מסדי נתונים

מבוסס על מצגותיו של ד"ר עמוס עזריה

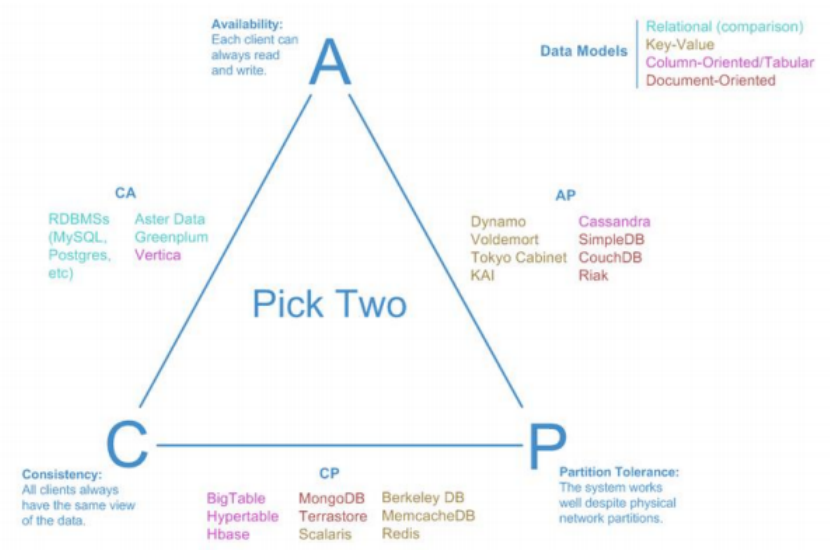
1 באוגוסט 2019

סיכום זה מבוסס על המצגות מהשיעורים. יש לקחת בערבון מוגבל

לתיקונים/הערות/הארות - 0508752542

בהצלחה לכולנו!!

נעם דומוביץ



תוכן עניינים

1 SQL . . . 3
1.1 $STORED\ PROCEDURES$ . . . 7
1.2 $Triggers$ . . . 8
2 אלגברה רלציוניות . . . 9
3 $Normalization$ . . . 9
3.1 $Dependencies$: . . . 9
3.2 $KEYS$ . . . 10
3.3 NF . . . 10
3.3.1 $1, 2, 3, 3.5, 4$ . . . 10
4 $XML - JSON$ . . . 13
4.1 $XML(eXtensible - Markup - Language)$ . . . 13
4.2 $XPath$ . . . 13
4.3 $Xquery$ . . . 14
4.3.1 $FLWOR$ . . . 14
4.4 $Validation$ . . . 15
4.4.1 DTD . . . 15
4.4.2 $XML - Schema$ . . . 17
4.5 $JSON$ . . . 19
5 $NoSQL$ . . . 20
5.0.1 $cap\ theorm$ . . . 20
5.1 ***redis*** . . . 21
5.2 ***cassandra*** . . . 23
5.3 $MongoDB$ . . . 27
5.4 ***elastic search*** . . . 30
5.4.1 $Tf - idf$ . . . 32
5.5 $ARQ/SPARQL$ . . . 34
5.6 $Neo4J$ . . . 36
6 $Java\ Stream$ . . . 40
6.1 $Stream$ . . . 40
6.2 $Parallel - Execution$ . . . 41
7 $Numpy$ . . . 41
8 $Spark$ . . . 42
9 Naïve Bayes . . . 45
9.0.1 מדד סיווג (פונ' $F - score$) . . . 47
10 $Linear\ Regression$ . . . 48
11 $Logistic\ Regression$ . . . 49

מסדי נתונים רלציונים

כל מסד נתונים רלציוני, פועל לפי 4 עקרונות:

- $Atomicity$ - כל פעולה ("שאילתא") צריך להתבצע במלואה או לא להתבצע בכלל.
  - הערה: עקרון זה עובד גם על **פעולות מורכבות**, ולא רק על פעולות אטומיות.
- $Consistency$ - עקביות - ביצוע פעולה לא פוגע בחוקי המסד נתונים.
- $Isolation$ - פעולות שונות יכולות להתבצע במקביל (או סדר שונה), והתוצאה תהיה זהה.
- $Durablitiy$ - המשכיות - ישנה המשכיות לפעולות שהופרעו באמצע עבודתן (דוגמה לפתרון, גיבוי).

1 SQL

$SELECT$ - שליפה מטבלה

- שליפה מטבלה: $SELECT\ \boldsymbol{col}\ FROM\ \boldsymbol{table}$ למשל: $SELECT\ id\ FROM\ studnts$
  - $SELECT\ *$ - מחזיר את כל העמודות
  - $SELECT\ DISTINCT$ - מוריד כפילויות
- הוספת תנאים - $WHERE$
  - שליפה עם תנאי: $SELECT\ \boldsymbol{col}\ FROM\ \boldsymbol{table}\ WHERE\ \boldsymbol{condition}$
  - למשל: $SELECT\ id\ FROM\ studnts\ WHERE\ degree < 2$
- סוגי תנאים:
  - $<, >, =, <=, ...$
  - AND, OR, NOT
  - $BETWEEN\ x\ and\ y$
  - $IN(x, y, z)$ למשל: $SELECT\ id\ FROM\ studnts\ WHERE\ id\ IN(111, 444)$
  - $LIKE(C\%)$ למשל: $SELECT\ id\ FROM\ studnts\ WHERE\ id\ LIKE(mo\%)$
- $Nested\ Queries$:
  - למשל:
  - SELECT lecturer FROM courses WHERE id NOT IN (SELECT courseId FROM grades)
  - איחוד טבלאות: UNION , למשל:
  - (SELECT....)UNION (SELECT)
  - הערות:
    - צריך שמספר העמודות יהיה זהה
    - צריך שסוג העמודות יהיה זהה
  - לא קיים ב $MySQL$ אבל יש גם INTERSECT,EXCEPT , אך ניתן לממש בדרכים אחרות
- $NULL$
  - תא ריק בטבלה.
  - $COALESCE(val1, val2, ...)$ - אם $va1 = Null$ הוא יביא את $val2$ אם $val2 = Null$ ....
  - ניתן כך להוסיף ערך שירשם במקום $Null$

$INSERT$ - **הכנסה לטבלה**

- הוספת רשומה (שורה) לטבלה: $INSERET\ INTO\ \boldsymbol{table}\ (\boldsymbol{co1}, \boldsymbol{co2}, \boldsymbol{...})\ VALUES\ (\boldsymbol{val1}, \boldsymbol{val2}, \boldsymbol{...})$

$OEDER\ BY$

- למשל: SELECT id,firstName FROM students ORDER BY id
- ניתן לסדר בסדר עולה - ASC (דיפולטיבי), ובסדר יורד - DESC .
- ניתן להגדיר סדרים פנימיים (משני)

$Aggregate\ Functions$

- למשל: SELECT COUNT(*) FROM students
- $count(*)$ - מספר השורות בטבלה, $count(\boldsymbol{col})$ - מספר השורות ללא $null$ בעמודה זו.
- ממוצע - $AVG()$, סכום - $SUM()$, $MAX(), MIN()$

: $GROUP\ BY$

- אם נרצה (למשל) לסכום משהו לפי קטגוריה , נשתמש ב $GROUP\ BY$

- SELECT year,COUNT(year) FROM courses GROUP BY year
- SELECT name, COUNT (lecturer) FROM courses GOURP BY semester

$HAVING$ - תנאי על $GROUP$

- למשל:

- SELECT year,COUNT(year) FROM courses GROUP BY year HAVING COUNT(year) > 1

- כלל אצבע:
  - התנאי WHERE יגיע **לפני** GROUP (לסנן רשומות)
  - התנאי HAVING יגיע **אחרי** GROUP (לסנן קבוצות)

סדר מימוש הפעולות:

- SELECT coursId ,AVG(grade) FROM (SELECT * FROM grades) WHERE passed >) GROUP BY courseId HAVING AVG(grade) < 70

1. מבפנים החוצה
2. הטבלה שאחרי ה $FROM$
3. תנאי ה $WHERE$
4. חלוקה לפי $GROUP\ BY$
5. תנאי ה $HAVING$
6. בחירת העמודות המופיעות ב $SELECT$

שליפה מ 2 טבלאות או יותר - $JOIN...ON$

- **חיתוך** בין 2 טבלאות (לפי עמודה מסויימת)

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

- ניתן גם לצרף יותר מטבלה אחת.

שיעור 2 - 3/5/19

$LEFT\backslash RIGHT\ OUTER\ JOIN$

- בנוסף לחיתוך מוסיף את העמודות שנמצאות בטבלה השמאלית/ימנית ולא נמצאות בטבלה הימנית/השמאלית

$UPDATE$

- באופן כללי:

- UPDATE **table** SET **col1=val1,col2=val2....** WHERE .....
- UPDATE grades SET grade=78, passes = 1 WHERE studentId=111 AND courseId = 20

$DELETE$

- DELETE FROM **table** WHERE .....

$CREATE$

נצור טבלה לחיות מחמד:

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

- ניתן להגדיר ערכים לעמודות $INT, REAL, BOOLEAN, XML, DATETIME...$
- $char(x)$ אורך הסטרינג

$KEYS$

- $PRIMARY\ KEY$- עמודה /מספר עמודות שמגדירה את הגישה לטבלה
- $UNIQUE\ KEY$ - עמודה שאין בה כפילויות , אך יכולה להיות עם ערך $null$ (למשל מספר דרכון)
- $INDEX$ - גישה מהירה על פי עמודה

$DROP$

- כלומר - DROP TABLE **table**
- כדאי מאוד להזהר עם זה..

$ALTER$

- הוספת/מחיקת עמודה לטבלה

- **ALTER** TABLE pet **ADD** death DATE - הוספת עמודה

- **ALTER** TABLE pet **DROP** death - מחיקת עמודה
- **ALTER** TABLE table_name MODIFY COLUMN column_name datatype; - עדכון עמודה

CASE

```
Select
    case when condition_1 then value1
    when condition_2 then value2
    else value3
    end as alias
from table;
```

דוגמה: כתבו שאילתה המחזירה את כל המוצרים – Id, שם, ובמקום הקטגוריה ירשום Yes אם המוצר שייך ל Beverages או Condiments ו NO אחרת

```
select p.ProductID, p.ProductName,
    case when c.CategoryName in ('Beverages', 'Condiments')
    then 'Yes'
    else 'No'
    end as category from Products as p, Categories as c
```

Variables

- SET @var:= val
- ניתן להצהיר על סוג המשתנה
- DECLARE @var AS INT

טבלה זמנית

CREATE TEMPRARY TABLE **tempTable**
INSERT INTO **tempTable...**

ALIASES

- נתינת שם אחר לטבלה או עמודה
- SELECT * FROM students AS s.....
- לשאילתה פנימית, חובה לתת שם ע"י *AS*

TRANSACTION

- סט פעולות (שאנחנו מגדירים) שמונעות את בעיות הסנכרון שהזכרנו, ובעצם שומר על ה *ACID*
- ב *MySql* מספיק להגדיר *START TRANSACTION* לפני הפעולות הדרשות, ו *COMMIT* אחרי
- אם ארע *ERROR* במהלך סט הפעולות, *MySQL* הוא יחזיר מיד את המצב לקדמותו
- ב*WorkBench* צריך לכתוב *ROLLBACK COMMIT* בסוף הסט, (בשביל להחזיר את המצב לקדמותו)

- דוגמה, הבעיה:

Suppose we want to transfer money from one bank account to another:

- SET @transferAmount = 1000;
- 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;

What might be the problem with this execution?

What if we run another instance of this query at this point?

What if the program crashes here?

