

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.

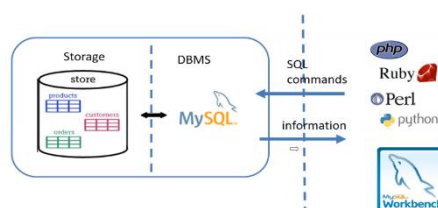
Durability – עמידות – כל בקשה שנשלח לdatabase חייבת להתבצע, כלומר גם במקרה של נפילה ה database חייב להבטיח שהוא ידע לשחזר את הפעולה ובנוסף גם להחזיק שירותי גיבוי.

2. ליניארי.

SQL :

מבנה DBMS – 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 מ-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.

שאלות 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.

INSERT INTO courses : לדוגמא

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

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

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

SELECT id,firstName FROM students ORDER BY : לדוגמא
lastName

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

כדי להפוך את הסדר ולמיון המגדול לקטן נשתמש במילה DESC.

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

SELECT courseId, AVG(grade) FROM grades : לדוגמא
GROUP BY courseId

בשאלתה 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 ב-databases 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** **students.id = grades.studentId** **STUDENT חדש שאין לו**

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 ו grade ל g, 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 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 ונייצג אותו בעזרת טבלאות ?

נסן 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) $A \rightarrow B$ כך ש A כלומר B תלוי A .

לדוגמא, אם ניתן לך את הת. של משהו 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 נרמול :

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 ו 3, לכל 2 קבוצות אם קיימת תלות ביניהם 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 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.

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.

: 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 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().getTagName()); //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();
        student.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 – לדוגמה 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 – לדוגמה :

