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 שהדatabase חייב לעמוד בהם :

Atomicity – מתבצע בשלמות או לא מתבצע בכלל.

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

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

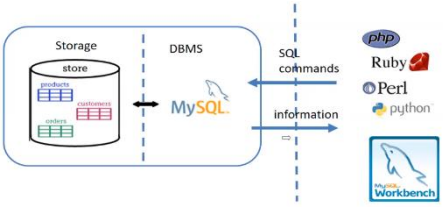
Durability – עמידות – כל בקשה שנשלח לdatabase חייבת להתבצע ,כלומר גם במקרה של נפילה ה database חייב להבטיח שהוא ידע לשחזר את הפעולה ובנוסף גם להחזיק שירותי גיבוי.

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 מdatabase (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 , <,=<,=,>,=>,<> ,AND,OR,NOT,IN .

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 בNOT 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
False	True
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 הlastName (כאשר המיון הדיפולטיבי הוא מהקטן לגדול).

כדי להפוך את הסדר ולמיין 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 Funcion – 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 ECECUTION 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.

בשיטה הנאיבית נעשה -

- SELECT * FROM students, grades

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 סטודנט חדש שאין לו

- SELECT * FROM students LEFT JOIN grades ON 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 אישיויות.

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 –

PRIMARY KEY and UNIQUE can appear right after type

- 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));

INDEX (or KEY) must be defined after a comma

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

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

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 ו grade לg ,נתתי כינויים שמות חדשים לטבלאות שלי.

ניתן גם לתת שם לשליפה שלמה כלומר אני יכול לשלוף 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 הdatabase שומר בצד את הפעולות וברגע שיש נפילה הוא הולך ל commit האחרון וממשיך ממנו או מוחק בצורה הפוכה.

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

```
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 שאילתות 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 אצטרך 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 numOfColumns = rs.getMetaData().getColumnCount();
        while (rs.next()){
          for (int col = 1; col <= numOfColumns; 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 ונייצג 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 ->B כלומר B תלוי בA .

לדוגמא, אם ניתן לך את הת.ז של מישהו 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.

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.

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

Normal Forms – שישה חוקי נירמול :

כל חוק צריך לקיים את החוק הקודם ומוסיף עליו.

1. **1NF-** כל תכונה (עמודה) צריכה להחזיק ערך אטומי יחיד, בנוסף אסור ערכים מחושבים למשל עמודת גיל ועמודת תאריך לידה – וכך נוצר כפילויות.

2. **2NF-**תכונות מסוג Non prime לא תלויות 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.

4. **3.5NF**- BCNF – משלים את חוקים 2 ו3 , לכל 2 קבוצות אם קיימת תלות בינהם אזי בהכרח אחת הקבוצות היא super – key. כלומר, האם קיימת כאן קבוצה שהיא תלויה בקבוצה אחרת והיא לא super -key אם כן אזי זה מקיים BCNF (אם אין כלל תלויות זה גם יעמוד בBCNF).

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 -> C1 אזי זו תלות רב ערכית, כלומר ישנם 2 מקומות שונים שיכולים להביא אותו לC1.

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

הנה דוגמא בעיתית :

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

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

: XML and JSON

פורמטים להעברת מידע מהdatabase .

התקן המפורסם ביותר נקרא **XML** – שפת סימון הניתנת להרחבה ,פורמט להעברת נתונים, היררכית ורגישה לאותיות גדולות/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 geyElement.

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();
      studentList.add(student);
      System.out.println("Degree : " + element.getAttribute("degree")); //just print degree ("PhD" when studentIdx=0)
      NodeList studentAllNodes = studentNode.getChildNodes();
      for (int stIdx = 0; stIdx < studentAllNodes.getLength(); stIdx++){
         Node stInnerNode = studentAllNodes.item(stIdx);
         switch (stInnerNode.getNodeName()){
            case "FirstName": student.firstName = stInnerNode.getTextContent(); break;
            case "LastName": student.lastName = stInnerNode.getTextContent(); break;
            case "id": student.id = Integer.parseInt(stInnerNode.getTextContent()); break;
            case "age": student.age = Integer.parseInt(stInnerNode.getTextContent()); break;
            case "Address":
               Address address = new Address();
               student.address = address;
               NodeList addressAllNodes = stInnerNode.getChildNodes();
               for (int adIdx = 0; adIdx < addressAllNodes.getLength(); adIdx++) {
                  Node adInnerNode = addressAllNodes.item(adIdx);
                  switch (adInnerNode.getNodeName()) {
                     case "Street": address.street = adInnerNode.getTextContent(); break;
                     case "City": address.city = adInnerNode.getTextContent(); break;
```

XPATH – 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 XPATH.