- כפי שאמרנו מחובתנו לשמור על ה $ACID$, למשל אי אפשר שנמשוך רק מלקוח כסף ושתוכנה תקרוס באמצע בלי שלקוח קיבל את הכסף.
- מכיון שכל פעולה לכשעצמה מקימת את ה $ACID$ נאגוד את כל הסט כ$Trancation$, למשל:

```
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
```

1.1 $STORED\ PROCEDURES$

- שמירה של שאילתה על השרת
- יתרונות: אם כמה קליינטים מנסים להריץ אותה, אז פשוט מריצים את המידע שקיים על השרת

```
DELIMITER $$
CREATE PROCEDURE student_avg
(IN stId INT)
BEGIN
  SELECT AVG(grade) FROM grades WHERE
studentId = stId;
END $$
DELIMITER ;
```

We need to change the delimiter, which separates commands, since we use ';' inside the stored procedure. When we finish we change it back to ';'

- פעולות אפשריות:
  - CALL student_avg(111);
  - DROP PROCEDURE student_avg
  - אין $ALTER$, לשינוי צריך למחוק ולצור חדש

1.2 $Triggers$

- נניח ואנחנו רוצים שתהיה עמודה עם ממוצע הציונים לתלמיד
- ונניח שביצענו שאילתה שעושה זאת, ישנה בעיה - והיא שהעמודה לא מתעדכנת, כלומר שינוי של ציוני של תלמיד לא ישפיע על העמודה של המומצע, לכן:

```
DELIMITER $$
CREATE TRIGGER new_grade_received
AFTER INSERT ON grades
FOR EACH ROW
BEGIN
   UPDATE students SET avg_grade = (SELECT AVG(grade) FROM
grades WHERE studentId=NEW.studentId) where id =
NEW.studentId;
END$$
DELIMITER ;
```

You can only define one event for each trigger (so might need multiple triggers)

You can use BEFORE if you want to access the DB before the change was made

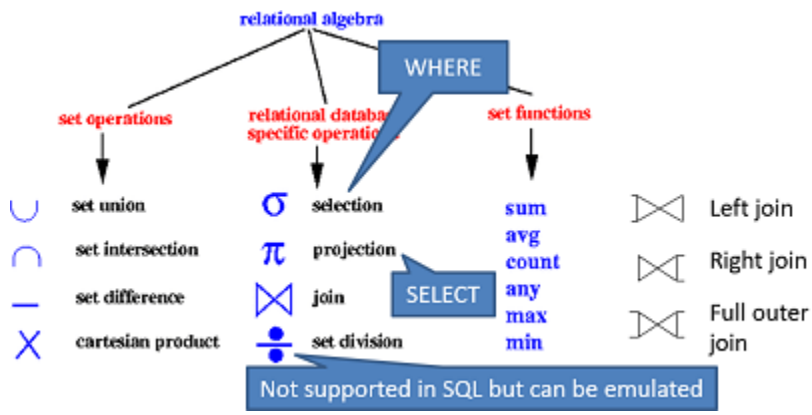
- וכעת כאשר יתבצע:
- INSERT INTO grades (courseId, studentId, grade, passed) VALUES (50, 111, 87, 1);
- הטריגר יופעל.
- מחיקה: DROP TRIGGER new_grade_received;

$VIEW$

- יצירת טבלה המבוססת על טבלאות בDB , לצורכי תצוגה בלבד (כלומר הטבלה לא באמת קיימת, ורק מחברת מידע הקיים בטבלאות השונות), **אך ניתן להשתמש באופן רגיל לגמרי**. למשל ליצירה:
- CREATE VIEW avg_grades_view AS
  SELECT students.firstName, AVG(grade) AS average
  FROM grades
  INNER JOIN students ON grades.studentId = students.id GROUP BY studentId;
- כעת קבלת נתונים: SELECT * FROM avg_grades_view
- ועדכון: UPDATE avg_grades_view SET average=75 WHERE firstName LIKE 'Chaya';
- הערה (ע"פ גוגל): באמת ניתן להוסיף/לעדכן/למחוק כל עוד זוהי טבלה "פשוטה" .

אם ה$view$ מורכב Subquery , JOIN ,GROUP BY, DISTINCT, Aggregate functions, וכד' זה לא יתאפשר. ע"פ: http://www.mysqltutorial.org/create-sql-updatable-views.aspx

2 אלגברה רלציוניות



נותן לנו אפשרות לכתוב שאילתות בצורה מתמטית, למשל:

- SELECT firstName, id FROM students WHERE degree > 1
- $\prod_{firstName,id}\left(\sigma_{(degree>1)}(Students)\right)$

נניח את מסד הנתונים הבא:

$$\begin{array}{cc} Person(name, age, gneder) & key = (name) \\ Frequnets(name, pizzeria) & key = (name, pizzeria) \\ Eats\,(name, pizze) & key = (name, pizze) \\ Serves\,(pizzeria, pizze, price) & key = (pizzeria, pizza) \end{array}$$

- <u>מצא מסעדות המבוקרות לפחות ע"י מבקר אחד בן פחות מ18:</u>

```
Select pizzeria from Frequnets
    left join Person on Frequnets.name = Person.name
    where Id < 18
```

$$\prod_{pizzeria}\left(\sigma_{age<18}\left(Person\right) \bowtie Frequnts\right)$$

- <u>מצא את שמות כל הנשים שאכלו פיצה זיתים או נפולטינה או שניהם</u>

```
Select name from Eats
    left join Person on Frequnets.name = Person.name
    where pizze in ("olive" , "napolitana") and  gender in ('female')
```

$$\prod_{name}\left(\sigma_{gender='famle' \wedge (pizza=napolitana' \vee pizza='olives')}\left(Person \bowtie Eats\right)\right)$$

3 *Normalization*

3.1 *Dependencies* :

- הגדרה: נאמר ש $attribute$ (או כמה) B תלויים ב $attribute$ (או כמה) A אם קיים קשר (=פונקציה) כך ש: $A \to B$

במילים אחרות, בהנתן A לא קיימים שני B שונים כך ש: $\begin{array}{c} A \not\to B \\ A \not\to B' \end{array}$ (הגדרת פונקציה)

- דוגמאות:
  - $A = ID$, $B = name$ נאמר שB תלוי פונקציונלית ב A

A	B	Dependency?
{Street, City, State}	Zip code	A→B
Day of week	Date = {Day, Month, Year}	B→A
First Name	Last Name	None
{University, Department}	DepartmentHeadId	A→B and B→A

3.2 $KEYS$

$candidte\ keys$ - סט מינמלי ליצירת שורה/לפניה לשורה. לכן שאר ה $attributes$ **תלויים** בו.

לדוגמה:

id	name	lecturer	year	semster
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Program...	David Gol	2022	2

- id יכול להיות ה key
- וגם $\{name, year, semster\}$ - היות ויכול להיות קורס אחד מסוגו בסמסטר.
- בטבלה יכול להיות יותר ממפתח אחד

$Prime$ - אם תכונה (עמודה) היא חלק מ $candidtae - key$ (סט מינימלי) כשלהו. , אחרת היא $non - prime$

$Super - Key$ - כל סט שמגדיר שורה באופן יחיד (כמו $candidate$ בלי המינמלי)

3.3 NF

3.3.1 $1, 2, 3, 3.5, 4$

$1 - NF$

- כל רחיב הוא אטומי ($searchablity$) , למשל טבלה שבה תהיה עמודה המחזיקה שם ושם משפחה יחדיו, אינה $1 - NF$ כי לא נתין להגיע בחיפוש לשם משפחה (אלא על ידי עיבוד המידע המתקבל מהשיאלתה)

$2 - NF$

- חייבת להיות $1NF$
- תכונה שהיא $non - prime$ אסור שתהיה תלויה בתת קבוצה חלקית של candidate key
- (אם לא מפתח תלוי בחלק ממפתח - בעיה)
- דוגמה - למצוא טובה יותר
  - $R(author, bookId, \#pages) \overset{\text{c.key}}{\Leftarrow} \{auther, bookId\}$
  - הבעיה: $bookId \to \#pages$ ו $\{bookId\}$ אינו key

פתרון לטבלה שיש לה עמודות תלויות שכאלה, הוא חלוקה לב טבלאות. בדוגמה

$$R1\,(auther, bookId)$$
$$R2\,(bookId, \#pages)$$

$3-NF$

- חייבת להיות $2NF$
- תכונה שהיא $non-prime$ אסור שתהיה תלויה בתת קבוצה שאינה super-key
- דוגמה $R1\,(studentId, courseId, grade, passed)$ $\{studentId, courseId\} \overset{\text{c.key}}{\Leftarrow}$
  - מתקיים ש: $\{studentId, courseId\} \to grade$ $\{studentId, courseId\} \to passed$ זה $2-NF$ כי אין עמודה התלויה בחלק ממפתח $\{courseId, grade\} \to passed$
  - נשים לב ש $passed \notin key$ ולכן היא $non-prime$
  - מצד שני $\{courseId, grade\} \to passed$ ו$\{courseId, grade\}$אינו $super-key$

$3.5NF$ = $BCNF$

- חייבת להיות $3-NF$
- לכל שני סטים X, Y כך ש $Y \nsubseteq X$, $X \to Y$, X הוא $super-key$

<u>וענה: $3NF$ עם single candidate אז $3.5NF$</u>

הוכחה:

- אם $x \to y$ וy $non-prime$ סיימנו
- אחרת y $prime$ לכן הוא חלק מקבוצה שהיא canditdae key נניח שקבוצה זו היא $\{A, B, ..., Y, ..\}$
- מכיון ו x קבוצה , ו $x \to y$ נוכל להחליף את y בקבוצה, ולקבל סט $\{A, B, .x_1, x_2, x_3, ...\}$, קבוצה זו היא $super-key$
- מהגדרת ה$super-key$ ניתן להופכו לסט מינמלי על ידי זריקת מפתחות, עד לקבלת key לכן:$\{A, \cancel{B}, .\cancel{x_1}, \cancel{x_2}, \cancel{x_3}, ...\}$
- ובהכרח: $\{A, \cancel{B}, .\cancel{x_1}, \cancel{x_2}, \cancel{x_3}, ...\} \neq \{A, B, ..., Y, ..\}$ - כי החלפנו את Y
- וזו סתירה לנתון שיש $single-candidtae$

לסיכום:

$$2NF: \underset{\text{connot be part of key}}{X} \longrightarrow \underset{\text{non-prime}}{Y}$$

$$3NF: \underset{\text{super key}}{X} \longrightarrow \underset{\text{non-prime}}{Y}$$

$$BCNF: \underset{\text{super key}}{X} \longrightarrow \underset{\text{any}}{Y}$$

<u>במילים (מגיל לוי):</u>

- אם B לא חלק ממפתח $\leftarrow A$ לא מוכל ממש במפתח
- אם B לא חלק ממפתח $\leftarrow A$ מפתח
- אם B כלשהו $\leftarrow A$ מפתח
- <u>כללים למציאת מפתח:</u>
  - עמודה שנמצאת בצד ימין ולא בשמאל - בהכרח לא חלק מהמפתח

- עמודה שנמצאת בשמאל ולא בימין - בהכרח חלק מהמפתח (אין מי שיביא לי אותה)
- עוברים על השילובים האפשריים עם עמודה אחת, שתיים...