• for \$x in /University/Student
where \$x/id > 0
return \$x

השמה של תת
העץ בתוך x
החלת תנאי על
התוצאות
החזרת תת העץ
לאחר החלת
התנאי

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 בדרך 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:

```
{
  "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 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 בJSON 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) שה SQL עמד בהם, אבל הוא כן תומך BASE.

– BASE

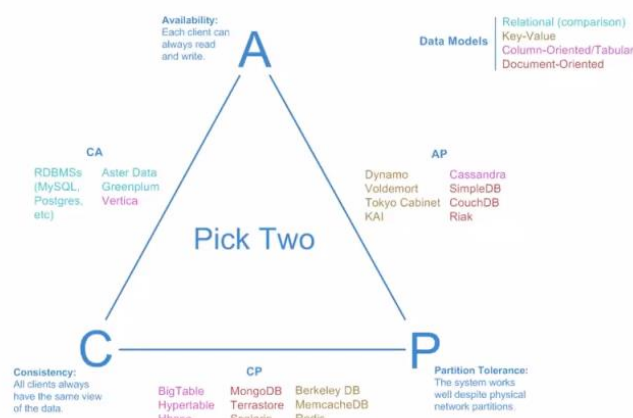
- **Basically Available:** data is mostly available.
- **soft State:** state may change even with no updates (since older updates are still propagating).
- **Eventual consistency:** if we let the data propagate enough time, it will become consistent.

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 ב SQL.
- 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 הNoSQL:

Key Value Store – לכל פריט יש key וvalue . אחסון מהיר, קל לשימוש, גמיש.

כל התשאול מתבצע באמצעות key (מהיר יותר).

פקודות בסיסיות – set, get, del .

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

פקודות על רשימה (LIST) – RPOP – דחיפה מימין (מהסוף), LPOP (מהתאמה), LRANGE – תציג לי את כל הרשימה.

פקודות על טבלאות גיבוב (Hashes), HSET – 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 מבוסס שיושב על כמה שרתים שעליהם databases נמצא, כיוון שה database יושב כל כמה שרתים אזי אין 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 שיצרתי לעיל.

➤ **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.

CREATE TABLE – יצירת טבלאות, לדוגמא :

הערכים שהכנסנו יהיו העמודות.

Exactly like SQL...

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

Must use single quotes!

INSERT INTO – הכנסת ערכים, לדוגמא :

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

➤ **SELECT *** from students WHERE id=111;

WHERE on a key is ok

WHERE – שליפה לפי מפתח :

אם ננסה לשלוח 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.

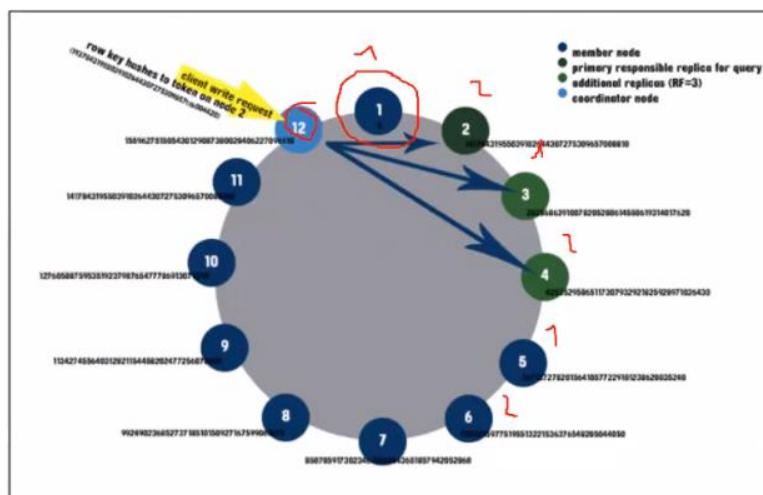
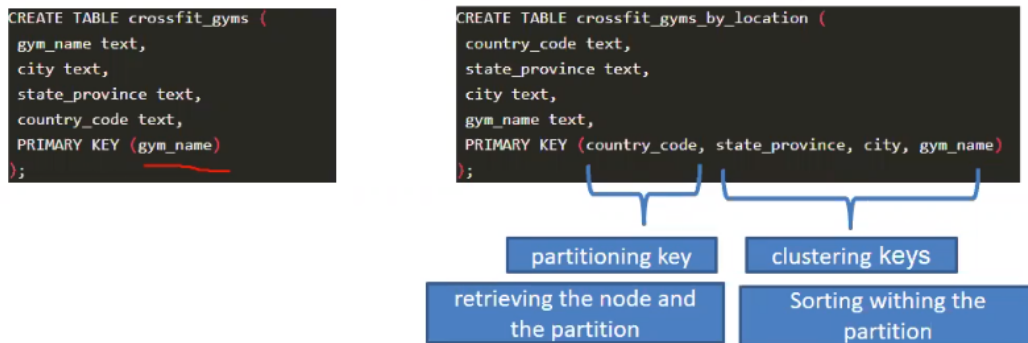


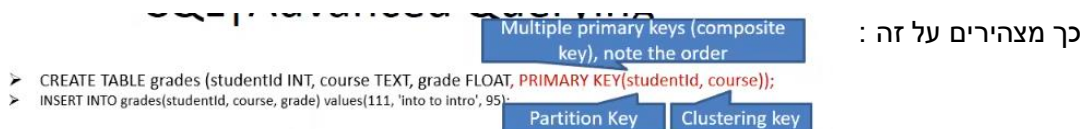
Figure 1 A 12 node cluster using RandomPartitioner and a keyspace with Replication Factor (RF) = 3, demonstrating a client making a write request at a coordinator node and showing the replicas (2, 3, 4) for the query's row key

<https://www.hakhalabs.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 :



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.



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 ראלציוני.

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

הזמינות בcassandra גבוהה יותר.

באשר לcassandra ה data model דינאמי.

באשר לcassandra 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 key שלו הוא מסמך מורכב.

כאשר בתוך value יכול להיות עוד מסמך שלם.

המסמכים נכתבים בפורמט של JSON כאשר כאן השאליות יודעות לתשאל את הערך את value ולא רק את key.

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 ה JSON 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"}]})
```

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

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

לדוגמא :

בשאלתה 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 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 :

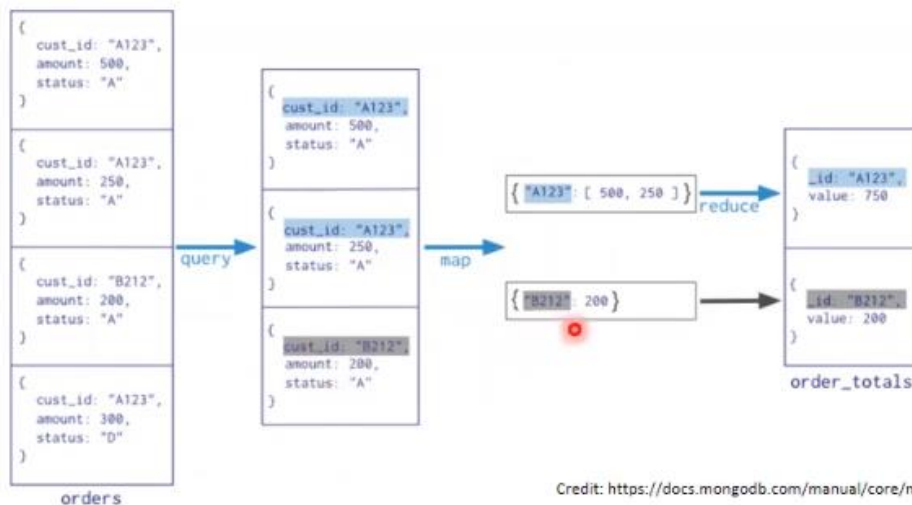
```

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 → "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 המאפ.



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 }

```

