDER BY הפקודה מקבלת משמעות יותר.

LIMIT 3,4 - מהמקום הרביעי (אחרי שאני עובר את 3) תביאי לי 4 רשומות.

Aggregate Function – ביצוע פעולות וחישובים מורכבים יותר כאשר מה שמוחזר זה התוצאה, כלומר הפונקציה לוקחת לבד את הנתונים, מחשבת ומחזירה לך את התוצאה.

COUNT(*) – מחזירה את מספר הרשומות בטבלה.

AVG(grade) – מחזירה את הממוצע על עמודה נבחרת (לא מחזיר null).

SUM(passed) – לסכום את כל מה שעבר.

MAX/MIX - מחזיר מקסימום\מינימום בעמודה נבחרת.

GROUP BY – קבץ לפי קבוצה, לדוגמא : **SELECT courseId, AVG(grade) FROM grades
GROUP BY courseId**

בשאילתה זו אנו במקשים את ממוצע כל הקורסים וע"י פקודת GROUP BY אנו מקבלים תוצאה מקבוצת.

HAVING – עושה תנאי על הקיבוץ שיצרתי.

QUERY EXECUTION ORDER – ניתוח המהלך בשאילה, מה קורה בdatabase כאשר אני מבקש שאילתה.

נניח ונקבל את השאילתה הבאה :

```
SELECT DISTINCT courseId, AVG(grade) FROM grades WHERE passed > 0
GROUP BY courseId HAVING AVG(grade) < 70 ORDER BY courseId,
LIMIT 2;
```

זה סדר הפעולות – FROM, WHERE, GROUP BY, HAVING, SELECT, DISTINCT, ORDER BY, LIMIT

Retrieving data from 2 tables – הוצאת נתונים משתי טבלאות.

- **SELECT * FROM students, grades** - בשיטה הנאיבית נעשה - אך צירוף זה ייתן לי כפל.

לכן נשתמש במושג **INNER JOIN** וכך זה יראה -

- **SELECT * FROM students INNER JOIN grades**
ON students.id = grades.studentId

בשיטה זו נוכל גם לצרף מיותר משתי טבלאות לדוגמא :

- **SELECT * FROM students INNER JOIN grades**
on students.id = grades.studentId INNER JOIN
courses on grades.courseId = courses.id

LEFT/RIGHT JOIN - יכניס לנו ערכי NULL למקומות שבהם לא התקבלו ערכים עדיין, לדוגמא שהגיע

- **SELECT * FROM students LEFT JOIN grades ON** סטודנט חדש שאין לו
students.id = grades.studentId

UPDATE - עדכון רשומים בעמודה קיימת.

- **UPDATE grades SET grade=78, passed=1**
WHERE studentId=111 AND courseId = 20

DELETE – מחיקה רשומות.

- **DELETE FROM grades WHERE studentId=600**
OR courseId=20

CREATE TABLE – פקודה יצירת טבלה.

- **CREATE TABLE pet (name VARCHAR(20), owner**
VARCHAR(20), species VARCHAR(20), sex CHAR(1),
birth DATE);

KEYS – הבחנה בין אישיות.

מפתח ראשי – זה עמודה או צירוף של עמודות שמזהות את האישות שלי בצורה חד חד ערכית.

מפתח ייחודי – זה עמודה או צירוף של עמודות שמזהות את האישות שלי בצורה חד חד ערכית אך יכול להיות רשומה בעלת ערך NULL.

Index – סימון, עוזר לdatabase להבין שזו רשומה חשובה ויהיו בה המון חיפושים.

דוגמא ליצירת טבלה עם KEYS –

- CREATE TABLE pet2 (petId INT PRIMARY KEY, name VARCHAR(20), ownerId INT NOT NULL, species VARCHAR(20), sex CHAR(1), birth DATE, INDEX myIndex (ownerId));

PRIMARY KEY and UNIQUE can appear right after type

INDEX (or KEY) must be defined after a comma

INTEGRITY Constraints – תנאים על הטבלה, נוכל להכניס לטבלה ערכים רק עם התנאים שנגדיר.

– CHECK (country IN ('USA', 'UK', 'Israel', 'India')) – באמצעות המילה CHECK

Foreign Key – מפתח זר, עמודה שהיא מפתח ראשי בטבלה אחרת לכן היא מפתח זר בטבלה הנוכחית.

DROP TABLE – מחיקת טבלה שלמה (delete עובד על רשומות).

ALTER – לעדכן את מבנה הטבלה (update עובד על רשומות).

שיעור 2 :**: Variables**

SET – השמה למשתנה (הכנסת ערך).

@ - מאפשר להעביר לי ערכים מפקודה לפקודה.

TEMPORARY TABLE – טבלה זמנית, כיוון שהמשתנים לא יכולים להחזיק טבלאות ניצור טבלה זמנית שבסוף הסשן תיעלם (תתמזג).

לדוגמא : **CREATE TEMPORARY TABLE tempTable2 AS (SELECT * FROM students);**

ALIASES – נתינת כינוי לפקודה מסוימת , לדוגמא :

- **SELECT * FROM students INNER JOIN grades ON students.id = grades.studentId**
- **SELECT * FROM students AS s INNER JOIN grades AS g ON s.id=g.studentId;**

כאן אני מקצר את המילה students ל s grade ל g, נתתי כינויים שמות חדשים לטבלאות שלי. ניתן גם לתת שם לשליפה שלמה כלומר אני יכול לשלוח טבלאות שלמות ולשמור אותם בשם מסוים. אני יכול בעזרת פקודה זו לתת גם כותרת. אני חייב להשתמש בAlias כאשר יש שאילתה פנימית.

Transaction - תנועה, מספר פעולות לוגיות שאני רוצה לעשות והם חייבות להתבצע כיחידה אחת, וכאשר יש נפילה אזי כל הפעולות שבוצעו חייבות להתבטל.

כלומר databases שומר בצד את הפעולות וברגע שיש נפילה הוא הולך ל commit האחרון וממשיך ממנו או מוחק בצורה הפוכה.

נשתמש במילים השמורות **COMMIT** ו **START TRANSACTION**.

```
SET @transferAmount = 1000;
START TRANSACTION;
SELECT @firstBalance := amount FROM bankBalances
WHERE userId = 777;
UPDATE bankBalances SET amount := @firstBalance -
@transferAmount WHERE userId = 777;
SELECT @secondBalance := amount FROM bankBalances
WHERE userId = 888;
UPDATE bankBalances SET amount := @secondBalance +
@transferAmount WHERE userId = 888;
COMMIT;
```

Stored Procedures – תהליכים מאוחסנים-איגוד מספר שאלות של SQL ושימוש בהם כחבילה, פרוצדורה.

חוסך זמן התחברות לשרת – לא מתקשר עם כל פקודה ופקודה אלא הכל כחבילה אחת.

דוגמא:

DELIMITER \$\$

CREATE PROCEDURE SP_student_avg

(IN stId INT)

BEGIN

SELECT AVG(grade) FROM grades WHERE
studentId = stId;

END \$\$

DELIMITER ;

קריאה לstored procedures ע"י המילה call ומחיקה ע"י המילה drop procedures.

Triggers – הדק, פעולה אחת תהיה ההדק (מה שיזניק) פעולה אחרת.

לדוגמא :

DELIMITER \$\$

CREATE TRIGGER new_grade_received

> AFTER INSERT ON grades

FOR EACH ROW

BEGIN

UPDATE students SET avg_grade = (SELECT AVG(grade) FROM grades
WHERE studentId=NEW.studentId) where id = NEW.studentId;

END\$\$

אני יוצר את הטריגר והוא יוזנק בכל פעם שאני מוסיף נתון לרשומה grade ואז אוטומטית הטריגר מעדכן את הממוצע שהיא רשומה אחרת.

View – מתאים להגבלת גישה ולשדות מחושבים.

Window Functions – כאשר אני רוצה להוסיף עוד עמודה עם עוד נתונים אשתמש במושג זה.

שיטה להוסיף מדדים נוספים לטבלה.

:Connecting to MySQL from java

כאשר אני רוצה להשתמש בנתונים בתוכנית שלי אני צטרך למשוך מה database בjava.

לדוגמא , איך עושים `SELECT *` בjava :

```
import java.sql.*;
public class Main{
    public static void main(String[] args){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            try(Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDbName", "user",
"pwd")){
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM students");
                int numColumns = rs.getMetaData().getColumnCount();
                while (rs.next()){
                    for (int col = 1; col <= numColumns; col++){
                        System.out.print(rs.getString(col) + " ");
                    }
                    System.out.println();
                }
            } catch (Exception ex){ex.printStackTrace();}
        }
    }
}
```

Reflection

Try with resources (java 7). No need to call con.close()

rs is initially located before the first row

| | | | | | | |
|-----|----|---|---|-------|-------|-------|
| 111 | 21 | 1 | 1 | Chaya | Glass | 73.33 |
| 222 | 28 | 1 | 3 | Tal | Negev | null |
| 333 | 24 | 0 | 1 | Gadi | Golan | null |
| 444 | 23 | 0 | 1 | Moti | Cohen | null |
| 700 | 26 | 1 | 2 | Maya | Levi | null |

41

לאחר שאעשה import ואפנה ל database ואכניס את השם משתמש והסיסמא ישנו מתשנה בשם statement שלה יש מתודה שקוראים לה `executeQuery` שמתודה זו מקבלת את פקודת הSQL. כעת מה שחזר זה אובייקט וממנו נוציא את המידע ע"י המתודה `getMetaData` והמתודה `getString`.

שיעור 3:

Normalization:

בניית database בצורה יעילה ואופטימלית.

איך ניקח את העולם שבחוץ ונניצג אותו בעזרת טבלאות ?

נסנן את מה שרלוונטי אלינו ומה שלא.

הגדרה : נרמול database זה תהליך שבונה את המבנה ה database באמצעות סדרת חוקים שנקראת normal forms (שישה חוקים) כדי לצמצם כפילויות ולשפר את שלמות המידע.

מושגי עזר :

תלויות – מאפיין או קבוצה של מאפיינים נקרא לו B נגיד שהיא תלויה במאפיין אחר בשם A אם יש יחס (פונקציה) כך ש $A \rightarrow B$ כלומר B תלוי A.

לדוגמא, אם ניתן לך את התז של מישור נוכל להגיד לך את השם.

כלומר אם ניתן לך ערך A לא יכול להיות שני ערכי B וזה נקרא תלות.