- אם הכל $Prime$ אז 2 ו 3 מקבלים חינם
- אם יש רק key אחד והטבלה היא כן $3-NF$ אז היא גם $BCNF$
- (פסילת $BCNF$ מסתכלים על התלויות ובודקים אם יש עמודה המתקבלת ממשהו שהוא לא מפתח)

$multivalued-dependency$, **הגדרה:**

- יהיו A, B סטים של תכונות, נאמר ש A multidetermines B ונסמן : $A--->>B$ אם:
  - אם $C = R\setminus(A\cup B)$ (שאר התכונות)
  - ויש לנו שתי שורות כך ש:
    - $x[A] = y[A]\ and$ *
    - $x[B] \neq y[B]\ and$ *
    - $x[C] \neq y[C]$ *
  - ויש שורה כך ש:
  - $z[A] = x[A]\ and\ (= y[A])\ and$
  - $z[B] = x[B]\ and$
  - $z[C] = y[C]$

	A	B	C
x	$\underline{a_1}$	b_1	c_1
y	$\underline{a_1}$	b_2	$\underline{c_2}$
z	$\underline{a_1}$	$\underline{b_1}$	$\underline{c_2}$
w			

$NF-4$

- חייב להיות $3.5NF$
- אסור שתהיה שורה המקיימת $multivalued\ dependencies$.

StudentId	Department	SportTeam
111	CS	Soccer
111	Biology	Soccer
111	CS	Baseball
111	Biology	Baseball

StudentId	Department	SportTeam
111	CS	Soccer
111	Biology	Soccer
111	CS	Baseball

- נתבונן בשלוש השורות האחרונות: ישנה שורה:
  - $x[id] = y[id] = 111$
  - $x[Depar] = CS \neq Biology = y[Depar]$
  - $x[sprot] = Baseball \neq soccer = z[sport]$
- וקיימת שורה (הרביעית):

- $z[id] = x[id] = y[id] = 111$
- $z[Depar] = Biology = Biology = x[Depar]$
- $x[sport] = Baseball = Baseball = y[sport]$

4 $XML - JSON$

4.1 $XML(eXtensible - Markup - Language)$

- xml לא "עושה"/"רץ" - זו רק דרך להעברת נתונים
- xml תקין נקרא $well\ formed$
- יש לו $root - element$ יחיד, והוא בנוי בצורה הרכית
- יש $attribute$, $Case - snestive$
- סגירת $element$: <element>....</element>
- אוביקט ריק ניתן לסמן: <Studnet / > (רק סלאש בסוף, ללא אלמנט סוגר)
- $Entity - References$, לדוגמה < = < ועוד.
  - דומה ל $HTML$ אבל שונה ממנו - XML הוא לא $HTML$, ו$HTML$ הוא לא XML
  - הראנו איך לקרוא XML בגאוה

4.2 $XPath$

- You can try using XPath (and XQuery) at: http://www.xpathtester.com/xpath
- הספירה מתחילה מ1 (ולא מ0)

path exp'	Desc
/	בחירת ה $root$
/rootTag	בחירת התג שנקרא $rootTag$
//tagName	מצא את ה$tagName$ שמסמך
text()	קבלת הטקסט שב$node$ הנבחר
@$name$	קבלת ה $atriibute$, @$name$
..	קבלת האב
[]	תנאי
avg,count,max,contains	פונקציות אגרגטיביות
University/Student/*	החזרת כל האלמנטים
University/Student/FirstName | /University/Student/Address	פעולת or מסומנת | - כל שמות וכתובות הסטודנטים
University//Street	החזרת כל הצאצאים (ולא בהכרח צאצא ישיר)

/doc/chapter[5]	פרק 5 תחת הdoc
/body/p[last()]	ה p האחרון במסמך (פיסקה)
/body/p[@class='a']	הp עם $class\ attribute$ השווה ל a
//p[@class and @style]	כל ה p במסמך שיש להם $@class$ ו $@style$ כ$atrribute$

$Xquery$ 4.3

- from: https://www.w3schools.com/xml/xquery_intro.asp

נניח ויש לנו קובץ של חנות ספרים, המכיל מידע על הספרים: קטגוריה, שם, סופר, שנת יציאה, מחיר. לדוגמה:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
....
</bookstore>
```

select

כפי שהראנו ב $XPAth$ השורה הבאה תחזיר לנו את כל שמות הספרים

```
doc("books.xml")/bookstore/book/title.text()
```

ובקשה :

```
doc("books.xml")/bookstore/book[price>30]
```

תחזיר את כל **הספרים** שמחירם גדול מ30

$FLWOR$ 4.3.1

- For - selects a sequence of nodes
- Let - binds a sequence to a variable
- Where - filters the nodes

- Order by - sorts the nodes
- Return - what to return (gets evaluated once for every node)

עבור הדוגמה עם הספרים שמחירם גדול מ30 נוכל לקבל את אותה תוצאה (נצמצם לקבלת ה $title$) ע"י:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

היתרון שנוכל לסדר את התוצאות:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

אפשר גם להשתמש ב $if - then - else$

XQuery

- Return the area of Mongolia.
- for $x in doc("countries.xml")//country where $x/@name = "Mongolia" return $x/data(@area)

- Return the average population of Russian-speaking countries.
- let $rus := doc("countries.xml")//country[language="Russian"]/@population
- return avg($rus)

4.4 $Validation$

- הראנו איך אנחנו מונעים חוסרים במסד נתונים רלציוני אבל איך נעשה זאת ב XML ?

4.4.1 DTD

- מעט מיושן
- יחסית פשוט אבל חלש לעומת ה XSD

```
<!DOCTYPE Name [.....]>
<!ELEMENT Name ContentSpecification>
   EMPTY, ,ANY, ELEMENT CONTENT, MIXED CONTENT
#PCDATA = Parse Text Data
```

commands	example/pattern	explantion

<!DOCTYPE Name [.....]>	<!DOCTYPE University [.....]>	העטיפה של המסמך
יצירת *elemnt*	<!ELEMENT Name ContentSpecification>	
ANY	<!ELEMENT WORLD ANY>	
EMPTY	<!ELEMENT IMG EMPTY>	אובייקט תמונה ריק
ELEMENT CONTENT	<!ELEMENT Name (#PCDATA)>	שם המכיל $\#PCDATA$ (*string*)
MIXED CONTENT	<!ELEMENT CD (TITLE, ARTIST)>	אובייקט בשם CD שחייב להכיל $TITLE$ ו $ARTIST$ בסדר הזה
	<!ELEMENT DWELLING (HOUSE|APATRMENT| TRAILER)>	האובייקט מכיל **אחד** מהשלושה שבסגוריים
מספר מופעים	<!ELEMENT student (FirstName, LastName+, Address*, id, age?)>	? = מופע 1 או 0. + = לפחות מופע 1 . * = לפחות 0 מופעים (כרצוננו)
הוספת *attribute*	<!ATTLIST ElemName AttrName Type DefultDecl>	
CDATA	<!ATTLIST PACKAGE status CDATA “OK”>	
Enumerated - one from a list	<!ATTLIST DWELLING type (house | apartment | trailer) >	
	<!ATTLIST PACKAGE ontime (yes | no) #REQUIRED >	ה*attribue* , *ontime*נצרך חייב, וערכו הוא yes or no
	<!ATTLIST PACKAGE ontime (yes | no) #IMPLIED >	ה*attribue* לא הכרחי
	<!ATTLIST PACKAGE ontime CDATA #FIXED “yes”>	*ontime* יקבל את הערך *yes*

```
<?xml version="1.0"?>
<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>
 <Student>
   <FirstName>Tal</FirstName>
   <LastName>Negev</LastName>
   <id>222</id>
   <Address>
      <Street>Rotem 53</Street>
      <City>Jerusalem</City>
      <Zip>54287</Zip>
   </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

```
<!DOCTYPE University
[
<!ELEMENT student
(FirstName,LastName,Address*,id,age?)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
]>
```

Element "student" must contain: FirstName,LastName, multiple Addresses ,id and optional age

FirstName and LastName are of type #PCDATA (parseable text data)

Note that DTD has no hierarchy.

4.4.2 $XML - Schema$

- בעל יותר כח, למשל ניתן לשלוט כמה אלמנטים,
- תמיד יהיה קובץ נפרד
- יש להגדיר בהתחלה את $namespace$ שנשתמש בו בד"כ $xs\backslash xsd$:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="elemName">
        ⋮
    </xs:element>
</xs:schema>
```

- הגדרת $element$: <xsd:element name type>
- הגדרת $type$ ע"י ... | date | anyURy | sPositiveInteg | integer | decimal | boolean,| xsd:string
- למשל:

```
<xsd:element name="MyElem" type=xsd:string>
</elemtn>
```

- ניתן לשלוט במספר המופעים ע"י $min\backslash max\ occuer$

<xsd:element name="Friend" type=xsd:string minoccurs="1" maxoccurs="unbounded">

  - באופן דיפולטיבי הם מוגדרים לאחד (=מופע אחד בדיוק)

- ניתן להוסיף $type$ משלנו, ובכך לצור הגבלות המתאימות לצרכינו ע"י $SimtpleType$ ו $restriction$

```
<xs:element name="dwelling">
    <xsd:simptleType>
        </xsd:restriction base="xsd:string">
            <xsd:enumeration value="house">
            <xsd:enumeration value="apartment">
            <xsd:enumeration value="trailer">
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

- ניתן גם להגדיר *restriction* על ידי *regax* למשל: <xsd:pattern value="[A-z]">
- *anyType* - אובייקט ללא הגבלות בכלל.
- *ComplexType* :
  - נגדיר אובייקט ריק כך:

```
<xs:element name="dwelling">
   <xsd:complexType>
   </xsd:complexType>
</xsd:element>
```

  - נגדיר אובייקט מורכב כך:

```
<xs:element name="dwelling">
   <xsd:complexType mixed="true">
     <xsd:sequnce>
       <xsd:element name="title">
     </xsd:sequnce>
   </xsd:complexType>
</xsd:element>
```

    * *sequnce* - הרכיבים הפנימיים צריכים להופיע בסדר זה
    * *choice* - ניתן לבחור אחד מהרכיבים הפנימיים
    * *all* - הרכיבים יכולים להופיע בכל סדר שהוא

- *attribute* :
  - נגדיר <xsd:attribute name type ?use?>
  - תמיד יוגדר כ *simpleType* , והtypeשלו כמו של *elemnt*
  - *use* (לא חובה) - $required\backslash optional\backslash prohibited\backslash defualt$
  - צריך להופיע בסוף
  - דוגמא 1:

```
<xs:element name="myImg">
```

```
    <xsd:complexType>
      <xsd:attribute name="src" type="xsd:string" use=required>
    </xsd:complexType>
  </xsd:element>
```

- דוגמה 2 :

```
  <xs:element name="Car">
    <xsd:complexType>
      <xsd:sequnce>
        <xsd:element name="make" type="xsd:string">
        <xsd:element name="model" type="xsd:string">
      </xsd:sequnce>
      <xsd:attribute name="year" type="xsd:string">
    </xsd:complexType>
  </xsd:element>
```