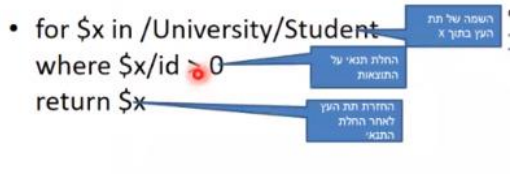
XPATH in Java – מאוד דומה לעבודה של XML עם java אך כאן אני אכין את הבקשה ואז אמיר אותה ל string .

```
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.

עובדים עם המילים השמורות – for, where,let, return לדוגמא :



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 הXML 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 בדרך כלל נתקן את הXML שיתאים לXSD.

שיעור 4:

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

בנוי מ } בשונה מXML שבנוי מ <.

לדוגמא :

```
{"University": {
  "Student": [
   {
    "FirstName": "Chaya",
    "LastName": "Glass",
    "Address": {
     "Street": "Hatamr 5",
     "City": "Ariel",
     "Zip": "40792"
    },
    "age": 21
   },
   {
    "FirstName": "Tom",
    "LastName": "Glow",
    "Address": {
     "Street": "Mishmar 5",
     "City": "Ariel"
    }
   }
  ]
 }
}
```

Array of "student"

Integer (no quotes)

Two additional types are booleans (true, false) and null.

JSON in Java -בצורה דומה לעבודה של XML כך גם ב JSON יש ספריות ייעודית, 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 ,לשלוף ולהסיק מסקנות ביעילות.

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

– BASE

- **B**asically **A**vailable: data is mostly available.
- soft **S**tate: state may change even with no updates (since older updates are still propagating).
- **E**ventual 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.

<u>CAP theorem</u> - 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 הNoSQL:

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

כל התשאול מתבצע באמצעות הkey (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 מבוזר שיושב על כמה שרתים שעליהם הdatabase 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.

Keys and column – לכל מפתח יש עמודות -

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

פקודות :

CREATE KEYSPACE – יצירת database, לדוגמא :

Replication refers to how the data is replicated across different nodes

JSON style

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

USE - להשתמש במה שיצרתי לעיל.

CREATE TABLE – יצירת טבלאות, לדוגמא :

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

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

הערכים שהכנסנו יהיו העמודות.

INSERT INTO– הכנסת ערכים, לדוגמא :

Exactly like SQL...

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

Must use single quotes!

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

WHERE – שליפה לפי מפתח :

➢ SELECT * from students WHERE id=111;

WHERE on a key is ok

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

Cassandra storage method- ישנו data center שבו מאוחסנים השרתים שאיתם עובד הdatabase, 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.

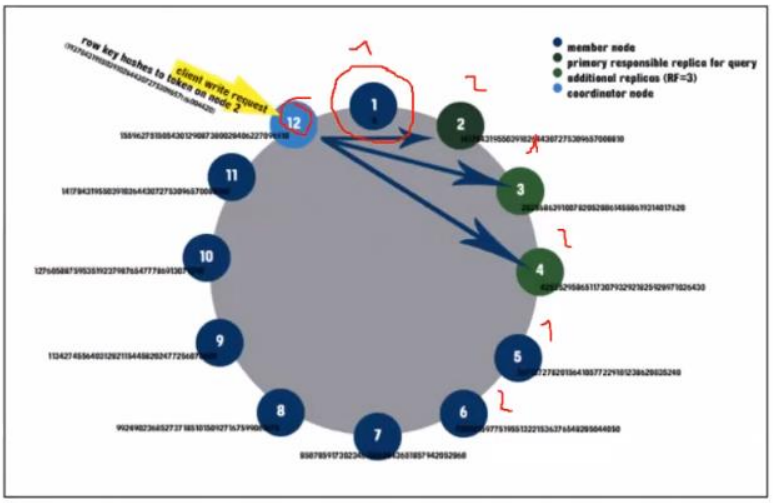


Figure 1 A 12 node cluster using RandomPartitioner and a keyspace with Replication Factor (RF) = 3, demonstrating a client making a write request at a coordinator node and showing the replicas (2, 3, 4) for the query's row key

https://www.bakkalabs.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 — retrieving the node and the partition

clustering keys — Sorting withing the partition

כאשר ה gym_name הוא ה partition key ואילו מצד ימין אנו מוספים מאפיינים שנועדו לסדר את הנתונים בתוך המחשב הפנימי והם הclustering 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, 'into 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 databaseראלציוני.

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

הזמינות בcassandra גבוהה יותר.

בcassandra הdata model דינאמי.

ב cassandra 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 הkey 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– יצירת ה database לדוגמא :

```
➢use University
```

() 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"}
})
```

שאילתות :

Find – יחזיר הכל ,לדוגמא :

```
➢db.students.find()
{ "_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" } }
```