מפתחות – מפתח אפשרי (candidate) – סט מינימלי של מאפיינים שקובע באופן ייחודי רשומה אחת בטבלה, כלומר כל שאר המאפיינים תלויים במפתח הזה, מספר תכונות מינימליות שנותנות לי את כל השורה.

Super – Key – מפתח בלי התנאי המינימלי, כלומר הוא מפתח אך יש בו ערכים מיותרים – קבוצה של מאפיינים שבעזרתם אני יכול לגשת לטבלה אך ללא התנאי שיהיה מינימלי, מספר תכונות שנותנות לי את כל השורה.

Prime/Non Prime – תכונות שהם חלק מאיזה מפתח אפשרי או תכונות שלא שייכות לאף מפתח.

Normal Forms – שישה חוקי נרמול :

כל חוק צריך לקיים את החוק הקודם ומוסיף עליו כאשר הפתרונות לכל המצבים זה לפצל טבלאות.

1. **1NF** – מניחים שהוא תמיד יתקיים, כל תכונה (עמודה) צריכה להחזיק ערך אטומי יחיד, בנוסף אסור ערכים מחושבים למשל עמודת גיל ועמודת תאריך לידה – כך נוצרים כפילויות.

2. **2NF** – תכונות מסוג Non prime לא תלויות בתת בקבוצה חלקית של המועמדים, הם חייבות להיות תלויות בכל המועמדים (candidate), כלומר, שדות מסוג Non- prime חייבות להיות תלויות בכל המפתח ולא רק בתת קבוצה שלו.

3. **3NF** – תכונות מסוג Non prime לא יכולות להיות תלויות בתכונה או בסט של תכונות שהוא לא super-key, כלומר היא צריכה להיות תלויה ב super-key.

4. **3.5NF** – BCNF – רק מכאן אנחנו מתחילים לדבר על תכונות שהם prime, ולכן אם אני מזהה שכל התכונות שלי הם prime אזי 1,2,3, מתקיים באופן דיפולטיבי, משלים את חוקים 2 ו 3, לכל 2 קבוצות- אם קיימת תלות ביניהם אזי בהכרח אחת הקבוצות היא super – key. כלומר, האם קיימת כזו קבוצה שהיא תלויה בקבוצה אחרת והיא לא super-key אם כן אזי זה מקיים BCNF (אם אין כלל תלויות זה גם יעמוד ב BCNF).

5. **4NF** – אסור שיהיו תלויות רב ערכיות (Multivalued Dependency) כלומר, כאשר יש יחס בין זוג דברים שמתאים לגורם שלישי לדוגמא $A1 - B1 = C1$ וגם קיים בו $B2 \rightarrow C1$ אזי זו תלות רב ערכית, כלומר ישנם 2 מקומות שונים שיכולים להביא אותו C1. כל עמודה לא גוררת כמה שורות.

אותו מקור מביא אותי לשתי תמונות שונות, וזו בעיה – יש לי שיכפול נתונים.

| | A | B | C |
|---|----|----|----|
| x | a1 | b1 | c1 |
| y | a1 | b2 | c2 |
| z | a1 | b1 | c2 |
| w | | | |

הנה דוגמא בעיתית :

6. 5NF – למצבים נדירים – ננסה לייעל כמה שיותר בהתאם לכל מצב.

סדר פעולות למציאת candidate :

נתבונן בצד ימין של התלויות ונראה איזו תכונה לא מופיעה משמע שיש תכונה שלא תלויה בכלום והיא candidate key.

נעבור תכונה תכונה ונראה אם ניתן לקבל את כולם (לגזור), אם כן היא candidate key.

לאחר מכן נעבור בזוגות ובבדוק אם ניתן לקבל את כולם וכך גם בשלוש.

כעת נבדוק האם קיימות תכונות non prime כי אם לא קיימות תכונות כאלה ישר נעבור לבדוק סיווג של 3.5NF.

: XML and JSON

פורמטים להעברת מידע מהdatabase.

התקן המפורסם ביותר נקרא XML – שפת סימון הניתנת להרחבה, פורמט להעברת נתונים, היררכית ורגישה גודלות/קטנות.

לדוגמא :

```
<University>
  <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <id>111</id>
    <age>21</age>
    <Address>
      <Street>Hatamr 5</Street>
      <City>Ariel</City>
      <Zip>40792</Zip>
    </Address>
  </Student>
</University>
```

ישנו שורש אחד שהוא פותח וסוגר – בדוגמא שלנו זה <University>.

| | |
|--------|---|
| < | < |
| > | > |
| & | & |
| ' | ' |
| " | " |

סימונים בוליאניים :

XML in Java – שליפת XML מ JAVA :

קבלת הנתונים מהdatabase – `import org.w3c.dom.*`.

לאחר שיצרנו אובייקט אני רוצה להכניס את המידע מהXML לתוך האובייקט.

אנחנו נעבוד על אובייקט בשם doc שלשם נקבל את הקובץ XML ולו יש מספר מתודות.

בעיקר נעבוד עם הפונקציות getElement.

לאחר מכן נכניס לרשימה ועל נכניס את הנתונים על האובייקטים בswitch case.

```
File inputFile = new File("student.xml");
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(inputFile);

System.out.println("Root element : " + doc.getDocumentElement().getNodeName()); //just print root (university)

NodeList nodeList = doc.getDocumentElement().getElementsByTagName("Student");
for (int studentIdx = 0; studentIdx < nodeList.getLength(); studentIdx++){
    Node studentNode = nodeList.item(studentIdx);
    if (studentNode.getNodeType() == Node.ELEMENT_NODE){
        Element element = (Element) studentNode;
        Student student = new Student();
```

XPATCH – עוזר לי להגיע לנקודה ספציפית בXML בלי לעבור על כך העץ.

הוא עובד כמו גישה לקובץ בתוך תיקיה – לדוגמא בקובץ XML הנ"ל אם אגש לפקודה הבאה –

- **University/Student[2]/Address/City**

אני אקבל את ירושלים- `<City>Jerusalem</City>`


ניתן גם להוסיף תנאים בבקשת XPATCH.

XPATCH in Java – מאוד דומה לעבודה של XML עם java אך כן אני אכין את הבקשה ואז אמיר אותה ל string.

```
File inputFile = new File("student.xml");
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document xmlDoc = builder.parse(inputFile);

XPathFactory xPathfactory = XPathFactory.newInstance();
XPath xpath = xPathfactory.newXPath();
XPathExpression expr = xpath.compile("University/Student[2]/Address/City");
String city = (String)expr.evaluate(xmlDoc, XPathConstants.STRING);
```

XQuery – סוג של SQL עם XML.

- `for $x in /University/Student` : 

Validation – הגנה, תנאים כדי ליצור בקשת XML חוקית.

ישנם שני סוגים של פרוטוקולים:

- DTD
- XML Schema (XSD) – בו נתמקד.

XML Schema (XSD) – מגדיר איזה טיפוסים XML שלי יכול לקבל ומה נחשב לבקשת XML תקינה.

בהתחלה הפרוטוקול בודק את תקינות הטיפוס שהוא מקבל באתר אינטרנט מסוים.

לאחר מכן הוא בודק את ההתאמה בין המיקום שהיה בבקשה לבין מה שהוא ציפה לקבל בפרוטוקול.

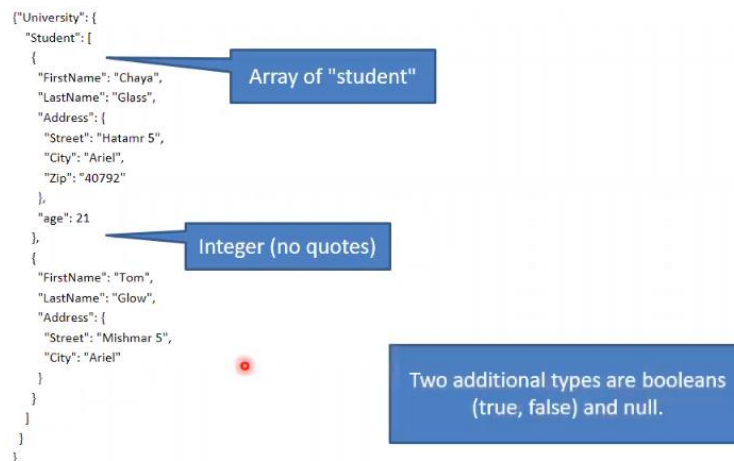
כאשר יש שגיאות בדרכ כלל נתקן את XML שיתאים לXSD.

שיעור 4:

JSON – java script object notation: תקן להעברת נתונים בין שרת לשרת – קריא כמו XML אך קצת יותר פשוט, תחליף יותר מתקדם מXML.

בנוי מ { בשונה מXML שבנוי מ < .

לדוגמא:



JSON in Java – בצורה דומה לעבודה של XML כך גם ב JSON יש ספריות ייעודיות, לולאות, ומתודות המגיעות מספריות.

לדוגמא:

```

String jsonTxt = new String(Files.readAllBytes(Paths.get("students.json")));
JSONObject json = new JSONObject(jsonTxt);
JSONArray jsonStudentArray = json.getJSONObject("University").getJSONArray("Student");
for (int studentIdx = 0; studentIdx < jsonStudentArray.length(); studentIdx++){
    JSONObject currentStudent = jsonStudentArray.getJSONObject(studentIdx);
    Student student = new Student();
    studentList.add(student);
    JSONArray studentInner = currentStudent.names(); //array of keys only!
    for (int stInnerIdx = 0; stInnerIdx < studentInner.length(); stInnerIdx++){
        String currentKey = studentInner.getString(stInnerIdx);
        switch (currentKey){
            case "FirstName": student.firstName = currentStudent.getString(currentKey); break;
            case "LastName": student.lastName = currentStudent.getString(currentKey); break;
            case "id": student.id = currentStudent.getInt(currentKey); break;
            case "age": student.age = currentStudent.getInt(currentKey); break;
            case "Address":
                Address address = new Address();
                student.address = address;
                JSONObject addressObject = currentStudent.getJSONObject(currentKey);
                if (addressObject.has("Street"))
                    address.street = addressObject.getString("Street");
                if (addressObject.has("City"))
                    address.city = addressObject.getString("City");
                if (addressObject.has("Zip"))
                    address.zip = addressObject.getString("Zip");
                break;
        }
    }
}

```

JSON Schema – לא כ"כ בשימוש.