דוגמת הסטודנטים

```
<?xml version="1.0"?>
<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>
 <Student>
   <FirstName>Tal</FirstName>
   <LastName>Negev</LastName>
   <id>222</id>
   <Address>
      <Street>Rotem 53</Street>
      <City>Jerusalem</City>
      <Zip>54287</Zip>
   </Address>
 </Student>
 <Student/>
<!-- A comment about the file... -->
</University>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="University">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Student" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="FirstName" type="xs:string" />
              <xs:element name="LastName" type="xs:string" />
              <xs:element name="age" type="xs:int" />
              <xs:element name="Address">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Street" type="xs:string" />
                    <xs:element name="City" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          <xs:attribute name="degree" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

This is where we get all the types from. Note the "xs" namespace prefix in all elements.

If we don't state either minOccurs or maxOccurs, the default is 1

"Address" is a complex type inside the "student" type

XSD supports many types

This allows the students to have a degree attribute (optional). We could force the students to have a degree attribute by: <xs:attribute name="degree" use="required"/> And have a default attribute by: <xs:attribute name="degree" default="BSc"/>

4.5 $JSON$

- פורמט פשוט, תוכנן עבור $JavaScript$
- מבנה $\{"idex" : "value\}$
- $Schema\backslash Path$ - קיימים, לא נפוצים, לא הרחבנו.

5 $NoSQL$

Relational Databases	NoSql Databases
Powerful, flexible, constrained.	Document model more closely matches code objects.
Designed for database administrators.	Designed for developers.

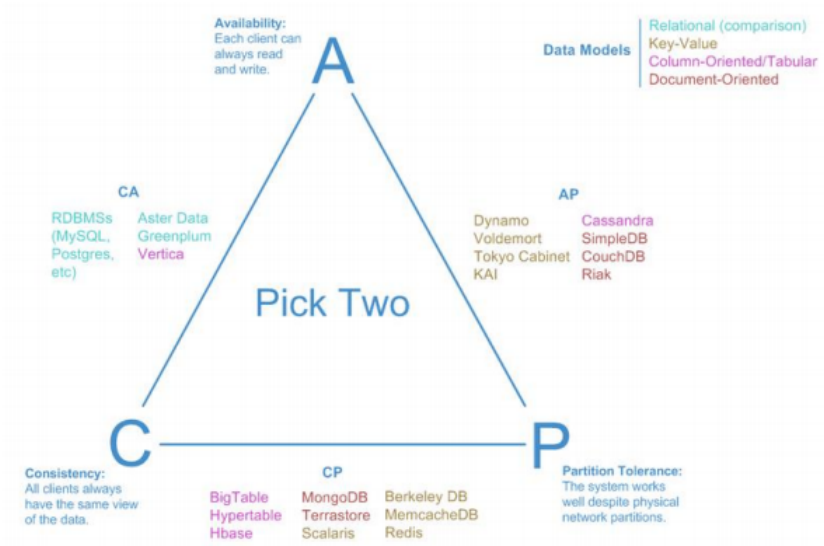
- ידוע גם כן $not-only-sql$ כלומר לא להשתמש ב $retaional-data-storage$
- משתדלים להמנע מ $join$ (שמכביד מאוד על כמות המידע), יותר גמיש, ותומך ב $big-data$
- למה זה טוב?
  - $Scalabilty$ (היכולת לגדול בקלות)
  - כשרוצים **מבנה** גמיש
- המחיר
  - השאילתות מוגבלות
  - לא נוכל להבטיח קיום $ACID$

5.0.1 $cap\ theorm$

טענה: אין מסד נתונים שיכול לקיים את שלושת התנאים הבאים במקביל:

- $Consistency$ - כל קריאה צריכה לקבל בחזרה ערך או טעות
- $Availabilty$ - כל קריאה צריכה לענות עם הערך האחרון
- $Partition\ tolerance$ - יכולת המערכת להגיב, לכמה מחשבים במקביל

מסד נתונים רלציוני - בוחר להצמד לשני העקרונות הראשונים. ואנו נלמד מסדי נתונים שבחרו לעקרנות אחרים בציור:



וכעת העקרון שינחו אותנו הוא $BASE$

- $basically - Avilable$: הנתונים זמינים לרוב
- $soft - State$: המצבים יכולים להשתנות , גם ללא עדכון (כי עדכון ישן עדיין רץ)
- $Eventual - consistency$: בשלב סופי כלשהו ה $data$ תהיה עקבית

5.1 $redis$

- למעשה זהו $hash - map$ גדול
- לשים לב ב $redis$ מקבלים את ה "$value$" בשלמותו (אין $select$שמחזיר רק חלק מה$node$)

- You can try it online at: https://try.redis.io/

להלן הפקודות ב $redis$:

command	example	הסבר
	פעולות כלליות	
set	▷set name “David”	הכנסת "רשומה"
get	▷get name “David”	קבלת "רשומה
getset	▷ getset name “Dan” “David” (בגלל ה get)	קבלת רשומה ועדכונה
mset/mget	▷set name “Dan” pass “aaa”	השמה/קבלה של מספר רכיבים
del	del name	מחיקה
exist	▷ exist name 0	קיים 1 אחרת 0

incr	▷ SET bottles 5 ▷INCR bottles (integer) 6	כמו x++
INCRBY key x	▷INCRBY bottles -3 (integer) 3	כמו x+= 3
expire	▷EXPIRE user:302 100	ימחק לאחר 100 שניות
	Hashes	
hmset	▷HMSET user:302 degree 3 gender "female"	יצירת $user : 302$ כ$HashMap$
hset	▷HSET user:302 last_name "Cohen"	עדכון ערך יחיד
hgetall	▷HGETALL user:302	קבלת כל $user : 302$
hmget		קבלת מספר ערכים
hget	▷HGET user:302 first_name "Tamar"	קבלת ערך יחיד
hexsit	▷hexsit user:302 address 0	האם קיים
	lists	
Rpush	RPUSH user:571:items "chair" "table" "water"	מוסיף לסוף
Lpush	LPUSH user:571:items "cup"	מוסיף להתחלה
lrange	▷ LRANGE user:571:items 1 2 (1) “chair" (2) "table"	מחזיר את הערכים שבטווח.
	LRANGE user:571:items 0 -1	מחזיר מהראשון עד האחרון (-2 יחזיר עד לפני אחרון, וכו')
lpop/rpop	▷LPOP user:571:items “cup”	שולף את הראשון
	Sets	
sadd	SADD mySet x y	סט עם ערכים x, y
srem	SREM mySet y	מחיקת הערך y מהסט
sismember	SISMEMBER mySet y	האם שייך לסט
smember		מחזיר את מה שבסט
SUNIOUN		מחזיר את איחוד הסטים
ZADD x val y	(add item y with score val to sorted set x)	
ZRANGE x y z	(return z items from x, starting at y)	

$\underline{SET, GET, DEL}$

- get על מפתח לא קיים

▷GET "world"

(nil)

- נהוג להכניס את הערך עם ה id + אפשר להכניס $string$ מורכבים

▷ SET user:1001 "<user><first_name>Joel</first_name> <last_name>Cohen</last_name></user>"

▷GET user:1001

"<user><first_name>Joel</first_name><last_name>Cohen</last_name> </user>"

$\underline{: INCER, INCRBY}$

- $incr$ לא על $integer$:

▷INCR name1

(error) ERR - can't increment string

$\underline{\text{קבלת } KEYS \text{ על ידי } pattern}$

נניח שיש לנו רשימה של $user : 1000, user : 1001, user : 1002, ...$

ניתן לרשום:

▷Keys user:100?

1) "user:1000"

2)"user:1001"

3) "user:1003"

...

ואם נרשום: Keys user:* , נקבל את כל המפתחות עם $user$

5.2 *cassandra*

לב הרעיון, שיותר חשוב לנו לקרוא מידע מהר, מאשר לכתוב אותו מהר.

- לקסנדרה יש מנגנון replicition - ש"מגבה" לנו את המידע" באופן חסכוני במקום
- הsyntax מאוד דומה לsql , עם הגבלות : אין $join$ ואין $nested - queries$, נקרא cql
- למשל חיפוש שלא לפי מפתח מצריך בקשת allow ומאוד לא מומלץ להתשמש בזה (צריך לזכור שמדובר ב big data)
- בחיפוש חייבים לתת את ה partition key במלואו ולהתייחס אליו עם =
- ה clustring key יכול לקבל אי־שיוון וגם זה מהסוף להתחלה

תכונות:

- $No - fixed - shcema$: לcassndra אין $schame$ ולכן אורך השורות בטבלה לא חייב להיות זהה
- $eventually - consistent$, בזמן נתון יכול להיות שהמידע לא יהיה זהה בכל ה$raplica$של הטבלאות.
- $partition\ key$ - מגדיר את ה $node$ ב $cluster$ כאשר ניגשים ל"שורה"
- $clustering\ column$ - מגדירה את הסדר בו השורות מאוחסנות

- $Nodes$ - $server$ ב $cassandra\ cluster$ נקרא $node$

$Wide - Column - Store$

- כל key מזוהה עם שורה בעלת מספר אלמנטים. לכן ניתן לזה כאל מילון, key מפנה לערכים שלו.
- לכן ניתן להסתכל על זה כעל מעין טבלה, אך בפועל כל key מחזיק מספר משתנה של עמודות

$RMDB - Habit$

Cassandra	**RDBMS**
Cassandra is used to deal with unstructured data.	RDBMS is used to deal with structured data.
Cassandra has flexible schema.	RDBMS has fixed schema.
In Cassandra, a table is a list of "nested key-value pairs". (Row x Column Key x Column value)	In RDBMS, a table is an array of arrays. (Row x Column)
In Cassandra, keyspace is the outermost container which contains data corresponding to an application.	In RDBMS, database is the outermost container which contains data corresponding to an application.
In Cassandra, tables or column families are the entity of a keyspace.	In RDBMS, tables are the entities of a database.
In Cassandra, row is a unit of replication	. In RDBMS, row is an individual record.
In Cassandra, relationships are represented using collections.	In RDBMS, there are concept of foreign keys, joins etc.

from: https://www.javatpoint.com/rdbms-vs-cassandra

- בעקרון $RMND$ נרצה שבdb נכתוב מה שפחות (ונחסוך זכרון), ב $cassandra$ נעדיף לכתוב יותר על מנת לקרוא מהר
- ב$RMDB$ נרצה שבdb לא יהיו כפילויות, ב $cassandra$ בכוונה משכפלים ($replication$) על מנת לקרוא מהר.

$partition\ key$

- יכול לכלול יותר מ"עמודה" אחת
- תהליך עדכון מידע: מחפשים את ה$partition\ key$ ב$hashMap$ ← מעדכנים את המידע ב $node$ שהתקבל (וב $replicas$ שלו)
- בחיפוש חובה לציין את **כל** ה $partition\ key$ (אלא אם בקשנו את כל הטבלה) -בשביל ש$cassandra$ תדע היכן לחפש.

$clustering\ key$

- כל מי שמגיע אחרי ה$partition$ ב$parimary\ key$
- אחראים לסידור המידע ב"שורה"
- ברירת המחדל הוא בסדר עולה

פעולות

- $top-level-space$ נקרא $KeySpace$
  - כאשר נגדיר את ה $Keyspace$ נצטרך להגדיר גם אסטרטגית ה $replication$, ומס' העתקים
  - $'SimpleStrategy'$- יוצר העתקים כמספר שצוין, עם מרכז אחד
  - $'NetworkTopologyStrategy'$ - יוצר העתקים כמספר שצוין, אחד מחלק אותם במספר מרכזים.

<u>יצירת מסד נתונים :</u>

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

הערות:

- $'SimpleStrategy'$ - מתאים ל single data center only
  - בנוסף יש: NetworkTopologyStrategy , שמתאים למספר מרכזים?
- $replication-factor'$: מספר השכפולים שנעשה

- לא ניתן למיין את המידע בשאילתה, יש להגדיר את הסדר ביצירת הטבלה ע"י:

```
WITH CLUSTERING ORDER BY (col ASC|DESC)
```

כעת נצור <u>טבלת $students$:</u>

```
USE university;
CREATE TABLE students
(id INT PRIMARY KEY, firstName VARCHAR, lastName VARCHAR, age INT);
```

- חשוב להגדיר את ה key!
- $varchar$ כמו ב sql רק שלא צריך להגדיר אורך

<u>דוגמה נוספת</u>

```
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)
) WITH CLUSTERING ORDER BY (state_province ASC, city DESC, gym_name ASC);
```

- כפי שהסברנו , $partition\ key$ הוא $country\_code$
- ה $clustering$ הם $state, city, gum$ על פי סדר זה, וכאשר נבקש את המידע נקבל אותו על פי:
  - ה $state$ בסדר **עולה**
  - לאחריו ה $city$ בסדר **יורד**
  - ולאחריו gum_name בסדר **עולה**

- $insert$:

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

- חובה $single-quotes$ $('Glass')$

- $select$:

```
SELECT * from students
```

- כעת אם נבצע

```
SELECT * from students WHERE id=111
```

נקבל את $'Chya\ Glass'$ כרגיל, אבל אם נבצע:

```
SELECT * from students WHERE lastname='Glass';
```

נקבל שגיאה, כיון ש $cassandra$ לא מסנן בכלל על בסיס הנתונים **אלא** רק על ידי ה id

- הערה - <u>וממש **לא** מומלץ להשתמש:</u> להוסיף $Allow-Filtering$(בסוף השאילתה)

- שגיאה נוספת:

```
SELECT * from students WHERE id > 100;
```

- זהו ניסיון להשוות ביחס ל $partition-key$

- באופן כללי ניתן לחשוב על חיפוש כחיפוש $path$ של קובץ, לכן, עבור:

```
Table T with Primary keys: x as the partition key and y, z as clustering keys
```

- <u>שאילתות תקינות:</u>

  טווח רק בשלב האחרון - אם השאילתה תקינה, הסדר לא משנה

```
SELECT * FROM T WHERE x = 5;
SELECT * FROM T WHERE x = 5 AND y < 6 AND y > 0;
SELECT * FROM T WHERE x =2 AND z < 4 AND y = 3;
```

- <u>לא תקינות</u>

  לא על פי סדר + אסור לדלג

```
SELECT * FROM T WHERE y = 5;
SELECT * FROM T WHERE x = 5 AND z = 7
```

ניתן לחפש בטווח רק "בשלב האחרון":

```
SELECT * FROM T WHERE x = 5 AND y < 3 AND z > 0;
SELECT * FROM T WHERE x = 5 AND z = 4 AND y > 0;
```

5.3 $MongoDB$

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)

use - מחליף ל $mydb$ הדרוש, אם לא קיים יוצר כזה, למשל : use University

ב$mongoDB$ לטבלה קוראים $collection$

drop - בשביל למחוק: $db.dropDatabase()$

create - $collection$

- נצור על ידי הקריאה: db.createCollection("students") , המגדירה באופן אוטומטי את ה$collection$
- ניתן גם להגדיר מראש פרמטרים כך:

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

  - $capped$ יוצר את $room$ ריק ל$collection$ - אם השתמשנו ב $capped$ אז חובה להגדיר $size$
  - $size$ בבתים
  - max : של רשומות (סטודנטים)
  - $autoIndexID$ - $true$ זו הברירת מחדל, כיון שהגדרנו את הגודל

insert

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

- נשים לב שהפורמט הוא $json$
- ניתן להכניס "רכיב בתוך רכיב" (כמו ה $Address$)
- הכנסה של כמה ביחד היא הכנסה של מערך ולכן צריך לעטוף ב[...,...]

find()

- חיפוש/קבלת כל הערכים ש: db.students.find()
  - הדפסה יפה: db.students.find().pretty();
- מציאת ערך מסוים: db.students.find({"FirstName": "Tal"})
- רכיב בתוך רכיב: db.students.find({"Adress.**City**": "Ariel"})
- and : db.students.find(**$and:** [{"FirstName": "Tal"},{"LastName": "Negav"}])
  - או בקיצור : $[condition1, condtion2, ...]$ (בלי ה and)
- or : db.students.find(**$or**: [{"LastName":"Negev"},{"LastName":"Glow"}])
- כיצד נבקש: Select FirstName From Students ?
  - תשובה: db.students.find({},{"FirstName":true, _id:false})
  - הוספנו $id : false$ כיון ש $mongoDb$ נותן לאובייקט id , ואותו אנחנו לא רוצים לקבל.

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

<u>$update$:</u>

- עדכון רגיל מתבצע כך:

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

כלומר, תמצא את Tom ותגדיר את הפרטים שלו כמצוין

- על מנת לקצר אפשר לכתוב:

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

- כאשר set\$ פרושו תשאיר את שאר השדות כפי שהיו
- $multi : true$ - על מנת שיעדכן את כל ה Tom שאנחנו מכירים, אחרת יעדכן רק את הראשון שפוגש.

$romove$

פורמט:

```
db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

- $DELETION\_CRITERIA$ - מה למחוק
- 1 - משתנה $boolean$, האם למחוק רק 1 (הראשון שמוצאים) או את כולם
- אם רוצים להגביל למחיקת של N פרטים מוסיפים בסוף $limit(N)$

דוגמה:

```
db.movies.remove("movie name":"baby boss"})
//
db.movies.remove("movie name":"baby boss"}).limit(10)
```

פונקציות נוספות

- $skip(k)$ - דילוג על המסמך ה k שמצאנו

```
db.movies.find({},{movie_name":1_id:0}).skip(5)
```

- $sort()$

```
db.movies.find().sort({key_name:1})
```

  - $sort$ מקבלת ערכים: $ASC = 1$, $DESC = -1$

$Map - Reduce$

- $Mapper$ - מחלק את המידע, מסנן, מריץ את התהליך
- $Shufle\ and\ sort/Grouping$ - ודוא שכל ה$workers$ קיבלו $data$
- $Reduce$ - עיבוד המידע שברשות ה $worker$ לקבלת התשובה הסופית

<u>דוגמה: מצא את הציונים המינימלים והמקסימלים לפי גיל סטודנט:</u>

- Map - נמפה ציון לגיל - ע"י הפונקציה $emit()$
- $Grouping$ - כל $worker$ יקבל set על פי הגיל
- $Reduce$ - כל $worker$ מוצא את ה min הmax על פי הגיל.

5.4 *elastic search*

נח לעבודה עם מסמכים וחיפוש על פי מילות מפתח (בגלל מנגנון ה *scoring*)

- We will use curl for HTTP commands: basically GET, POST, PUT, HEAD and DELETE.
- There is no need to create a Database) or a table, we can right away insert a document using commands
- לכן למשל הכנסת רשומה חדשה:

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

  - ב $XPUT$ מציינים את ה id, ב $XPOST$ - מייצר id לבד

- לקבלת מידע נשתמש ב $XGET$

```
curl -XGET "http://localhost:9200/university/students/333" {
"_index":"university","_type":"students","_id":"33 3","_version":1,"found":true,
"_source":{"FirstName": "Gadi", "LastName": "Golan", "age": "24"}}
```

  - נשים לב שבשורה הראשונה מגיעים גם נתונים נוספים על החיפוש, ב $Source$ קיבלנו את המידע עצמו.

- $XHEAD$ - בודק אם רשומה קיימת
- $XDELETE$ - מחיקת רשומה

חיפוש - $\_\_search$

- כיצד נקבל את select * from Students ?
  - פתרון: curl –XGET "http://localhost:9200/university/students/**__search**"
- כיצד נקבל את SELECT * FROM students WHERE LastName='Negev' ?
  - פתרון: curl -XGET "http://localhost:9200/university/students/_search? q=LastName:Negev"

חיפוש מתקדם

נניח ויש לנו מסד נתונים $bank$, עם טבלת $account$ שבה יש $last\_name, first\_name, balance, acount\_number, state$ ועוד על מנת לקבל את Select * from accounts where state="CA" , נבצע:

```
GET bank/account/_search {
```

```
    "query": {
        "match": {
            "state" : "CA"
        }
    }
}
```

- זהו חיפוש פשוט על פי פרמטר אחד, כאשר $match$ לרוב מתאים לחיפוש על פי $text$
  - $match, trm, range, exists, type...$
- על מנת לחפש על פי מספר דרישות, נשתמש ב $bool$
- למשל עבור: Select * from accounts where state="CA" and first_name="Jhon"

```
GET bank/account/_search {
    "query": {
        "bool" {
            "must":
            [
              { "match": { "state" : "CA" }},
              { "match": { "first_name" : "Jhon" }}
            ]
        }
    }
}
```

- כאשר $must$ מתנהג כמו and , ו $should$ מתנהג כמו or , בנוסף יש $filter$ - שמסנן החוצה את כל מי שעומד בתנאי, $must\_not$ שמראה את שלילת התנאי.

$Scoring$

- היחודיות של $elastic - search$ הוא באפשרות לדרג את התוצאות, (דבר שמתאים מאוד לחיפוש במסמכים)
- בשאילתות עם $filter$ או $must\_not$ אין $scoring$ ($score = 0.0$)
- הדירוג יתבצע על פי מודל $TF - RDF$ שנלמד בהמשך
- בדוגמה הבאה ה $boost : 3$ אומר שרשומות עם $jhon$ יקבלו ניקוד של פי שלוש לעומת רשומה עם $"CA"$

```
GET bank/account/_search {
    "query": {
        "bool" {
          "must":
          [
```

```
            { "match": { "state" : "CA" }},
            { "match": { "first_name" :{"query": "Jhon","boost":3 }}}
          ]
        }
      }
    }
```

- כפי שאמרנו מאוד נח כאשר רוצים לחפש ביטוים במסמך, ניתן גם לבצע חיפוש מדויק על ידי החלפת $match$ ב $match\_phrase$

5.4.1 $Tf - idf$

- Tf - $Term - Frequency$. ספירה של כמה פעמים מילה מופיעה במסמך (משפט), לעומת מספר המלים במסמר $\frac{|\text{w in doc}|}{|\text{all doc }|}$
- idf - $Inverse - Document - Frequency$. לוג של סך המסמכים (כל הקטלוג) חלקי מספר המסמכים בה המילה מופיעה (בקטלוג)

כלומר, נחלק זאת למספר פרמטרים (d= מסמך יחיד, da= מסמך שמכיל מילה , doc = כל המסמכים, Q כל המילים ב d):

- כמה פעמים מופיעה מילה במשפט $tf = \frac{\text{\# times q in d}}{\text{\# word in d}}$
- כמה פעמים מילה מופיעה בכל המסמכים שלנו $df = \frac{\text{\#da have q}}{\#doc}$
- אנחנו רוצים להעלות את הדירוג של הנדירים: לכן ניקח את ההפוכה ונשים לוג: $idf = \log\left(\frac{\#doc}{\text{\#da have q}}\right)$
- סהכ: $\sum_{q=0}^{|Q|}\left(\frac{\text{\# times q in d}}{\text{\# word in d}}\right)\log\left(\frac{\#doc}{\text{\#da have q}}\right)$

למדנו שמאגר החיפוש מדרג את התוצאות, כיצד? לדוגמה

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?

נבנה טבלה, לפי המשפט שאנחנו מחפשים

	who	***is***	***the***	***President***	***Of***	***United***	***State***	***#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
#Doc	2	2	3	1	3	3	2	

כעת על פי הנוסחה

$$\sum_{q=0}^{|Q|}\left(\frac{\text{\# times q in d}}{\text{\# word in d}}\right)\log\left(\frac{\#doc}{\text{\#da have q}}\right)$$

- Q1; $\underbrace{0}_{who} + \underbrace{\frac{1}{6} \cdot \log\left(\frac{4}{2}\right)}_{is} + \underbrace{0}_{the} + \underbrace{\frac{1}{6} \cdot \log\left(\frac{4}{1}\right)}_{president} + \underbrace{0}_{of} + \underbrace{\frac{1}{6} \cdot \log\left(\frac{4}{3}\right)}_{United} + \underbrace{\frac{1}{6} \cdot \log\left(\frac{4}{1}\right)}_{state} = 0.736$
- Q2; $\underbrace{0}_{who} + \underbrace{0}_{is} + \underbrace{\frac{3}{15} \cdot \log\left(\frac{4}{3}\right)}_{the} + \underbrace{0}_{president} + \underbrace{\frac{2}{15} \cdot \log\left(\frac{4}{3}\right)}_{of} + \underbrace{\frac{1}{15} \cdot \log\left(\frac{4}{3}\right)}_{United} + \underbrace{0}_{state} == 0.166$
- Q3,Q4.....