Find().pretty() – מסדר את הJSOM יפה.

```
➢db.students.find().pretty()
```

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 :

בשאילתה זו אנו 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 בלעדיהם):

```
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.

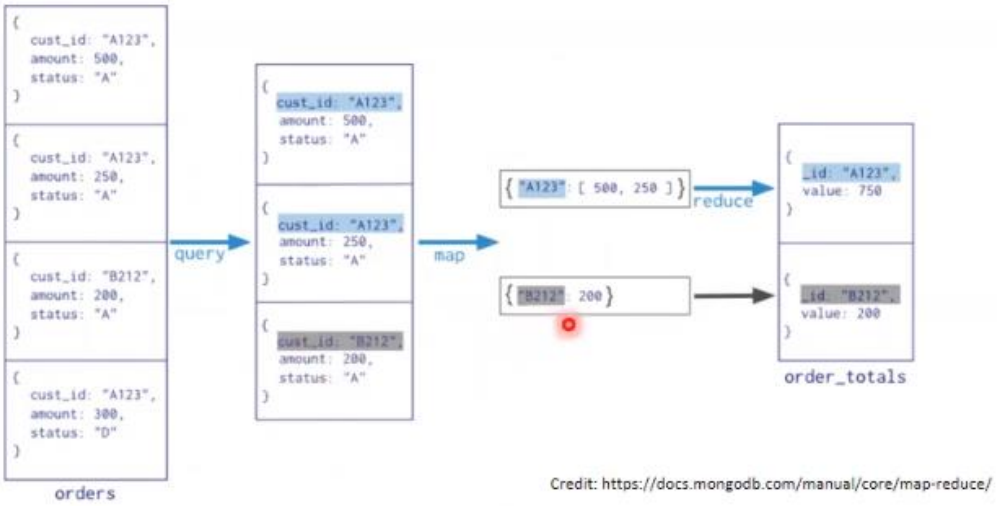
אנו רוצים לקבל את הסכום ששולם עבור כל לקוח שנמצא בסטטוס 'A'.

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 האפשריות ומכין את המap.



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

כדי לקבל את התוצאה נעשה find על הorder_totals –

```
➢db.order_totals.find()
{ _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.

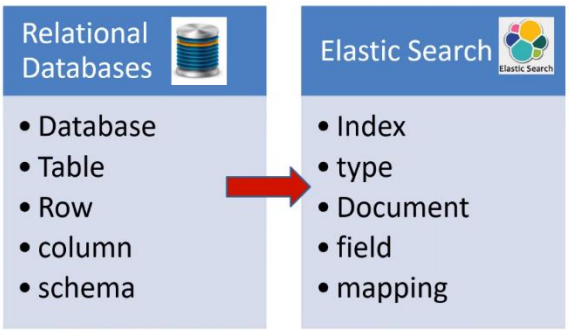
Search Engine Database – databases מסוגי מנועי חיפוש.

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

Elastric Search – database שהוא real -time כלומר, לוקח בערך שניה מהרגע שהעלתי מסך עד שהוא יופיע בתוצאות החיפוש.

נותן פתרונות לחברות שמנועי חיפוש זה לא המוצר העיקרי שלהם.

סינטקס -



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.

XHEAD– בודק האם המסך קיים או לא :

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

XDELETE – מחיקת המסמך :

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

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/lA9AInsBL04IeaKD9LL9/_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 הdatabase.

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 אפשריות סינון יותר רחבות :