JSON Vs.XML - שניהם ניתנים לקריאה, היררכיים.

הייתרון בJSOM שהוא יותר קצר ויש בו מערכים, והיתרון הגדול שלו שהוא יכול להיות מפורסם באמצעות JS כלומר JAVA SCRIPT ינתח אותו ביותר קלות כי הוא נועד בשבילו.

NoSQL

לא רק SQL, מתייחס למסדי נתונים שלא מיוצגים בטבלה (למשל גרף), יותר גמיש – אפשר להוסיף לו נתונים ועמודות ביותר גמישות וקלות, מהיר ובעל יכולת להתרחב.

הוא תומך ב big data - אוסף מידע עצום שאני יכול לאחסן, לשלוק ולהסיק מסקנות ביעילות.

יכולת התשאול פה מוגבלות כאן קצת, והוא לא יכול להבטיח את התכונות (ACID) שהSQL עמד בהם, אבל הוא כן תומך בBASE.

– BASE

- **Basically Available:** data is mostly available.
- **soft State:** state may change even with no updates (since older updates are still propagating).
- **Eventual consistency:** if we let the data propagate enough time, it will become consistent.

ישנם כמה סוגים של NoSQL :

- Key-Value.
- Wide column - בסיסי נתונים שמבוססים על עמודות – מאפשר לנו יכולת לאחסן בצורה גמישה ואין צורך להחזיק עמודות ריקות כמו בSQL.
- Document – שמירת נתונים ע"י קבצים כגון JSON,XML.
- Graph - שמירת נתונים ע"י user כלומר גרף שמייצג את מה שה user יצר.
- Search – מסדי נתונים שתומכים במנועי חיפוש.

CAP theorem - ניתן לקיים רק 2 מתוך 3 התכונות הבאות :

- Consistency – עקביות, ביצוע כל הפעולות ברצף.
- Availability – זמינות, כל בקשה מקבלת תשובה.
- Partition tolerance – המערכת ממשיכה לפעול גם שכמה הודעות מתעכבות בצמתים.



נעבור על סוגי ה־NoSQL:

Key Value Store - לכל פריט יש key ו־value . אחסון מהיר, קל לשימוש, גמיש.

כל התשאול מתבצע באמצעות ה־key (מהיר יותר).

פקודות בסיסיות – set, get, del .

INCR – להוסיף , INCRBY - כמה לקדם את הערך הנמצא במפתח.

פקודות על רשימה (LIST) - RPUSH - דחיפה מימין (מהסוף), בהתאמה LPUSH , LRANGE – תציג לי את כל הרשימה.

פקודות על טבלאות גיבוב (Hashes), נועד בשביל להכיל הרבה מידע במפתח אחד – HSET - הכנסה , HGET – הוצאה, HMSET - הכנסה של כמה פריטים , HGETALL – להחזיר את הכל.

הפקודה KEYS – פקודה שפועלת על מפתחות עם תנאי.

עבודה עם קבוצות – ברשימה מותר כפילות, כלומר מותר לאברים לחזור על עצמם, בקבוצה הוא יתווסף רק פעם אחת לא משנה כמה פעמים נבצע את פקודת ההוספה.

EXPIRE – עוד זמן מוגבל שאקציב הערך יימחק מה database, הערך יופג.

TTL - time to live - כמה זמן נשאר לערך לחיות.

Wide – Column store - גמישות בעמודות, אין חובה שלכולם יהיה את אותה המבנה.

יש לי יכולת להוסיף עוד מאפיינים פר רשומה ובצורה פרטית ולא כללית.

הדאטה בייס Cassandra - יש לו שפת שאילתות שקוראים לה CQL – אין שם join , ואין שם אפשרות לעשות שאילתה בתוך שאילתה.

RMDB – הדגש הוא המהירות ולא היעילות, כלומר השאילתות מגדירות את הטבלאות.

Data Model – מודל הנתונים מורכב מהדברים הבאים :

Cluster – אשכול, מסד נתונים מבוצר שיושב על כמה שרתים שעליהם databasen נמצא, כיוון שה database יושב כל כמה שרתים אזי איך נוכל לדעת מאיזה שרת נצטרך למשוך מידע, בשביל זה יש את פונקציית hashing שמקבלת key (מספר) ויודעת למפות את המספר לשרת בו הוא נמצא ע"י התחום.

Keyspace – מרחב המפתחות שלי, הנושא עצמו, לדוגמא אוניברסיטה וכו', הוא מאגד מתחתיו כמה טבלאות.

Column family - משפחה של עמודות, הטבלאות עצמם.

| 103 | email | name | tel | tel2 |
|-----|----------|------|------|-------|
| | karl@a.b | karl | 6789 | 12233 |

Keys and column – לכל מפתח יש עמודות -

פקודות :

CREATE KEYSPACE – יצירת database, לדוגמא :

Replication refers to how the data is replicated across different nodes

➤ CREATE KEYSPACE university WITH
REPLICATION = {'class': 'SimpleStrategy',
'replication_factor': 2};

JSON style

USE - להשתמש במה שיצרו

➤ CREATE TABLE students (id INT PRIMARY KEY,
firstName VARCHAR, lastName VARCHAR, age
INT);

CREATE TABLE – יצירת טבלאות, לדוגמא :

Like SQL. But there is no need to specify the size for VARCHAR.

הערכים שהכנסנו יהיו העמודות.

Exactly like SQL...

➤ INSERT INTO students (id, firstName, lastName,
age) VALUES (111, 'Chaya', 'Glass', 21);

INSERT INTO – הכנסת ערכים, לדוגמא :

Must use single quotes!

לא חייבים להכניס ערכים לכל העמודות הקיימות, למה שלא נכניס הוא יקבל אוטומטית את הערך .NULL

WHERE on a key is ok

WHERE – שליפה לפי מפתח :

➤ SELECT * from students WHERE id=111;

אם ננסה לשלוף ללא מפתח נקבל שגיאה (המפתח צריך להיות ספציפי).

Cassandra storage method - ישנו data center שבו מאוחסנים השרתים שאיתם עובד databasen, כאשר אנו כותבים נתונים במקביל, השרת שעליו אנו עובדים מעתיק את הנתונים לעוד שלושה שרתים.

זה נועד לצורך גיבוי או מקרה בו אחד השרתים לא זמין בזמן שפונקציית hashing מחפשת אותו ע"י המיפוי.

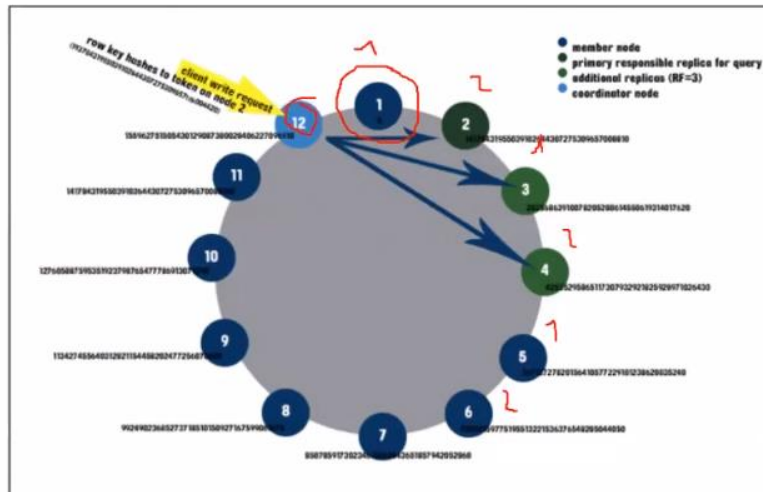


Figure 1 A 12 node cluster using RandomPartitioner and a keyspace with Replication Factor (RF) = 3, demonstrating a client making a write request at a coordinator node and showing the replicas (2, 3, 4) for the query's row key

<https://www.hakkalabs.co/articles/how-cassandra-stores->

ה partition key אומר לנו באיזה מחשב אני ממופה, באיזה מחשב אני מאוחסן, בנוסף יש תת מפתח שנקרא clustering key שעוזר לי למיין ולסדר את המידע בתוך אותו מחשב, לדוגמא :

```
CREATE TABLE crossfit_gyms (
  gym_name text,
  city text,
  state_province text,
  country_code text,
  PRIMARY KEY (gym_name)
);
```

```
CREATE TABLE crossfit_gyms_by_location (
  country_code text,
  state_province text,
  city text,
  gym_name text,
  PRIMARY KEY (country_code, state_province, city, gym_name)
);
```

partitioning key

clustering keys

retrieving the node and
the partition

Sorting within the partition

כאשר ה gym_name הוא ה partition key ואילו מצד ימין אנו מוספים מאפיינים שנועדו לסדר את הנתונים בתוך המחשב הפנימי והם ה clustering key.

כך מצהירים על זה :

Multiple primary keys (composite key), note the order

```
CREATE TABLE grades (studentid INT, course TEXT, grade FLOAT, PRIMARY KEY(studentid, course));
INSERT INTO grades(studentid, course, grade) values(111, 'intro to intro', 95);
```

Partition Key

Clustering key

עד הפסיק הראשון זה יהיה ה partition key והוא יכול להיות מורכב מכמה דברים.

בכל שאילתה שאנו כותבים ה partition key חייב להיות מסופק.

- Cassandra Vs. RDBMS

ישנם מספר הבדלים בין Cassandra לבין database ישראלי.

cassandra אין מקום אחד שבו שמחזיק את כל המידע – אם משהו יפול אז לא כל ה database יתרוסק.

הזמינות בcassandra גבוהה יותר.

בcassandra הdata model דינאמי.

בcassandra אני תמיד יכול להגדיל את השרתים והאחסון כך שהוא יכול להחזיק big data.

| Property | Cassandra | RDBMS |
|---------------------------|---|---|
| Core Architecture | Masterless (no single point of failure) | Master-slave (single points of failure) |
| High Availability | Always-on continuous availability | General replication with master-slave |
| Data Model | Dynamic; structured and unstructured data | Legacy RDBMS; Structured data |
| Scalability Model | Big data/Linear scale performance | Oracle RAC or Exadata |
| Multi-Data Center Support | Multi-directional, multi-cloud availability | Nothing specific |

Document store - קצת דומה ל key and value רק ה key שלו הוא מסמך מורכב.

כאשר בתוך הvalue יכול להיות עוד מסמך שלם.

המסמכים נכתבים בפורמט של JSON כאשר כאן השאילתות יודעות לתשאל את הערך את הvalue ולא רק את הkey.