מסדי נתונים גרפיים

- שומר את המידע כגרף
- מאפשר חיפושים לפי החיבורים חבר של חבר $(foaf)$ עד 10 רמות
- RDF :
  - מודל להעברת נתונים
  - ניתן למזג בקלות נתונים, גם אם ה $schemas$ שלהם שונה
  - תומך ב $schemas$ שהתפתחו בלי לעדכן את כל הנתונים
- $RDF - Triples$
  - בשיטה זו יש "טבלה אחת" המכילה בדיוק שלושה חלקים

$$Subject \xrightarrow{predicate} object$$

« נניח שיש לנו את 2 הרשומות הבאות ברשימה של דיסקים:

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988

```
<?xml version="1.0"?>

<rdf:RDF   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:cd="http://www.recshop.fake/cd#">

           <rdf:Description        rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
                      <cd:artist>Bob Dylan</cd:artist>
                      <cd:country>USA</cd:country>
                      <cd:company>Columbia</cd:company>
                      <cd:price>10.90</cd:price>
                      <cd:year>1985</cd:year>
           </rdf:Description>

           <rdf:Description        rdf:about="http://www.recshop.fake/cd/Hide your heart">
                       <cd:artist>Bonnie Tyler</cd:artist>
                       <cd:country>UK</cd:country>
                       <cd:company>CBS Records</cd:company>
                       <cd:price>9.90</cd:price>
                       <cd:year>1988</cd:year>
           </rdf:Description>
           .
           .
</rdf:RDF>
```

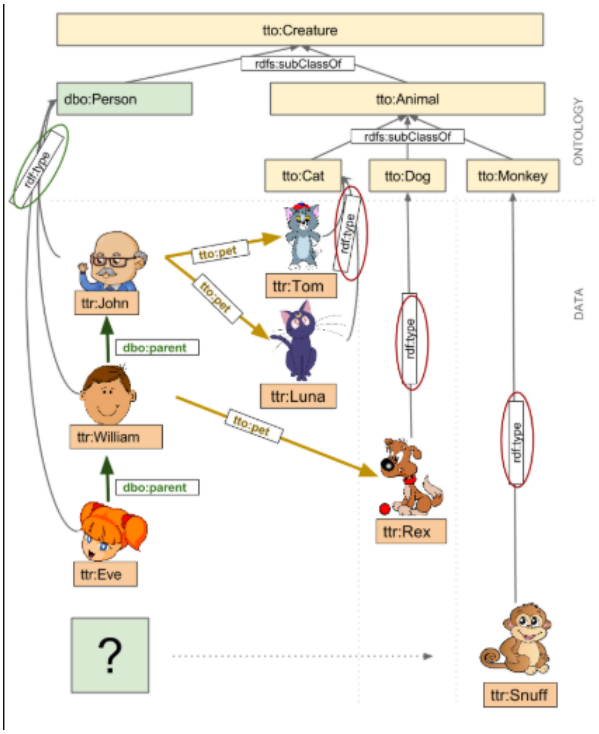
5.5 $ARQ/SPARQL$

- בקשת RDF נקראת $SPARQL$ וממומשת ע"י ARQ
- נלמד רק $SELECT$

- You can try SPARQL at: http://sparqlplayground.sib.swiss/

דוגמאות

נצייר חלק מה $data$ שלנו בגרף



- כעת אם נרצה את כל ה $data$ דבר השקול ל $Select *$ אז:

```
SELECT * WHERE {
    ?subject ?predicate ?object. }
```

ונקבל:

subject	predicate	object
ttr:Eve	dbo:parent	ttr:William
ttr:Eve	dbp:birth	Date "2006-11-03"
	⋮	
ttr:John	tto:pet	ttr:LunaCat
	⋮	

כלומר את כל הקשרים בגרף.

- הערה: ttr, tto, dbo הם כל מיני $namespace$ הרלוונטים ל $data$, $namespace$נפוץ הוא $foaf = friend - of - a - friend$

- אם נרצה את שמות כל האנשים (בנים ובנות):

```
SELECT ?something WHERE {
    ?something rdf:type $dbo.Person. }
```

נקבל:

something
ttr:Eve
ttr:John
ttr:William

, נשים לב ש $?something$ הוא שם הטור שהתקבל

- אם נרצה את שמות כל ה**נשים**

```
SELECT ?women_name WHERE {
    ?women_name rdf:type $dbo.Person.
    ?women_name tto:sex "famele".
}
```

– כלומר מצא את הקודקוד שיש לו קשר ל $person$ ול $female$

- האנשים שיש להם חתול

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

- האנשים ללא חתול

```
SELECT DISTINCT ?person WHERE {
    ?person rdf:type $dbo:Person.
    FILTER NOT EXISTS {
        $person tto:pet/rdf:type tto:Cat
}}
```

– חסכנו שורה על ידי השרשור

- כלל ה $Creatures$

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

```
}
```

- $a = ref : type$ (קיצור)
- כלומר מצא את כל הבנים והנכדים של $Creature$. מה קורה אם יש "דור שלישי" ?

```
SELECT ?thing WHERE {
    { ?thing a /rdf:subClassOf+ tto:Creatue }
```

- (+: $x \geq 1$, *: $x \geq 0$, ? : $x = 0|1$, כאשר x מספר מופעים)

5.6 $Neo4J$

Compare to SQL

RDBMS	Neo4J
Table	Graph
Row	Node
Column and Data	Property and its values
Constraint	Relationship
Join	Traversal

- שפת השאילתות $CQL - Cypher\ Query\ Language$
- ניתן לנסות: http://console.neo4j.org/

CQL	USAGE
CREATE 2 nodes:	Create (<node-name>:<label-name> { <Property1-name>:<Property1-Value>, ... , <PropertyN-name>:<PropertyN-Value> })
create - relationship:	Create (node1)-[:RelationshipType]->(node2)
create nodes + rel'	Create (Koby)-[:married]->(Michal),
MATCH	MATCH (node:label) RETURN node
	MATCH (node:label)-[: Relationship]->(n) RETURN n
RETURN	Return query results
WHERE	Provide conditions to filter retrieval data
DELETE	Delete nodes and relationships
REMOVE	Delete properties of nodes and relationships MATCH (node:label{properties . . . . . }) REMOVE node.property RETURN node
ORDER BY	Sort retrieved data
SET	Add or update labels
Aggregation	Count,Max,Min,Sum,Avg, Collect (return list of)

- מחיקת כל הקודקודם והקשתות הקיימים:

```
MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n,r
```

- יצירת שני קודקודים (לא מחוברים)

```
Create
(Michal:Person {name:'Michal', age: 36,gender:"female"}),
(Koby:Person {name:'Koby', age: 39,gender:"male"})
```

- יצירת חיבור בין קודקודים (יעבוד רק אם יתבצע ביחד עם השאילתה הקודמת, כיון שאחרת צריך לקבל קודם את $koby$ ומ $Michal$)

```
Create (Koby)-[:married]->(Michal)
```

- קבלת כל הקודקדים מסוג $Person$:

```
MATCH (n:Person)
RETURN n
```

- קבלת $person$ מסוג זכר (ע"פ הקשר שלהם)

```
MATCH (p:Person {gender:"male"})-[:owner]->(n)
RETURN n.name
```

- קבלת כל שמות בעלי החים של יואב

```
Match (a{name:'Yoav'})-->(b:Pet) RETURN b.name
```

- כל הקשרים שיש לקובי (לא רק ישירים):

```
MATCH (a {name:'Koby'})-[*]-(b) RETURN DISTINCT b
```

- כל הקשרים מסוג son ומסוג married:

```
MATCH (a)-[:son|:married]-(b) RETURN DISTINCT b
```

- חיפוש לפי תכונות: קבלת כל ה$person$ מסוג $male$ וגילם מעל 18:

```
MATCH (Person)
WHERE Person.gender = "male" AND Person.age>=18
RETURN Person
```

- חיפוש לפי קשרים:

```
MATCH (n)
WHERE (n)<-[:son]-({name: "Yoav"})
RETURN n
```

- כעת, נוכל להוסיף קשרים על קודקים **קיימים:**

```
MATCH (a:Person),(b:Pet)
WHERE a.name = 'Tuti' AND b.name = 'Lassy'
CREATE (a)-[r1:likes]->(b)
```

  - כלומר הוסף את הקשר (Tuti)-[r1:likes]->(lassy)

- **מחיקת כל** הקודקודים:

```
MATCH (n) detach delete n
```

- מחיקת קודקוד מסוים:

```
MATCH {GoldenFishy:Pet {....} DETACH DELETE GoldenFishy
```

- הוספת דור ($r1$):

```
Create(Eliezer:Person {name:'Eliezer', age:82,gender:"male"})
MATCH (a:Person),(b:Person) WHERE a.name = 'Eliezer' AND b.name = 'Koby'
CREATE (a)-[r1:father]->(b)
```

- **ממוצע** גילאים:

```
Match (n:Person) Return avg(n.age)
```

- **רשימת** כל הגילאים

```
Match (n:Person) Return collect(n.age)
```

- קבלת **מסלול** - קבלת כל המסלולים מאורך 2 עד 4 של קודקדים בקשר של $knows$ מ $Gadi$

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

- באותו מידה נוכל לבקש את **המסלול הקצר ביות**ר (מותר כל קשר מכל כיון) בין Tal ל $Gadi$

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

- קבלת שמות הסטודנטים שלומדים את **כל** הקורסים

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

- קבלת כל הסטודנטים שלומדים לפחות 4 קורסים

```
MATCH (s:student)-[:studies]->(c:course)
WITH s count(c) as num_courses WHERE num_courses <= 4
return s.name
```

6 *Java Stream*

עיבוד מידע יכול להתבצע, בdb כמו ה $stored\ procedures$ ב SQL או בקוד כמו $javastream$, תוך הסתמכות על:

- $lambda\ expressions$: פונקציות אנונימית. (נתמכת ב $java\ 8$ והלאה)
- $Syntax$: $(argumnet - list) - > \{body\}$
- דוגמאות:

1)

(x+y)->return x+y

2)

```
increment = x -> x + 1; System.out.println(increment.apply(5));
```

print: 6

- פונקציות ללא ארגומנט $(Callable)$

```
Callable<Integer> printAndReturn7 =
    () -> {int x = 7; System.out.println(x); return x; };
System.out.println(printAndReturn7.call());
```

6.1 *Stream*

- למעשה $class$ אבל לרוב לא נגדיר משתנה כזה
- זהו רצף של אלמנטים, שיוצרים מ $array/collection$
- $stream$ מחזיר $stream$
- ניתן "לבטל" את ה$stream$ ע"י $collect$
- פונקציות נפוצות: filter,map,sort,reduce,fint,match

דוגמאות

<u>מיון מערך</u>

```
List<String> myList = Arrays.asList("yes", "no", "hello", "goodbye", "none");
myList.stream().sorted().forEach((a)->System.out.println(a));
//or:
myList.stream().sorted().forEach(System.out::println);
```

<u>חיפוש</u>

$classroom$ הוא $hashMap$ מ$classrom$ (שם כיתה) ל $List < Studnt >$, <u>נרצה למצוא את הכיתות בהם יש סטודנטים שצריכים הנגשה (לנכים)</u>

```
List<String> roomsReqAccess = classrooms.entrySet().stream() // stream כל שורה הפוך ל
```

```
.filter(a->a.getValue().stream()
    .anyMatch(x->x.reqAccesability)) // סנן מכל שורה (ע"י stream) את מי שצריך הנגשה
.collect(Collectors.toList(c->c.getKey()));  // אסוף חזרה לרשימה וקח את שם הכיתה
```