Match – מחזיר את המסמכים שמתאמים 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":{"First
Name":"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","LastNam
e":"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\" } },
   \"filter\" :
   { \"range\" :
    { \"age\" :
     { \"lt\" : 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},"hit
s":{"total":1,"max_score":0.0,"hits":[{"_
index":"university","_type":"students",
"_id":"111","_score":0.0,"_source":{"Fi
rstName": "Chaya", "LastName":
"Glass", "
age": "21", "Address": { "Street":
"Hatamr 5", "City": "Ariel","Zip":
"40792"}}}
]}}
```

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 בשאילה ומדרג אותה.

הוא מחולק ל2:

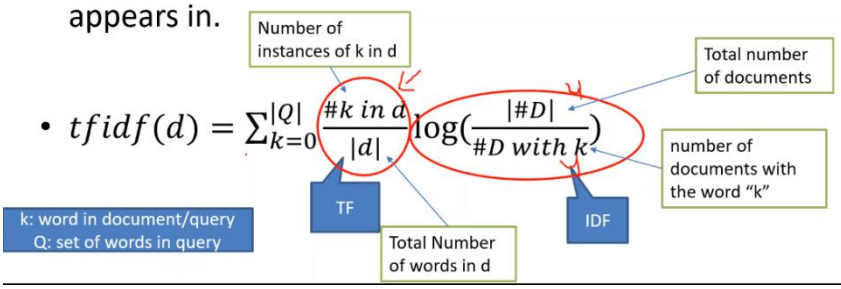
TF – ספריה של מספר המופעים של מילה במסמך כאשר הוא מנורמל למספר המילים שיש בתוך המסמך, כלומר אם יש יותר מופעים של המילה במסמך קצר יותר אזי המילה חשובה יותר.

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 :



אז למעשה ניקח את השאילה וניצור טבלה כאשר כל עמודה מייצג 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 :

Q	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 :

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

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 שאילתות שונים.

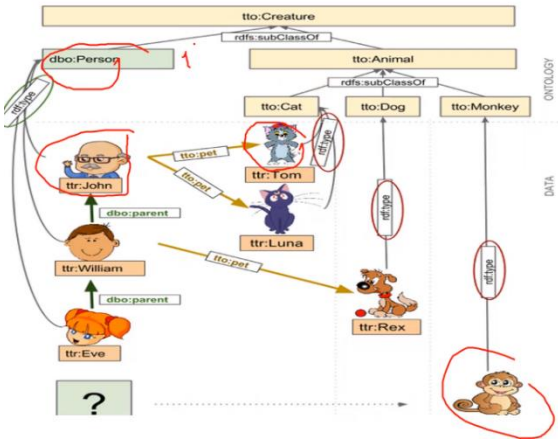
RDF- Resource Description Framework – מודל סטנדרטי, אך הייחוד בו שהכל מורכב משלשות – נושא, משוא, ומושא (Subject,predicate,Object).

ובעצם שלשות כאלה מאוחסנות בטבלה.

Jena- database מבוסס גרפים שבו אנו נתרכז בשפת השאילתות 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 :



פקודות :

* Select - יחזיר לי את כל השלשות הקיימות, לדוגמא :

```
• SELECT * WHERE {?s ?p ?o}
```

כאשר בהתאמה ה s זה הנושא ה p זה המשוא(היחס) וה o זה המושא.

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

לדוגמא :

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

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

כדי לקבל את כל התוצאות בצורת שלילה לדוגמא, שאילתה שתביא לי את כל אלא שאין להם חיית מחמד , נשתמש במילים FILTER NOT EXISTS :

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

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 .
        }
}
```

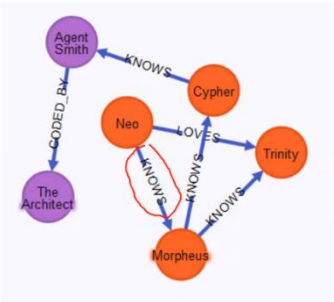
thing
ttr:Eve
ttr:John
ttr:William
ttr:LunaCat
ttr:TomCat
ttr:RexDog
ttr:SnuffMonkey

ניתן להכניס את כל הנ"ל בשאילתה אחת עי הסימן + שאומר שתכיל לי את היחס גם ברמה הנוכחית וגם ברמה מעליה (חוסך את החלוקה לבלוקים).

```
• 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 reference ,type, properties) –

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'})

Properties

יצירת כמה צמתים בבת אחת –

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

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.: (glass)-[:taught_by]->(negev).

MATCH– לאחר יצירת הצמתים ניתן לחבר ביניהם קשת ע"י הפקודה MATCH, לדוגמא :

```
➢ 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 דברים מורכבים יותר לדוגמא חיפוש צמתים עם קשר מסוים :

```
➢ MATCH (a)-->(b{name:'Chaya Glass'}) RETURN a
```

Variable – length pattern -התאמה של תבניות עם אורך משתנה, לדוגמא תחזיר לי מסלול באורך 2 מ a לb -

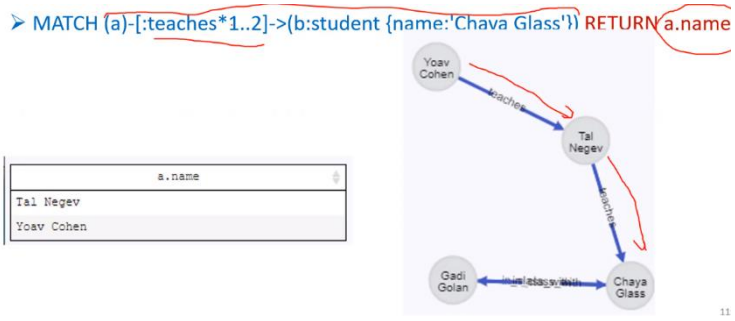
- (a)-[*2]->(b)

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).

כך תיראה השאילתה :

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



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 בסוגרים).

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 .