MongoDB – מלשון המילה עצום – יודע להתמודד עם נפחים גדולים של מידע.

הפקודות:

➤ use University – יצירת ה database לדוגמא:

Db.dropDatabase () – מחיקת database.

Collection – ישות שמקבילה לטבלאות, לדוגמא, ניצור טבלה שנקראת סטודנט:

➤ db.createCollection("students", { capped : true, size : 6142800, max : 10000, autoIndexID : true })

Docs – מקביל לשורות בטבלה.

הכנסת רשומה לתוך collection שקוראים לו סטודנט ע"י הפקודה insert ובאמצעות JSON –

```
➤ db.students.insert({"FirstName": "Chaya",
  "LastName": "Glass",
  "id": "111",
  "age": "21",
  "Address": {
    "Street": "Hatamr 5",
    "City": "Ariel",
    "Zip": "40792"}
})
```

שאליות :

➤ db.students.find()

Find – יחזיר הכל, לדוגמא :

```
{ "_id" : ObjectId("589afa8c44a5653a862dd692"), "FirstName" : "Chaya", "LastName" : "Glass", "Id" : "111", "age" : "21", "Address" : { "Street" : "Hatamr 5", "City" : "Ariel", "Zip" : "40792" } }
{ "_id" : ObjectId("589afa9244a5653a862dd693"), "FirstName" : "Tom", "LastName" : "Glow", "Address" : { "Street" : "Mishmar 5", "City" : "Ariel" } }
{ "_id" : ObjectId("589afa9244a5653a862dd694"), "FirstName" : "Tal", "LastName" : "Negev", "Address" : { "Street" : "Yarkon 26", "City" : "Jerusalem" } }
```

➤ db.students.find().pretty()

Find().pretty() – מסדר את הJSON יפה.

החזרה על בסיס תנאי - יחזיר את כל הסטודנטים בשם טל.

➤ db.students.find({"FirstName": "Tal"})

```
{ "_id" : ObjectId("589afa9244a5653a862dd694"), "FirstName" : "Tal", "LastName" : "Negev", "Address" : { "Street" : "Yarkon 26", "City" : "Jerusalem" } }
```

And, Or – החזרת כל המסמכים שעונים על התנאי הבא (and) :

➤ db.students.find({\$and: [{"FirstName": "Tal"}, {"LastName": "Negev"}]})

החזרת כל המסמכים עם תנאי or :

➤ db.students.find({"FirstName": "Tom", \$or: [{"LastName": "Negev"}, {"LastName": "Glow"}]})

Projection – הטל, כאשר אני רוצה לקבל איזה מימד מהנתונים שלי, בחירת מימד מסויים של נתונים.

לדוגמא :

בשאלית זו אנחנו מבקשים שתחזיר לי את כל המסמכים שבהם השם הפרטי הוא טים אבל בנוסף תחזיר לי מתוך ה JSON רק את השדה הזה ולא את כל המסמך.

➤ db.students.find({"FirstName": "Tim"}, {"FirstName": true})

```
{ "_id" : ObjectId("589afa9244a5653a862dd693"), "FirstName" : "Tim" }
```

Update – עדכון, לדוגמא, נעדכן כל מסמך JSON בו השם הפרטי טום נעדכן לטים (וכיוון שאנו מעדכנים את המסמך אנחנו צריכים לציין גם את שאר השדות אחרת המסמך יתעדכן בלעדיותם):

Syntax: db.collection.update(query, update, options)

➤ db.students.update({"FirstName": "Tom"}, {"FirstName": "Tim", "LastName": "Glow", "Address": {"Street": "Mishmar 5", "City": "Ariel"} })

MongoDB will search for a FirstName="Tom", and change the whole document to be: {"FirstName": "Tim", "LastName": "Glow", "Address": {"Street": "Mishmar 5", "City": "Ariel"}}

בדוגמא זו נשתמש ב set ונעדכן רק שדה ספציפי בתוך המסמך (ולא צריך לציין את שאר השדות):

➤ db.students.update({"FirstName": "Tom"}, {"\$set: {"FirstName": "Tim"}, {multi: true}})

Set will leave all other fields unchanged.

If we don't set multi to true, MongoDB will only set the first item it finds

אם לא נשים את multi כ true אזי הוא יעדכן אר במסמך הראשון שהוא ימצא.

Map- Reduce Paradigm – בעולם מרובה משאבים, שרתים ננסה לחלק את הבעיה לבעיות קטנות ולאחר מכן לעבד הכל לתוצאה כללית.

לדוגמא, אם נרצה לספור את כמות האנשים במדינה יהיה יעיל יותר שכל עיר תספור את כמות התושבים שלה ולאחר מכן נחבר הכל.

בעל כמה מאפיינים:

Mapper - מפצל את המידע ומחלק אותו לכמה תהליכים.

Shuffle and sort/Grouping – סידור המידע לפני ביצוע תחילת העבודה.

Reduce – כל עובד מבצע את העבודה במקביל.

כלומר תהליך העבודה יתבצע כאשר המערכת קודם כל תמפה את הנתונים בצורה ממוינת ע"י תנאי מסוים ולאחר מכן תחלק בצורה מקבילית כל את ה"מפה" לעובדים.

לדוגמא נתון לי המסמך הבא המייצג הזמנות:

```
{
  _id: ObjectId("50a8240b927d5d8b5891743c"),
  cust_id: "abc123",
  ord_date: new Date("Oct 04, 2012"),
  status: 'A',
  amount: 25,
  items: [ { sku: "chocolates", qty: 5, price: 2.5 },
            { sku: "oranges", qty: 5, price: 2.5 } ]
}
```

SKU = Stock Keeping Unit
is an item identifier.

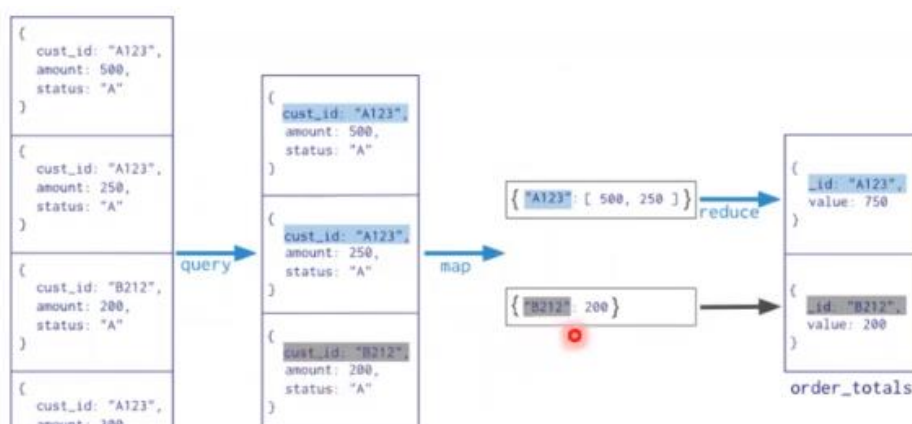
אנו רוצים לקבל את הסכום ששולם עבור כל לקוח שנמצא בסטטוס 'A'.

נעשה זאת כך:

```
Collection
  ↓
db.orders.mapReduce(
  map → function() { emit( this.cust_id, this.amount ); },
  reduce → function(key, values) { return Array.sum( values ); },
  query → { query: { status: "A" },
  output → out: "order_totals"
  )
```

ה order מייצג את שם המסמך שלי ועליו אני מפעיל mapReduce.

כעת ה query שלי יעבוד רק על מי שהסטטוס שלו A, - אני מצמצם את האפשרויות ומכין את הmap.



כעת ב reduce נחלק את זה לעובדים שפשוט יסכמו את כל ה values ולאחר מכן נחזיר את order_totals.

➤ `db.order_totals.find()` – כדי לקבל את התוצאה נעשה find על order_totals –

```
{_id: 'Cam Elot', value: 60 }
{_id: 'Don Quis', value: 155 }
{_id: 'Busby Bee', value: 125 }
{_id: 'Ant O. Knee', value: 95 }
```

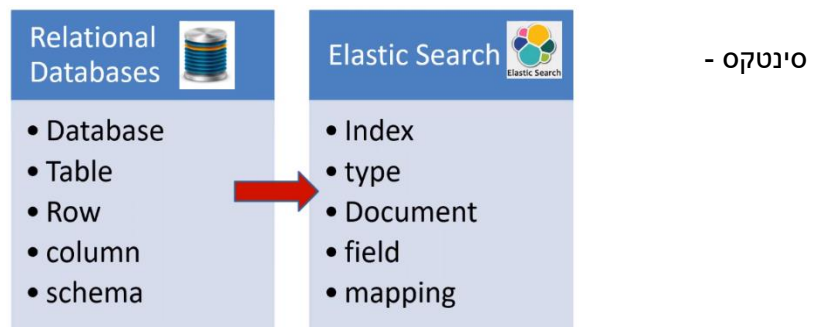
שיעור 5- המשך מעבר על סוגי NoSQL.

Search Engine Database – databases מסוגי מנועי חיפוש.

תת סוג של documents store כי אנחנו מאחסנים טקסט בחיפוש שלנו אך ייחודי יותר כיוון שאני מחפש את התוצאה הרלוונטית ביותר בצורה מדורגת.

Elastic Search – database שהוא real-time, לוקח בערך שניה מהרגע שהעליתי מסמך עד שהוא יופיע בתוצאות החיפוש.

נותן פתרונות לחברות שמנועי חיפוש זה לא המוצר העיקרי שלהם.



פקודות:

Adding documents – אני לא צריך ליצור אינדקס (database) ואז להעלות אותו אלא אפשר ישירות:

ע"י הפקודה XPUT -

dbServer address index type docID

```
➤ curl -XPUT "http://localhost:9200/university/students/111" -H
"Content-Type: application/json" -d '{"FirstName": "Chaya",
"LastName": "Glass", "age": "21", "Address": {"Street":
"Hatamr 5", "City": "Ariel", "Zip": "40792"}}'
```

פקודה נוספת להכנסה בשם XPOST כאשר אני לא מספק id והindex אוטומטים ייתן להם id:

POST without id, will
generate id automatically

```
➤ curl -XPOST http://localhost:9200/university/students -H
"Content-Type: application/json" -d '{"FirstName": "Tal",
"LastName": "Negev", "age": "28"}'
```

Generally, in REST API, PUT is idempotent ($n\{msg\} = \{msg\}$),
and POST isn't. (What will happen if we send each of the
above messages twice?)