- $anyMatch$ מחזיר $true$ ברגע שנמצא תלמיד "מתאים"
- הערה: התנאי ב$filter$ הוא **מה שישאר.**

6.2 $Parallel - Execution$

- רווח נוסף היא האפשרות לנצל את המעבד , נשתמש כאשר:
  - לא חשוב לנו סדר הפעולות בתהליך
  - עיבוד כל חלק לוקח זמן לא מבוטל או שיש הרבה חלקים
  - יש ברשותנו $threads$ זמינים
- איך? נחליף את $stream()$ ב $parallelStream()$

  ספירת כמה מספרים ראשוניים יש במערך:

```
numberList.parallelStream().filter(a ->naivePrimeTest(a)).count();
```

7 $Numpy$

Numpy Array

```
>>> a=np.array([6,7,8])
>>> a
array([6, 7, 8])
>>> a = np.array([1.0, 2, 3.4])
>>> a
array([ 1. , 2. , 3.4])
>>> a.dtype
dtype('float64')
>>> a = np.array([1, .40 ,"Hello"])
>>> a
array(['1', '0.4', 'Hello'],
    dtype='<U32')
>>> np.array(range(12))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11])
>>> a = np.arange(12).reshape(3,4)
```

```
>>> a
array([[ 0,  1,  2,  3,],
    [ 4, 5,  6,  7],
    [8, 9, 10, 11]])
>>> a.shape
(3, 4)
>>> a.ndim
2
>>> a.dtype
dtype('int32')
>>> a.size
12
>>> np.array(range(12)).reshape(3,3)
ValueError: total size of new array must be
unchanged
```

```
>>> mat = np.array([[3, 7, 5], [4, 1, 2]])
>>> mat
array([[3, 7, 5],
    [4, 1, 2]])
>>> mat[0]          (1
array([3, 7, 5])
>>> mat[0,:]
array([3, 7, 5])
>>> mat[:,1]        (2
array([7, 1])
>>> mat.shape
(2, 3)
>>> np.array([[3, 7, 5], [4, 1, 2, 2]])
array([[3, 7, 5], [4, 1, 2, 2]], dtype=object)
```

- כשמערבבים $types$, $numpy$ בוחר את המכנה המשותף הנמוך ביותר
- תומך במטריצות
- בדוגמה אנחנו עובדים עם מטריצה $dim(2 \times 3) \Leftarrow \begin{bmatrix} 3 & 7 & 5 \\ 4 & 1 & 2 \end{bmatrix}$
  1. $mat[0,:]$ מביא את כל השורה הראשונה
  2. $mat[:,1]$ מביא את כל העמודה השניה (אינדקס 1)

- פעולות אריתמטיות - כמו שאנחנו מכירים מאלגברה לינארית

```
>>> [2, 4, 5] + [7, 3, 2]
[2, 4, 5, 7, 3, 2]
>>> np.array([2, 4, 5]) + np.array([7, 3, 2])
array([9, 7, 7])
>>> [2, 4, 5] * 3
[2, 4, 5, 2, 4, 5, 2, 4, 5]
>>> np.array([2,4,5]) * 3
array([ 6, 12, 15])
>>> np.array([2, 4, 5]) * np.array([7, 3, 2])
array([14, 12, 10])
```

```
>>> np.array([[2, 4, 5], [4, 5, 2]]) * np.array([[6,3,4], [6,1,3]])
array([[12, 12, 20],
       [24,  5,  6]])
>>> np.dot(np.array([[2, 4, 5], [4, 5, 2]]), np.array([[6,3,4], [6,1,3]]))
ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
>>> np.dot(np.array([[2, 4, 5], [4, 5, 2]]), (np.array([[6,3,4], [6,1,3]]).T))
array([[44, 31],
       [47, 35]])
>>> np.array([[2, 4, 5], [4, 5, 2]]) @ (np.array([[6,3,4], [6,1,3]]).T) #Python 3
array([[44, 31],
       [47, 35]])
```

8 $Spark$

$RDD - Resilinet - Distributed - Dataset$

- הגדרה: אוסף נתונים גמיש ומבוזר
- אוסיף יחידות או פריטים מקיימים ביניהם חוסר תלות ולכן ניתן לחשיוב מקבילי (לדוגמה שורות בקובץ)
- ב$Spark$ אפשר לצור RDD מרישמה קיימת .
- ניתן לבצע שרשרת של טרנספורמציות שכ"א תניב RDD חדש. לבסוף נבצע איסוף למבנה נתונים פשוט ע"י $collect()$
- יתרונותיו:
  - מומלץ להריץ אותו על מספר מחשבים/מעבדים על מנת להשתמש בחישוב המקבילי
  - עבור $bigData = HDFS$
  - $highly - Scalable$

דוגמה:

<u>ספירת מילים בספר:</u>

```
>>> 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))
```

- flatMap maps every entry to a list, and then flattens the list back to an RDD (with possibly a longer length)

המשך - אותה שאילתא עם מיון בסדר יורד של הופעות המילים תביא לנו:

```
>>> word_counts = (text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a,b: a+b) \
    .map(lambda xy: (xy[1],xy[0])) \
    .sortByKey(False)\ #False is for descending order
    .map(lambda xy: (xy[1],xy[0])) \
    .collect())
>>> for word,count in word_counts[0:15]:
    print("the word: \"%s\" appears %d time(s)" %(word,count))
```

- ביצענו פעמיים map כדי למיין ע"פ **מספר** ההופעות - היות ויש רק פונקציה $srotByKey$ - לכן החלפנו בין ה key ל $value$ מיינו בסדר יורד - והחלפנו חזרה.
- על מנת לחסוך בזמן ריצה אפשר :

```
>>> text_file.cache()
```

$Bi-gram$

- בעיבוד שפות טבעיות, לרוב נתעניין בצמדי מילים (bi) מאשר מילים בודות על מנת לגלות את משמעות הקובץ.
- $tri-grams$ הוא יצירת שלשות, $n-gram$ - nיות..
- פונקציית zip - של פייתון - מחזירה רשימה של $tuples$ משתי רשימות:

```
>>> zip([1, 2, 3, 6], [10, 16, 23, 57]) = [(1,10), (2, 16), (3, 23), (6, 57)]
```

- לכן על מנת לקבל את כל הזוגות:

```
def bigram(line):
   words = line.split()
   return zip(words, words[1:])
```

- ובאופן דומה למילים בודדות:

```
>pairs = text_file.flatMap(bigram)
>>>count = pairs
    .map(lambda x: (x, 1))
    .reduceByKey(lambda a, b: a + b)
>>>print(count.map(lambda xy: (xy[1],xy[0])).sortByKey(False).map(lambda xy:
(xy[1],xy[0])).collect())
```

$Data-frame$

- $Data-frame$ דבר השווה לטבלאות של $spark$ ב $rmdb$, ניתן לצור מ:
  - $relational-database, RDD, CSV, XML/JSON$

– וניתן להשמתש בשאילתות sql לשם כך

Data-frames example

```
from pyspark.sql import Row
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
student_list = [(111, 'Chaya', 'Glass', 21), (222, 'Tal','Negev',28),
        (333, 'Gadi', 'Golan', 24), (444, 'Moti', 'Cohen', 23)]
student_rdd = sc.parallelize(student_list)
students_rows = student_rdd.map(lambda x:
        Row(id=int(x[0]),age=int(x[3]), firstName=x[1], lastName=x[2]))
df_students = sqlContext.createDataFrame(students_rows)
df_students.show()
```

```
+---+---------+---+--------+
|age|firstName| id|lastName|
+---+---------+---+--------+
| 21|    Chaya|111|   Glass|
| 28|      Tal|222|   Negev|
| 24|     Gadi|333|   Golan|
| 23|     Moti|444|   Cohen|
+---+---------+---+--------+
```

- ניתן גם לייצא ולצור $data-frame$ מהפורמטים שהזכרנו - דוגמאות במצגת

ניתן גם להוסיף "עמודה" לטבלה:

```
df_students2 =
    df_students.withColumn('age_plus_id', df_students.age + df_students.id)
df_students2.show()
```

דוגמה לשאילת sql ב $spark$ (פשוט לרשום את השאילתה כ$string$) :

```
df_students.registerTempTable("student_table") ending_with_n =
sqlContext.sql("SELECT firstName, lastName, id
    FROM student_table WHERE lastName LIKE '%n'")
ending_with_n.show
```

- דוגמה במצגת למימוש $map-reduce$ ב$mongoDb$ למול RDD ו $data-frame$ ב $Spark$

RDD v. Data Frames

RDD	Data Frames
"How" to do	"What" to do
Unstructured Data	Structured and semi-structured data
Less optimal and efficient	optimization and performance benefits available with DataFrames

עד כה עסקנו ב$data$ קיימת, נרצה לפתח כלים ל $Machine\ learning$

Train/Test Spilt

- ראשית אנחנו צריכים לאמן את ה $classifier$ שלנו. לשם כך אנחנו צריכים $data$ שאנחנו יודעים בה הכל
- לכן נפצל את ה$data$שקיימת ברשותנו ל $train$, כדי לפתח את הכלי, ו $test$, $data$ שעליה נבדוק אם הכלי שפתחנו עובד (בשלב השלישי נעבוד על $data$ בלי אפשרות לטסטים)

9 Naïve Bayes

חוק בייס : $P(B)P(A|B) = P(B|A)P(A)$ <u>הוכחה + שקילות:</u>

$$P(B)P(A|B) \overset{\text{def}}{=} \frac{P(B)P(A\cap B)}{P(B)} = P(A\cap B) = \frac{P(A)P(A\cap B)}{P(A)} \overset{\text{def}}{=} P(B|A)P(A) \iff \frac{P(B)P(A|B)}{P(B|A)} = P(A)$$

- נניח שיש לנו $data\ x_1, x_2, ..., x_m$ (הודעות למשל)
- נסמן את ה$Classifier$ ב $y_1, ..y_s$
- נרצה לנתח ולהבין מה $class$ של x_m (את השאר אנחנו יודעים) אז אנחנו בעצם מחפשים את $P(y=k|x_m)$, לכל $k \in [1,s]$ וניקח את ההסתברות הגבוהה ביותר $(argmax)$
  - הערה: $argmax$ ־ הגורם שבעקבותיו התוצאה מקסימלית
- על חוק בייס: ניתן לחשב : $P(y=k|x_m) = \frac{P(y=k,x_m)}{p(x_m)}$
- בשיטה זו מניחים ש**המאורעות בלתי תלוים** ־ בדוגמת ההודעה המילים אינן תלויות אחת בשניה (במציאות זה לא ככה)
  - על פי הגדרה חלוקת x_t למאורעות זרים תוביל ל:

$$p(x_t, y=j) = p(y=j)\, p(x_{t1}|y=j)\, p(x_{t2}|y=j, x_{t_1})\, p(x_{t3}|y=j, x_{t_1}, x_{t_2}) ....$$

  - מההנחה של $naive\,bayes$ המאורעות בת"ל ,ולכן מתקיים ש:

$$\overset{def}{=} p(y=j)\, p(x_{t1}|y=j)\, p(x_{t2}|y=j)\, p(x_{t3}|y=j)$$

$$\iff p(y=j) \prod_{i=1}^{m} p(x_i|y=j)$$

- כעת נחזור לחישוב שלנו:

$$P(y=k|x_m) = \frac{P(y=k,x_m)}{p(x_m)} = \frac{p(y=k)\prod_{i=1}^{m} p(x_i|y=k)}{p(x_m)}$$