XGET – שליפה לפי id :

```
➤ curl -XGET "http://localhost:9200/university/students/333"
{"_index": "university", "_type": "students", "_id": "333", "_version": 1, "_seq_no": 1, "_
primary_term": 1, "found": true, "_source": {"FirstName": "Gadi", "LastName":
"Golan", "age": "24"}}
```

Note all the metadata.

```
➤ curl -I -XHEAD http://localhost:9200/university/students/333
- will return: OK
```

```
➤ curl -XDELETE "http://localhost:9200/university/students/333"
```

XDELETE – מחיקת המסמך :

UPDATE – הוספת שדה. לדוגמא, ניקח רשומות קיימות ונוסיף להם תיאור :

```
➤ curl -XPOST http://localhost:9200/university/students/111/_update -H
"Content-Type: application/json" -d '{"script": "ctx._source.description =
\\\\"Likes learning but gets board very quickly. Doesn't enjoy trips that
much.\\\\""}'
➤ curl -XPOST http://localhost:9200/university/students/333/_update -H "Content-Type:
application/json" -d '{"script": "ctx._source.description = \\\\"Doesn't show-up to
lessons, but is very smart and learns a lot.\\\\""}'
➤ curl -XPOST
http://localhost:9200/university/students/IA9AInsBL04IeaKD9LL9/_update -H
"Content-Type: application/json" -d '{"script": "ctx._source.description =
\\\\"Doesn't know anything. Goes on trips all day, never showed-up to a single
lesson.\\\\""}'
```

Search – החיפוש עצמו , לב databases.

חיפוש בסיסי ע"י המילה search יחזיר את כל הרשומות - `curl -XGET "http://localhost:9200/university/students/_search"`
חיפוש בעזרת query string תחזיר רק התוצאה המתאימה –

```
➤ curl -XGET http://localhost:9200/university/students/_search?q=LastName:Negev"
```

חיפוש בעזרת request body אשר נותן אפשרויות סינון יותר רחבות :

Match – מחזיר את המסמכים שמתאימים לטקסט שסופק לה, סטנדרטי.

מכיל אופציות לfuzzy matching – חיפוש עם סובלנות לשגיאות.

לדוגמא, נתאים את ה description שהוספנו ב update ל query שנספק עכשיו –

```
curl -XGET "http://localhost:9200/university/students/_search" -H "Content-Type: application/json"
-d '{"query": {"match": {"description": {"query": "very smart quickly"}}}}
```

It is interpreted as "very" OR "smart" OR "quickly"

בעצם חיפשנו סטודנטים שבתיאור שלו מופיע very smart quickly אך אין סטודנט עם סטרינג כזה לכן הוא מחפש או very או smart או quickly .

התוצאות שנקבל בעלות score והם יתועדפו מה score הגבוה לנמוך, כאשר הדרך לקביעת score גבוהה הם כמות המילים המתאימות, כך אנו מקבלים את התוצאה הכי רלוונטית.

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2,
      "relation": "eq"
    },
    "max_score": 1.5127167,
    "hits": [
      {
        "_index": "university",
        "_type": "students",
        "_id": "111",
        "_score": 1.5127167,
        "_source": {
          "FirstName": "Chaya",
          "LastName": "Glass",
          "age": 21,
          "Address": {
            "Street": "Hatamr 5",
            "City": "Ariel",
            "Zip": "40792"
          },
          "description": "Likes learning but gets board very quickly. Doesn't enjoy trips that much."
        }
      },
      {
        "_index": "university",
        "_type": "students",
        "_id": "333",
        "_score": 1.4658242,
        "_source": {
          "FirstName": "Gadi",
          "LastName": "Golan",
          "age": 24,
          "description": "Doesn't show-up to lessons, but is very smart and learns a lot."
        }
      }
    ]
  }
}
```

כאשר נרצה להתאים או לחפש את המשפט בדיוק (המילים בסדר מסוים) נשתמש בmatch_phrase.

חיפוש בעזרת bool query – כאשר אני רוצה למצוא במסמכים שלי נתון המורכב מכמה תנאים נשתמש במילה bool וכמה תנאים : must זה בלוק שהתנאים בו חייבים להימצא במסמך.

Should זה בלוק שלא חייב להופיע בתוצאות אך ישפר את הדירוג שלו אם כן.

```
POST employees/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "position": "manager"
          }
        },
        {
          "range": {
            "experience": {
              "gte": 12
            }
          }
        }
      ],
      "should": [
        {
          "match": {
            "phrase": "versatile"
          }
        }
      ]
    }
  }
}
```

ניתן לבצע את הסינון הנ"ל גם באמצעות המילה filter – שהוא יסמן רק את המסמכים שהשדה מתאים לתנאים שאני רוצה אך בשימוש ב filter הדירוג לא רלוונטי.

לדוגמא :

➤ curl -XGET "http://localhost:9200/university/students/_search" -d"

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "match": {
            "Address.City": "Ariel"
          }
        },
        {
          "range": {
            "age": {
              "lte": 30
            }
          }
        }
      ]
    }
  }
}
```

Boolean combination of several queries.

All students living in Ariel

Students under 30

```
{
  "took": 6,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0.0,
    "hits": [
      {
        "_index": "university",
        "_type": "students",
        "_id": "111",
        "_score": 0.0,
        "_source": {
          "FirstName": "Chaya",
          "LastName": "Glass",
          "age": 21,
          "Address": {
            "Street": "Hatamr 5",
            "City": "Ariel",
            "Zip": "40792"
          },
          "description": "Likes learning but gets board very quickly. Doesn't enjoy trips that much."
        }
      }
    ]
  }
}
```


Must not – למסמך שיגיע בתוצאה אסור להכיל את התנאי המסופק, גם במקרה זה הדירוג (score) אינו רלוונטי.

לסיכום :

| keyword | meaning | Scoring the results |
|------------|--|---------------------|
| Should | Finding the text will increase the score | Yes |
| Must | The results must contain the string | Yes |
| filter | The results must contain the string | No |
| → Must not | The results must not contain the string | no |

Information Retrieval Document Ranking – איך מדרגים ממאגר מסמכים גדול את המסמך הכי רלוונטי ?

Tf-idf – הינו האלגוריתם שאחראי לדירוג, הוא עובר על כל מילה ומילה בשאלה ומדרג אותה. הוא מחולק ל-2:

TF – ספריה של מספר המופעים של מילה במסמך כאשר הוא מנורמל למספר המילים שיש בתוך המסמך, כלומר אם יש יותר מופעים של המילה במסמך קצר יותר אזי המילה חשובה יותר. לדוגמא, אם יש מילה שמופיעה במסך בעל שלוש מילים אזי המסמך הזה ככל הנראה רלוונטי לי כרגע.

IDF – היפוך תדירות במסמכים, נחלק את כמות המסמכים שיש לי בכמות המסמכים שהמילה שאני מחפש מופיעה בהם, וככל שהמילה שכיחה יותר במסמכים היא פחות משמעותית ולכן המילה תהיה פחות משמעותית.

לדוגמא, המילה is תופיעה בהמון מסמכים ולכן המשמעות שלה נמוכה.

הנוסחה הכללית :

appears in.

$$tfidf(d) = \sum_{k=0}^{|Q|} \frac{\#k \text{ in } d}{|d|} \log\left(\frac{|D|}{\#D \text{ with } k}\right)$$

k: word in document/query
 Q: set of words in query

Number of instances of k in d
 Total number of documents
 number of documents with the word "k"
 Total Number of words in d
 TF
 IDF

אז למעשה ניקח את השאילה וניצור טבלה כאשר כל עמודה מייצגת מילה, נעבור על כל המסמכים ונספור כמה פעמים מופיעה כל מילה ונעדכן בטבלה.

לאחר שנקבל את כל הנתונים נציב בנוסחה הנ"ל וכך נקבל את התיעדוף.

לדוגמא :

| | Who | Is | The | President | Of | United | States | #words |
|-----------|-----|----|-----|-----------|----|--------|--------|--------|
| D1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 6 |
| D2 | 0 | 0 | 3 | 0 | 2 | 1 | 0 | 15 |
| D3 | 1 | 1 | 1 | 0 | 1 | 3 | 1 | 14 |
| D4 | 1 | 0 | 2 | 0 | 1 | 0 | 0 | 11 |
| #D with k | 2 | 2 | 3 | 1 | 3 | 3 | 2 | |

number of documents with the word "k"

- Q: Who is the president of the united states?
- D1: Donald Trump is United States' president.
- D2: We are the most united out of all the people and of all the places.
- D3: The United States of America is united again, who is more united than it?
- D4: Who would like to take the box out of the kitchen?

הצבה בנוסחה :

$$tfidf(d) = \sum_{k=0}^{|Q|} \frac{\#k \text{ in } d}{|d|} \log\left(\frac{|D|}{\#D \text{ with } k}\right)$$

| Doc | Tf-idf score |
|-----|---|
| D1 | $(1/6) * \log(4/2) + (1/6) * \log(4/1) + (1/6) * \log(4/3) + (1/6) * \log(4/2) = 0.736$ |
| D2 | $(3/15) * \log(4/3) + (2/15) * \log(4/3) + (1/15) * \log(4/3) = 0.166$ |
| D3 | $(1/14) * \log(4/2) + (1/14) * \log(4/2) + (1/14) * \log(4/3) + (1/14) * \log(4/3) + (3/14) * \log(4/3) + (1/14) * \log(4/2) = 0.363$ |
| D4 | $(\log(4/2) + 2 * \log(4/3) + \log(4/3)) / 11 = 0.204$ |

התיעדוף הגבוהה ביותר שקיבלנו הוא המסמך D1.

Database מסוגי גרף :

בנוי מעיקרון שמירת נתונים בגרף המורכב מצמתים והקשר בין הצמתים שזה מאפשר לנו סוגי שאילות שונים.

RDF - Resource Description Framework – מודל סטנדרטי, אך הייחוד בו שהכל מורכב משלוש – נושא, משוא, ומושא (Subject, predicate, Object).

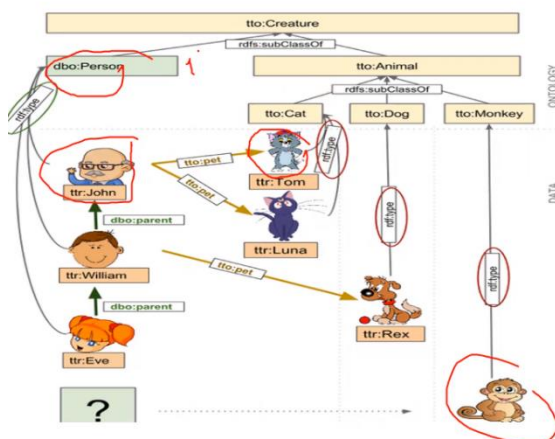


ובעצם שלוש כאלה מאוחסנות בטבלה.

Jena - database מבוסס גרפים שבו אנו נתרכז בשפת השאילות SPARQL.

Ontology – הגדרה שמגדירה את כל סוגי הישות שקיימות בdatabase כך שכל שלשה שנכנסת לטבלה היא חוקית ועומדת באונטולוגיה של אותו עולם (למשל קשרים בין חיות וכו').

לדוגמא היחסים האלו בין בני אדם לחיות :



| s | p | o |
|-----------------|---------------|--------------|
| ttr:Eve | dbo:parent | ttr:William |
| ttr:Eve | dbp:birthDate | "2006-11-03" |
| ttr:Eve | dbp:name | "Eve" |
| ttr:Eve | tto:sex | "female" |
| ttr:Eve | rdf:type | dbo:Person |
| ttr:John | dbp:birthDate | "1942-02-02" |
| ttr:John | dbp:name | "John" |
| ttr:John | tto:pet | ttr:LunaCat |
| ttr:John | tto:pet | ttr:TomCat |
| ttr:John | tto:sex | "male" |
| ttr:John | rdf:type | dbo:Person |
| ttr:LunaCat | dbp:name | "Luna" |
| ttr:LunaCat | tto:color | "violet" |
| ttr:LunaCat | tto:sex | "female" |
| ttr:LunaCat | tto:weight | "4.2" |
| ttr:LunaCat | rdf:type | tto:Cat |
| ttr:RexDog | dbp:name | "Rex" |
| ttr:RexDog | tto:color | "brown" |
| ttr:RexDog | tto:sex | "male" |
| ttr:RexDog | tto:weight | "8.8" |
| ttr:RexDog | rdf:type | tto:Dog |
| ttr:SnuffMonkey | dbp:name | "Snuff" |
| ttr:SnuffMonkey | tto:color | "golden" |
| ttr:SnuffMonkey | tto:sex | "male" |

ייצוג של הגרף בטבלת RDF :

פקודות :

- **SELECT * WHERE {?s ?p ?o}** : Select * - יחזיר לי את כל השלשות הקיימות, לדוגמא :
כאשר בהתאמה ה s זה הנושא ה p זה המשווא (היחס) וה o זה המושא.

ניתן גם לשים תנאים על השלשה הנמצאת בסוגרים ולקבל את התוצאות בהתאם.

- **SELECT DISTINCT ?person WHERE {**
 ?person rdf:type dbo:Person .
 ?person tto:pet ?type .
 ?type rdf:type tto:Cat .
}

לדוגמא :

כאשר בדוגמא זו אנו משתמשים בSELECT DISTINCT כדי לקבל את התוצאה המדויקת ביותר אדם שיש לו חיות מחמד מסוג חתול.

כדי לקבל את כל התוצאות בצורת שלילה לדוגמא, שאילתה שתביא לי את כל אלא שאין להם חיות

- **SELECT ?person WHERE {**
 ?person rdf:type dbo:Person .
 FILTER NOT EXISTS {?person tto:pet ?pet } .
}

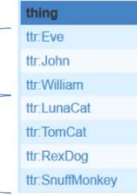
מחמד , נשתמש במילים FILTER NOT EXISTS :

UNION – כאשר נרצה לקבל משהו כולל שנמצא בשתי מחלקות או במחלקה ותת מחלקה נשתמש במילה UNION , לדוגמא, אנו נרצה לקבל את כל בעלי החיות אז קודם ניגש ל type מסוג person ואז לתת המחלקה של החיות וכיוון שבמחלקה הזו ישנה עוד מחלקת חיות (סוג של נכד) ניגש גם אליה ונעשה UNION.

```

SELECT ?thing WHERE
{
    ?thing rdf:type ?type .
    {
        ?type rdfs:subClassOf tto:Creature .
    }
    UNION
    {
        ?type rdfs:subClassOf ?subcreature .
        ?subcreature rdfs:subClassOf tto:Creature .
    }
}

```

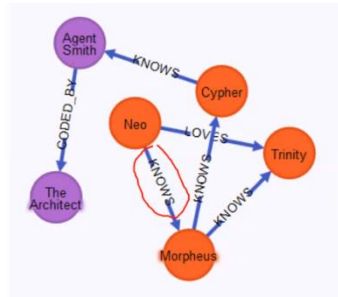


ניתן להכניס את כל הנ"ל בשאילתה אחת עי הסימן + שאומר שתכיל לי את היחס גם ברמה הנוכחית

- `SELECT ?thing WHERE {
?thing rdf:type / rdfs:subClassOf+ tto:Creature .
}`

Neo4J – ה database השני מסוג גרפים – לשפת השאילתות קוראים Cypher, בשונה מהRDF כל צומת בו יכולה להכיל מסמך ממש אשר לה יש יחס לצומת שגם מכיל מסמך.

לדוגמא גרף המכיל צמתים ואת היחסים ביניהם:



(Neo)-[:LOVES]->(Trinity)
(Neo)-[]->(Trinity)
(Neo)-->(Trinity)
(Trinity)--(Neo)

(Architect)-[:CODED_BY]-(Smith)
(Morpheus)-[:KNOWS]-(Trinity)

פקודות :

CREATE – יצירה של צומת - `CREATE (n)`

יצירה של צומת עם תכונות (מקבלת type, properties, reference) –

The node reference ("glass") can only be used during the same query

Here student is a label. Labels act like categories or types.

➤ `CREATE (glass:student {name: 'Chaya Glass', id:111, age:21, degree:'1'})`

➤ `CREATE (:student {name: 'Tal Negev', id:222, age:28, degree:'3'}), (:student {name: 'Gadi Golan', id:333, age:24, degree:'1'})`

יצירת כמה צמתים בבת אחת –

כבר ביצירת הצומת ניתן להוסיף ולייצר את הקשרים (הקשתות) –

➤ `CREATE (glass:student {name: 'Chaya Glass', id:111, age:21, degree:'1'}), (negev:student {name: 'Tal Negev', id:222, age:28, degree:'3'}), (golan:student {name: 'Gadi Golan', id:333, age:24, degree:'1'}), (negev)-[r1:teaches]->(glass), (golan)-[:in_class_with]->(glass), (glass)-[:in_class_with]->(golan)`

All edges are directional. It is redundant to create the inverse relationship e.g.:

MATCH – לאחר יצירת הצמתים ניתן לחבר בינ

➤ `MATCH (a:student),(b:student) WHERE a.name = 'Tal Negev' AND b.name = 'Chaya Glass' CREATE (a)-[r1:teaches]->(b)`

כיוון שבעצם MATCH מחפש את התבנית ורק אז קושר אותה ניתן לעשות באמצעות חיפוש זה

➤ `MATCH (a)->(b{name:'Chaya Glass'}) RETURN a` : דברים מורכבים יותר לדוגמא חיפוש צמתים עם קשר מסוים :

Variable – length pattern – התאמה של תבניות עם אורך משתנה, לדוגמא תחזיר לי מסלול באורך

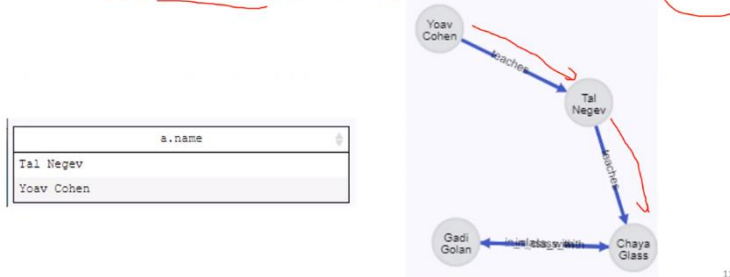
2 מ a ל b - `(a)-[*2]->(b)`

דוגמא נוספת, מסלול שהוא בגודל של מינימום 3 ומקסימום 5 - `(a)-[*3..5]->(b)`

דוגמא נוספת – אם נתון לי הגרף הבא ואני רוצה למצוא את כל השמות של המרצים שמלמדים את חיה גלאס או מורים שמלמדים את המורים של חיה גלאס אז אשתמש ביחס כאורך, כלומר כיוון שיש לי אופציה להיות מורה ישיר של חיה גלאס (אורך של צלע אחת) או להיות המורה של המורה של חיה גלאס (אורך של שתי צלעות) אזי אשתמש ביחס אורך של 1 עד 2 (אבא ונכד).

➤ `MATCH (a)-[:teaches*1..2]->(b:student {name:'Chava Glass'}) RETURN a.name`

כך תיראה השאילתה :



Paths – מציאת מסלול, לדוגמא (מה שמסומן הוא המסלול) השאילתה מבקשת שתחזיר את כל המסלולים שגדי גולן מכיר מדרגה שניה עד רביעית :

➤ `MATCH p=(a {name:'Gadi Golan'})-[:KNOWS*2..4]->(b) RETURN p`

מציאת המסלול הקצר ביותר, לדוגמא –

➤ `MATCH p=shortestPath((s1:student {name:'Gadi Golan'})-[*]-(s2:student {name:'Tal Negev'})) RETURN p`

WITH - סינון, סימון הנתונים לפני שאנו עוברים לשליפה הבא, לדוגמא :

➤ `MATCH (c:course) WITH COLLECT(c) AS courses
MATCH (s:student) WHERE ALL (x IN courses WHERE (s)-[:studies]->(x))
RETURN s.name`

בדוגמא זו אנו קודם עושים התאמה שמוצאת את כל הקורסים, את התוצאה אנו מכניסים לאוסף שיצרנו בעזרת המילה COLLECT (ונתנו לו שם בסוגריים).

כעת בהMATCH השני אנו מבקשים לקיים את התנאי הבא על הסטודנטים, התנאי הוא שהסטודנט לומד את כל הקורסים (כלומר השתמשנו באוסף שעשינו בMATCH הראשון שעזר לנו בתנאי).

שיעור 6 .