  - הערה: אנחנו מחפשים מקסימום, ומבצעים חישוב על כל הערכים האפשרים לy , לכן החלוקה ב $p(x_m)$ לא משפיע על החישוב וניתן להזניח אותו

- סה"כ:

$$y^* = \underset{k=1,\dots,K}{\text{argmax}}\, p(y=k) \prod_{i=1}^{n} p(x_i|y=k)$$

- תוספת: באופן כללי הראנו שאנחנו רוצים למצוא למקסם את הסבירות של $\prod_{i=1}^{n} p(x_i, y_i)$

- מכאן נובע שאם נחלק את מספר המופעים במספר ההודעות ב $class$,(ע"י משתנה מקרי) נקבל מדד יותר טוב בשבילנו, כלומר:

$$y^* = \underset{k=1,\dots,K}{\text{argmax}} p\left(y=k\right) \prod_{i=1}^{n} \frac{\text{num of msg in class k with word i}}{\text{num of msg in class k}}$$

$$y^* = \underset{k=1,\dots,K}{\text{argmax}} p\left(y=k\right) \prod_{i=1}^{n} \frac{\sum_{i=1}^{m} 1\{x_{ti}=1 \wedge y_y=k\}}{\sum_{i=1}^{m} 1\{y=k\}}$$

- שורה תחתונה צריך לחשב:
  - P_{tot} = כמה משפטים סה"כ
  - P_k = כמה משפטים סה"כ במחלקה ה K
  - P_{k_i} = כמה משפטים במחלקה k מכילים את המילה i ב x

$$y^* = \underset{k=1,\dots,K}{\text{argmax}} \frac{p_k}{p_{tot}} \prod_{i=1}^{n} \frac{p_{k_i}}{p_k}$$

- חישובים כאלו קלאסיים לחישוב מקבילי

דוגמה חיזוי $Spam$

נניח שה$data$ שברשתונו היא :

Spam:	**Real (Non-Spam):**	**Test**
Buy it, pay later! Click me!	Will See you later.	Are you paying too much? Click now!
You Won 10000 Dollars! Click here!	Will you want to meet later?	
	I'm waiting for you.	

ראשיתת נכניס את ה $train$ לטבלה:

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

כעת נרצה לחשב על פי הנוסחה

$$y^* = \underset{k=1,\dots,K}{\text{argmax}} \frac{p_k}{p_{tot}} \prod_{i=1}^{n} \frac{p_{k_i}}{p_k}$$

נשים לב שיש מכפלות , ולכן האפסים בעייתים, כי הם יאפסו את כל הΠ , לכן נשתמש ב $Laplace's\ Smoothing$ ופשוט נוסיף לכל אחד 1 :

	Examples	**are**	**you**	**paying**	**too**	**much**	**?**	**click**	**now**	**!**
Spam	3	1	2	2	1	1	1	3	1	3
Real	4	1	3	1	1	1	2	1	1	1

ניתן לחשוב על זה כעל הוספת המשפטים משפט ריק ומשפט עם המילים שאנחנו מחפשים (לכן מתווסף 2 לסה"כ של כל $class$)

$$y^* = \underset{k=1,\dots,K}{\text{argmax}} \frac{p_k+1}{p_{tot}+1\cdot K} \prod_{i=1}^{n} \frac{p_{k_i+1}}{p_k+1\cdot K}$$

(אצלנו $K = 2$)

כעת:

$$p(y=0) = \underbrace{\frac{3}{7}}_{\text{prob' to in spam}} \cdot \underbrace{\frac{1}{4}}_{\text{sum of sentences in class}} \quad \tfrac{2}{4}\tfrac{2}{4}\tfrac{1}{4}\tfrac{1}{4}\tfrac{1}{4}\tfrac{3}{4}\tfrac{1}{4}\tfrac{3}{4} = 5.87 \cdot 10^{-5}$$

$$+ 2 = \text{lap' smo'}$$

$$p(y=1) = \underbrace{\frac{4}{7}}_{\text{prob' to in real}} \quad \tfrac{1}{5}\tfrac{3}{5}\tfrac{1}{5}\tfrac{1}{5}\tfrac{1}{5}\tfrac{2}{5}\tfrac{1}{5}\tfrac{1}{5}\tfrac{1}{5} = 2.35 \cdot 10^{-6}$$

המקיסמום כאשר $y = 0$, לכן זוהי הודעה שנחשיב כ $Spam$

- לרוב ניקח log כדי לעבוד עם מספרים נוחים

9.0.1 מדד סיווג (פונ' $F - score$)

דוגמה

נניח ויש לנו תוכנה שחוזה עם אנשים בריאים או חולים, ומתקיים:

קבע ש:	**בריא**	**חולה**
בריא באמת	צודק	5% טעות
חולה באמת	0% טעות	צודק

מבחינתנו זה בעייתי מאוד, כיון שלטעות באבחנה על אנשים חולים תגרום לכך שלא יקבלו טיפול - בעוד שטעות על בריאים בסה"כ תוביל לבדיקות מיותרות.

לכן נגדיר:

מהו $classifier$ טוב?

- $Accuracy$ - נכונות. כמות ההצלחות ביחס לכל החיזויים. $\frac{\text{true}}{\text{all}}$
- $Racll$: $\frac{\text{hits from healthy}}{\text{how many healthy}}$ - כמה צדקנו בתוך הקגטוריה (קבענו ש $\cap$ באמת הם <u>מתוך</u> כל הבאמת)
- $Precision$: $\frac{\text{hits from healthy}}{\text{how many we determine as healthy}}$, כמה אנחנו מדויקים בקטגוריה (קבענו ש $\cap$ באמת הם <u>מתוך</u> כל הקבענו ש)
- ואת הפונקציה $F_{score} = \frac{2 \cdot P \cdot R}{P+R}$

<u>תרגיל: נתון מדגם של 100אנשים , כאשר 60 בריאים, ו40 חולים</u>

- ע"פ C_1 :
  - מהבריאים הוא קבע ש: 30 חולים, 30 בריאים
  - מחולים הוא קבע ש: 35 חולים , 5 בריאים
- ע"פ C_2:
  - מהבריאים הוא קבע ש: 50 בריאים, 10 חולים
  - מהחולים הוא קבע ש: 25 חולים, 15 בריאים
- למי יש $score$ גבוה יותר

נחשב את $f_{scroe}(C_1)$:

$$R_1 = \frac{30}{60}, P_1 = \frac{30}{35} \Rightarrow F_1 = \frac{2 \cdot \frac{1}{2} \cdot \frac{39}{35}}{\frac{1}{2} + \frac{30}{35}} = \frac{12}{19}$$

נחשב את $f_{scroe}(C_2)$:

$$R_2 = \frac{50}{60}, P_2 = \frac{50}{65} = F_1 = \frac{2 \cdot \frac{5}{6} \cdot \frac{50}{65}}{\frac{5}{6} + \frac{50}{65}} = \frac{4}{5}$$

10 *Linear Regression*

<u>למידה</u>: ע"פ (x_i, y_i) :

- x_i - הוא נתון בודד או מערך של פיצ'רים
- y_i - הוא הערך המוצמד ל x_i

<u>מבחן</u>: מקבל x צריך לחזות את y (אצלנו תמיד y ערך בודד)

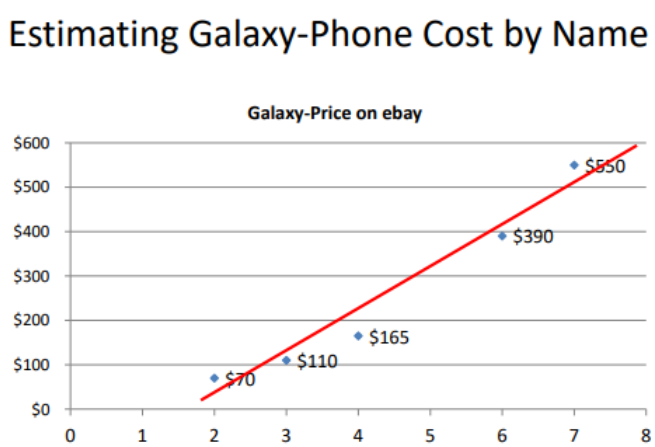
איך נחזה?

נגדיר $Loss - f$: פונקציה המודדת את הטעות הממוצעת של המכונה (בהנחה ויש m ערכים ל i)

$$\frac{1}{m} \sum_{i=1}^{m} |y - y_i|$$

באופן כללי אם נמצא חיזוי כלשהו ל y , כל שנותר לחפש זה את המינימום על f

באופן פשוט נגדיר פונקציה לינארית $y = wx + b$ לחיזוי נאיבי



לכן:

$$f(w,b) = \frac{1}{m} \sum_{i=1}^{m} |wx_i + b - y_i| \Rightarrow J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (wx_i + b - y_i)^2$$

- על מנת למצוא מינמום min צריך פונקציה גזירה וערך מוחלט אינה כזאת, לכן העלנו בריבוע שזו גם פונקציה שתמיד חיובית
- דוקא בריבוע ולא חזקה גבוה יותר, כי חזקה גבוה מגדילה מאוד את הטעויות **הגדולות** , ומכאן שמקטינה את הערך של הטעויות הקטנות ודוקא אותם אנחנו רוצים לתפוס
- איך נדע שנקודת הקיצון היא מינימום? זו פרבולה צוחקת....
- חלוקה ב $2m$ על מנת לצמצם את את ה2 בגזירה

אלגוריתם (גרדינט)

- בחירה רנדומלית ל w, b
- כעת נחשב את הגרדינט (גזירה על פי כל פרמטר:

$$J'_w = \frac{1}{m}\sum_{i=1}^{m} x_i\left(wx_i - y_i\right)$$
$$J'_b = \frac{1}{m}\sum_{i=0}^{n} 1\cdot\left(h(x_i) - y_i\right)$$

- בחירה α ($learning\,rate$) למשל 0.01
  - עדכן את $w = w - \alpha\frac{1}{m}\sum_{i=0}^{n} x_i\left(h(x_i) - y_i\right)$
  - עדכן את $b = b - \alpha\frac{1}{m}\sum_{i=0}^{n} 1\cdot\left(h(x_i) - y_i\right)$

מה קורה כשיש כמה פיצ'רים? כלומר $x_i = (x_{i1}, x_{i2}, ..., x_{is})$ נגדיר את $w = (w_1, x_2, ..., x_f)$, לכן:

- עבור w: כל מכפלה של $w\cdot x_i$ היא בעצם מכפלה פנימית, ולבסוף (בגלל ה שx_i הוא וקטור) נעדכן את הוקטור w
- עבור b: נקבל מספר ונעדכן בהתאם

דרך נוספת ל"דייק" את החיזוי הוא על ידי הוספת חזקות כפיצרים, כלומר להגדיר:

$$h(x) = \left(w_1x + x_2x^2 + b\right)$$

- בכך נצור עקומה בין הנקודות ולא קו ישר
- בדרך זו ניתן להוסיף עוד עוד חזקות ע"מ לדייק את החיזוי
- יש חשש ל$over - fitting$ ולכן צריך להזהר

11 *Logistic Regression*

כאשר אנחנו רוצים חיזוי של כן ולא, נציב ע"י $h(x) = \frac{1}{1+e^{-(Wx+b)}}$

המשמעות שלה היא החישוב $P(y=1|x)$

כעת ה $Loss - f$:

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

האלגורית (אותו דבר ה h שונה):

- בחירה α ($learning\,rate$) למשל 0.01
  - עדכן את $w = w - \alpha\frac{1}{m}\sum_{i=0}^{n} x_i\left(h(x_i) - y_i\right)$
  - עדכן את $b = b - \alpha\frac{1}{m}\sum_{i=0}^{n} 1\cdot\left(h(x_i) - y_i\right)$