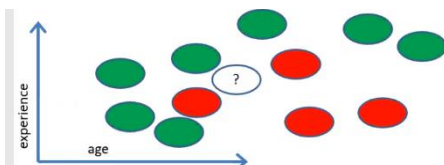
הקדמה ללמידת מכונה :

עד עכשיו התעסקנו עם מידע קיים ורק עבדנו עליו, למידת מכונה מגיעה כדי לענות על מקום בו אין מידע רלוונטי.

נצטרך מהמידע הקיים להוציא מידע רלוונטי – לסווג אותו, לחלק לקטגוריות.

כלומר נרצה לאמן את המחשב שלי להסיק את הערך של אדם חדש שמגיע ע"י מדדים שכבר קיימים בקבוצה של אנשים כאשר בקבוצה זו המידע קיים. לקבוצה הזו קוראים training set.

לדוגמא נתונן הגרף שמייצג את הקבוצה הבאה - בא הירוק מסמן אדם שעובד ואילו האדום מסמן אדם מובטל, ע"י שני פרמטרים – גיל וניסיון.



מדוגמא זו ניתן להסיק שהאדם בסימן שאלה שהצטרף עובד.

: Machine Learning

הגדרה: אומרים על תוכנית מחשב שהיא לומדת מהניסיון אם הביצועים משתפרים עם הניסיון (ניסיון = דאטה נתונה) שגדל.

ההבדל הוא שהמחשב "לומד", הוא לא מתוכנת בצורה שבה הוא מחזיק את כל התרחישים.

בניית machine learning classifier: בניית מודל מסווג כך שנותנים לו מידע חדש והוא יודע לנבא או לחזור את התבנית שלו.

כדי לאמן את המחשב אנו חייבים כנקודת פתיחה מידע מסווג כבר (label data), כעת לאחר שמחשב מסווג את המידע החדש שהגיע נרצה לדעת האם הסיווג נכון.

אזי יהיה חכם לאמן את המודל על 80% מהמידע הנתון לי ולאחר מכן לבדוק אותו על ה-20% הנותרים וכך אני יוכל להעריך האם הסיווג נכון או לא.

אלגוריתמי סיווג:

Naïve Bayes – מבוסס על נוסחת בייס, עובד עם שני מושגים עיקריים – פיצורים (מאפיינים) וקטגוריות, יכול לעבוד גם שחסר מידע.

לדוגמא: סיווג הודעה כהודעת ספאם ע"י בניית מודל מהמידע הקיים ושימוש בtrain and set.

כעת נשתמש בחוק בייס להסתברות מותנה (מה ההסתברות ל Y בהינתן X) - $p(Y|X) = \frac{p(Y)p(X|Y)}{p(X)}$

לדוגמא, מה ההסתברות שלי שההודעה היא ספאם כאשר המילה "you" חלק מהמשפט?

| | Examples | are | you | paying | too | much | ? | click | now | ! |
|------|----------|-----|-----|--------|-----|------|---|-------|-----|---|
| Spam | 2 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 2 |
| Real | 3 | 0 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

כאשר נתונה הטבלה הבאה:

$$p(\text{spam} | \text{with "you"}) = \frac{P(\text{spam}) \times P(\text{with "you"} | \text{spam})}{P(\text{with "you"})} = \frac{\frac{2}{5} \times \frac{1}{2}}{\frac{4}{5}} = \frac{1}{4}$$

נחשב על פי הנוסחה:

וכן סביר להניח שעל פי ההסתברות המשפט לא ספאם.

כעת נרצה לעבור על כל המילים במשפט ולכן נשתמש בנוסחה הבאה: $y^* = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} p(y = k) \prod_{i=1}^n p(x_i | y = k)$

כאשר x_i זה המילה שאנו נמצאים בה.

K זה הקטגוריות שלנו (למשל ספאם או אמיתי).

n זה כמות המילים שבמשפט.

ואז ניקח את המקסימום.

כעת במקרה שיש לנו 0 לא נוכל לדעת איך לסווג אותו לכן הגיע חוק laplace ואומר שנסווג אותו

באמצעות המודל הבא: $p_{i, \alpha\text{-smoothed}} = \frac{x_i + \alpha}{N + \alpha K}$

לרוב האלפא תהיה שווה 1.

לדוגמא:

$$\begin{array}{ll} x_2 = \text{"you"} & p(x_2 | y = \text{real}) = \frac{3}{3} = 1 \\ x_1 = \text{"are"} & p(x_1 | y = \text{real}) = \frac{0}{3} = 0 \end{array} \quad \Rightarrow \quad \begin{array}{ll} x_2 = \text{"you"} & p(x_2 | y = \text{real}) = \frac{4}{5} \\ x_1 = \text{"are"} & p(x_1 | y = \text{real}) = \frac{1}{5} \end{array}$$

כלומר למונה אני מוסיף 1 ולמכנה 2, לאחר הטיפול בכל הערכים נקבל את הטבלה הבאה ללא אפסים:

| | Examples | are | you | paying | too | much | ? | click | now | ! |
|------|----------|----------|----------|--------|-----|------|---|-------|-----|---|
| Spam | 3 | <u>1</u> | 2 | 2 | 1 | 1 | 1 | 3 | 1 | 3 |
| Real | 4 | 1 | <u>4</u> | 1 | 1 | 1 | 2 | 1 | 1 | 1 |

כעת נחשב מה ההסברות שכל אחד הוא ספאם או אמיתי (כל המשפט) והגדול יותר הוא ספאם.

$$p(y = 0|x) = \frac{3}{7} \cdot \frac{1}{4} \frac{2}{4} \frac{2}{4} \frac{1}{4} \frac{1}{4} \frac{1}{4} \frac{3}{4} \frac{1}{4} \frac{3}{4} = 5.87 \cdot 10^{-5} \quad \text{לדוגמא :}$$

$$p(y = 1|x) = \frac{4}{7} \cdot \frac{1}{5} \frac{4}{5} \frac{1}{5} \frac{1}{5} \frac{1}{5} \frac{2}{5} \frac{1}{5} \frac{1}{5} \frac{1}{5} = 2.34 \cdot 10^{-6}$$

ההנחה מבוססת על הנאביות שאין תלות בין המשתנים ולכן אני יכול לרשום אותה בצורה הבאה :

Naïve Bayes assumption $p(x_t, y_t) = p(y_t)p(x_{t1} | y_t)p(x_{t2} | y_t)p(x_{t3} | y_t) \cdot \dots$

וכעת על פי חוק בייס אנחנו יכולים לפתח את הנוסחה שלנו הנמצאת לעיל.

תהליך המימוש :

קודם נבדוק כמה מסמכים יש לנו בסך הכל ונסמן את הנתון הזה כ-Ptot.

לאחר מכן נבדוק כמה מסמכים יש בכל מחלקה ונסמן את זה כ-Pk.

כעת נסווג את כל המילים שנמצאים במסמכים שיש לנו כבר ונספור את כמות הפעמים שהם מופיעים בקטגוריה שלהם, נסמן את זה כ-Pki.

$$y^* = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} \frac{p_k}{p_{tot}} \prod_{i=1}^n \frac{p_{k_i}}{p_k} \quad \text{כעת נקבל את המשפט ונעבור עליו עם הנוסחה שאנו מכירים :}$$

כלומר כיוון שעשינו את כל עבודת ההכנה וסיווגנו את כל המילים שנמצאו בתוך המסמכים (אנחנו מכילים את כל המילים מראש) אזי התהליך הרבה יותר קל ומהיר.

לדוגמא, נרצה לסווג משפט האם הוא ברכת שלום או ברכת להתראות.

נכניס את הפקודה יחס עם מידע ראשוני.

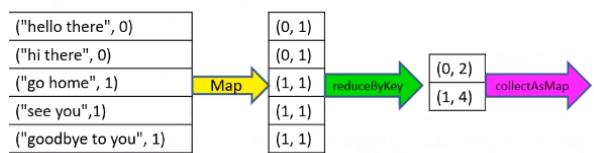
Suppose the data is stored in an RDD as (message, class)
tuples. E.g. (greeting / valediction):

```
input_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go home", 1),  
("see you", 1), ("goodbye to you", 1), ("bye bye", 1)])
```

כעת נמצא את Ptot, Pk, Pki.

```
>>> pk = input_data.map(lambda tup: (tup[1], 1)) \
.reduceByKey(lambda a,b: a+b).collectAsMap()
```

נמצא את pk -



> `ptot = sum(pk.values())`

– נמצא את Ptot – הסכום של כל Pk

```
>>> pki = input_data\
    .flatMap(lambda tup: list(set([(tup[1],w) for w in tup[0].split()]))) \
    .map(lambda tup: (tup, 1)) \
    .reduceByKey(lambda a,b: a+b).collectAsMap()

input_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go
home", 1), ("see you", 1), ("goodbye to you", 1), ("bye bye")])
```

– נמצא את Pki

– כעת נקבל משפט חדש ונסווג אותו

```
>>> import numpy as np

>>> query = "hello hi"

>>> class_probs = [pk[k]/float(ptot)*np.prod(np.array([pki.get((k,i),0)/float(pk[k])
for i in query.split()] for k in range(0,2))]
```



– בעזרת הנוסחאות הבאות נוכל לדעת האם מודל הסיווג שלנו נכון

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Number Of Examples}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

- $2 \cdot (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

– ממוצע הרמוני בין שני המדדים הנ"ל

שיעור 7

Java Stream

נותן לנו את האופציה להפוך אובייקט לרצף של ביטים וכך לעשות פעולות בצורה יותר מהירה.

יש לנו כמה פונקציות:

– Filter – מקבלת אוסף ועוברת עליהם עם תנאי ומסננת לפיו (מה שבסוגריים זה פונקציית הלמדה).

לדוגמא, החזרת האברים הקטנים מ-10: `Arrays.stream(arr).filter(s -> (s < 10))`

– Map – מקבל את הנתונים ועובר איבר אחר איבר ומשנה את האיבר באוסף שלי.

– Sorted – למיין, לדוגמא החזרת האברים בסדר עולה: `Arrays.stream(arr).sorted()`

– Match – מחזיר ערך בוליאני - ברגע שאני מקבל true סיימתי את המעבר על האוסף שלנו (anyMatch), all match, יעבור על כל האוסף והכל חייב להתקיים.

– Reduce – מבצע פעולה על כל הרשימה ומחזיר ערך בודד.

לדוגמא, נקבל 3 אברים נחבר בין השניים הראשונים ונשמור אותם בא ולאחר מכן נחבר עם Y:

הפלט יהיה 52: `Stream.of(1,45, 6).reduce((x,y) -> x+y);`

דוגמא נוספת, כאן לא נרצה לקחת את השניים הראשונים ב stream אלא שא יקבל את הערך 0,

בחלק השני הוא חישוב סופי לאחר החלק הראשון: `Stream.of(1,45, 6).reduce(0, (x,y) -> x+1, (x,y)-> x+y)`

– Collect – הופך את האלמנטים לסוג אחר של נתונים.

```
Stream.of(1, 2, 45, 78, 3, 48, 23, 105, 5, 15)
    .collect(Collectors.groupingBy(x->x%10));
```

לדוגמא , הופך את האובייקט הזה לאובייקט אחר שמחולק במודולו 10 :

דוגמא נוספת, כל פעולה שנעשה באמצעות summarizingInt על X תעשה על כל הX:

```
Stream.of(1, 2, 45, 78, 3, 48, 23, 105, 5, 15)
    .collect(Collectors.summarizingInt((x->x)));
```

זה יהיה הפלט:

```
IntSummaryStatistics{count=10, sum=325,
min=1, average=32.500000, max=105}
```

ב join נוכל לשרשר את המילים.

Parallel stream – כאשר נרצה להשתמש ב threads .

: Spark

נועד לנצל את המשאבים של המחשב בצורה הכי יעילה (open source).

עובד עם אובייקטים RDD , נשתמש בזה כאשר אנו רוצים לעבוד עם ביג דאטה.

פונקציות :

flatMap – שנתונה לי מטריצה ואני רוצה להתייחס אליה כאל מערך רציף אזי אני אשטח את המטריצה.

לדוגמא, קבלת מערך, הפיכתו לאובייקט RDD והכנסת של ערך ע"י map ושימוש ב reduce כדי לקחת הערך ומחבר אותם ובסוף מחזיר לי אותו כאובייקט באמצעות collect :

```
>> text_file = sc.textFile("myDir/story.txt")
>> word_counts = text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a,b: a+b) \
    .collect()
>> for word,count in word_counts:
    print("the word: \"%s\" appears %d time(s)" %(word,count))
```

Result:

```
...
the word: "retrograde" appears 1 time(s)
the word: "grounds" appears 4 time(s)
the word: "VOUS" appears 2 time(s)
the word: "Flatterers" appears 1 time(s)
the word: "injustice," appears 1 time(s)
the word: "reciprocity," appears 1 time(s)
the word: "inflicted" appears 2 time(s)
the word: "limbs." appears 1 time(s)
the word: "christened" appears 2 time(s)
the word: "majority--where" appears 1 time(s)
the word: "three-fourths" appears 1 time(s)
the word: "dish," appears 1 time(s)
the word: "73." appears 1 time(s)
the word: "ENVIRONMENT," appears 1 time(s)
the word: "honesty," appears 1 time(s)
```

SortByKey – ימין לי את הערכים מהגדול לקטן (מהקטן לגדול – נרשום false)

Sort by Value – נצטרך למיין אותם לפי סדר ההופעה שלהם ולכן נצטרך לשנות את הערכים שלהם, כלומר נשתמש בפונקציה map (וכיוון שזה אובייקט RDD זה טופל שבפיתון לא ניתן לשנות אותו לכן אנו עושים טופל חדש ב-map, ובמקום (x,y) נרשום (y,x).

לאחר מכן נמיין אותו בסדר יורד (sort by key (false)).

שוב נשנה את ה-map כמו שעשינו לעיל.

לדוגמא :

```
word_counts = (
    text_file.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a,b: a+b)
    .map(lambda (x,y): (y,x))
    #False is for descending order
    .sortByKey(False)
    .map(lambda (x,y): (y,x))
    .collect()
)
```

Bi-grams – מחלק את הטקסט שלנו לזוגות עוקבים.

נממש באמצעות הפונקציה zip, לדוגמא נניח נתונים שני מערכים ואותם נכניס ל zip אזי נעבור מערך מערך ובהתאמה עושה ממנו זוג ושומר אותם כרשימה:

```
a=[1,2,3]
b=['a','b','c']
```

```
zipped = zip(a,b)
print(list(zipped))
```

Tri-grams – מפצל את הטקסט לשלוש עוקבות .

שיעור 8:

Linear Regression :

בא לחזות איזה ערך מסוים.

איך המודל עובד ?

אנו עובדים על הדאטה בגרף של ציר X וציר Y, כלומר הדאטה תהיה מיוצגת ע"י x וy כאשר X הוא ווקטור, ומה שאנו נרצה לעשות זה להעביר קו ישר כמה שיותר קרוב לכל הנקודות בגרף.

כלומר המטרה היא לבנות משוואת קו ישר : $Y = wx + b$ כאשר W היא המשקל על כל פיצ'ר (כלומר כמה הוא משפיע על הנתון X).

נפרק את הפונקציה הזו - $b + wx_1 + wx_2 + \dots + wx_n$ כאשר זה ירכיב את כל הקו הישר.

לפונקציה הזו נקרא $h(x)$ והיא תיקרא פונקציה החיזוי שלנו.

איך נגלה את ה-w וה-b ?

אנחנו נשתמש באלגוריתם ש"יאמן" את w1 עד wn וגם את b וכך הוא מוצא אותם.

(ב-naive base חישבנו הסתברויות וכאן אנו נאמן ע"י מציאת הפיצ'רים (n+1)).

כדי למצוא את w נרצה למצוא את ממוצע המרחקים (VAR) כך שהוא יהיה מינימלי,

• Loss function: $J(w, b) = \frac{1}{m} \sum_{i=1}^m |wx_i + b - y_i|$: נשתמש בנוסחה הבאה :

כאשר m זה כמות הנתונים ב-database, ואנו עוברים ומחשבים את מרחקים לכל נתון ממשוואת הקור הישר.

אבל נוסחה זו קשה לגזירה ולכן נשתמש בנוסחה הבאה : $J(w, b) = \frac{1}{2m} \sum_{i=1}^m (wx_i + b - y_i)^2$

כעת בצורה האידיאלית היינו גוזרים את הפונקציה ומוצאים נק' קיצון אך לתכנת גזירה זה מורכב ולכן נשתמש במושג Gradient Descent .

: Gradient Descent

Gradient - נגזרת של ווקטור (שיפוע של הווקטור) כאשר הוא תמיד הולך לטופ וכאשר נלך נגד כיוון הגרדיאנט יש מקום שהנגזרת מתאפסת וזו תהיה נקודת הקיצון שלנו – וזו אחת השיטות למצוא את נקודות הקיצון – ללכת בכיוון הנגדי.

תזכורת לנגזרות וקטורים : נגזור כל פעם לפי הפרמטר שלו ונחלק בכמות הפרמטרים.

$$\begin{aligned} f_1(j_1, j_2) &= 2j_1 \cdot j_2 + 7j_1 \\ \nabla(f_1(j_1, j_2)) &= (2j_2 + 7, 2j_1) \\ f_2(j_1, j_2, j_3) &= 3j_1^2 j_2 j_3^3 + 5j_1 j_2 \\ \nabla(f_2) &= (6j_1 j_2 j_3^3 + 5j_2, 3j_1^2 j_3^3 + 5j_1, 9j_1^2 j_2 j_3^2) \end{aligned}$$

אז כעת נרצה לגזור את loss function שלנו שהיא וקטור בגודל 2 כדי ללכת בכיוון הנגדי של הגרדיאנט (נגזור לפי w ואז לפי b) :

Our loss function: $J(w, b) = \frac{1}{2m} \sum_{i=1}^m (wx_i + b - y_i)^2$

$$\begin{aligned} \frac{\partial J}{\partial w} &= \frac{1}{2m} \cdot 2 \sum_{i=1}^m ((wx_i + b - y_i)x_i) \\ &= \frac{1}{m} \sum_{i=1}^m (wx_i + b - y_i)x_i \end{aligned}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (wx_i + b - y_i)$$

כעת נרצה לקחת את הנגזרות (השיפועים) וללכת בכיוון ונשתמש באלפא (a=0.01) שאותה נכפיל באותם נגזרות ו אנחנו מתכנסים לנקודה מסוימת :

- Update w to $w - \alpha \frac{1}{m} \sum_{i=0}^m x_i (h(x_i) - y_i)$
- Update b to $b - \alpha \frac{1}{m} \sum_{i=0}^m 1 \cdot (h(x_i) - y_i)$

: האלגוריתם

```
import numpy as np
galaxy_data = np.array([[2,70],[3,110],[4,165],[6,390],[7,550]])
w = 0
b = 0
alpha = 0.01
for iteration in range(10000):
    gradient_b = np.mean(1*(galaxy_data[:,1]-(w*galaxy_data[:,0]+b)))
    gradient_w = np.mean(galaxy_data[:,0]*(galaxy_data[:,1]-(w*galaxy_data[:,0]+b)))
    b += alpha*gradient_b
    w = w + alpha*gradient_w
    if iteration % 200 == 0 :
        print("it:%d, grad_w:%.3f, grad_b:%.3f, w:%.3f, b:%.3f" % (iteration, gradient_w, gradient_b, w, b))
print("Estimated price for Galaxy S5: ", w*5 + b)
```

: Logistic Regression

נועד להבדיל לנו בין classes והוא עונה לנו בכן ולא.

נשתמש בפונקציית החיזוי הבאה כאשר היא בין 0 ל 1 :

$$h(x) = \frac{1}{1 + e^{-(wx+b)}}$$

כאשר גם פה נרצה לאמן את ה w ו b.

וזו תהיה הפונקציה :

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^m (y_i \log(h(x_i))) + (1 - y_i) \log(1 - h(x_i))$$

וכך היא לאחר הגזירה ובה נשתמש :

$$+ \frac{1}{m} \sum_{i=0}^n x_i (h(x_i) - y_i)$$

$$\cdot \text{ Update } w \text{ to } w - \alpha \frac{1}{m} \sum_{i=0}^m x_i (h(x_i) - y_i)$$

נקדם את w ו b :

$$\cdot \text{ Update } b \text{ to } b - \alpha \frac{1}{m} \sum_{i=0}^m 1 \cdot (h(x_i) - y_i)$$

וכך אנו מקבלים מספרים בין 0 ל 1 וכל אנחנו יכולים לבא עם משהו קורה או לא.