מצגות

חסר:

1-IntroductionFile
2-SQLFile
3-AdvSQLandConnectorFile

4-NormalizationFile

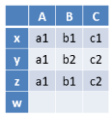
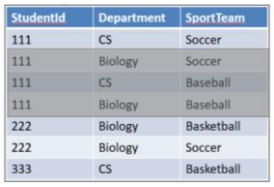
מהו נרמול טבאלות ולמה זה טוב?
הרנמול הוא תהליך **שיפור יעילות** בסיס הנתונים שלנו.
חוקי הנרמון הם ארבעת חוקים הגיוניים ולוגיים שמאפשרים לנו לתכנן את בסיס הנתונים שלנו בצורה הגיונית, למנוע בעיתות בעתיד, לצמצם את שטח הדיסק שבסיס הנתונים תופס ולהביא את בסיס הנתונים למצב שיהיה קל לעבוד איתו בעזרת SQL

מושגים:

תכונה, מקביל לעמודה בטבלה	Attribute
נאמר על תכונה B שהיא תלויה בתכונה אחרת A אם קיימת פונקצייה (יחס תלות) כך ש: $A \Rightarrow B$ **אפשר להסתכל על זה כאם אנחנו יודעים את A האם אנחנו יודעים את B ?** **דוגמא:** A=Student ID, B=Student Name אם $B \subseteq A$ אזי בהכרח $A \Rightarrow B$	Dependencies
<u>מפתח</u> אוסף **<u>מינמלי</u>** של שדות אשר גורר את **<u>כל</u>** השורה, מינמלי באופן שבו אם נסיר שדה אחד מהקבוצה, כבר לא נוכל לגרור את כל השדות. **דוגמא**: עבור מסד נתונים אשר מכיל שמות, גיל וכו' אזי $\{Student\ ID\}$ היא ה-Candidate key	Candidate keys

כל טבלה יכולה להכיל יותר **ממפתח** בודד	
Prime – חלק ממפתח Candidate keys Non-Prime – לא חלק ממפתח	Prime / Non-Prime
כל קבוצה של שדות אשר גוררים את כל השורה, לדוגמא כל צירוף של מפתחות ביחד עם עוד שדה, ניתן להתייחס להגדרה זאת כמפתח, רק **<u>שאין</u>** דרישה למינמאלי.	Super-Key

חוקי נרמול

להוסיף שורות	**1NF**	טבלה מנורנלת ברמה הראשונה אם ורק אם כל רכיב בה הוא **אוטומטי**	כל רכיב בה הוא **אטומי**
לפצל טבלאות נקח את הקשר שיצר את הבעיה, נכניס לטבלה חדשה ובטלה המקורית נמחק את הNon-Prime הבעייתים	**2NF**	מכיל את **1NF** וכל שדה שהוא לא Prime, לא צריך להיות תלוי בקבוצה **<u>חלקית</u>** של Candidate keys.	$X \Rightarrow Y$ אם Y לא Prime אז **אסור** ש X יהיה תת קבוצה **<u>חלקית</u>** של מפתח
לפצל טבלאות	**3NF**	מכיל את **2NF** וכל שדה לא Prime לא יכול להיות תלוי במשהו שהוא לא Super-Key	$X \Rightarrow Y$ אם Y לא Prime אז X חייב להיות **Super-Key**
לפצל טבלאות	**BCNF (3.5 NF)**	מכיל את **3NF** וכל שדה חייב להיות להגרר ע"י **<u>בלבד</u>** Super-Key	$X \Rightarrow Y$ עבור כל גרירה, X הוא **Super-Key**
להשלים	**4NF**	 	אם מתוך חלק מהשורות ניתן להסיק חלק אחר של השורות

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

StudentId	Department	SportTeam
111	CS	Soccer
111	Biology	Soccer
111	CS	Baseball
111	Biology	Baseball
222	Biology	Basketball
222	Biology	Soccer
333	CS	Basketball

<u>הערה:</u>

- אם בטבלה אין Non-Prime אז מקבלים את NF2 ו-3NF מתנה, היות ותמיד נסתכל על רלצייה $X \Rightarrow Y$ כאשר Y הוא Non-Prime.
- אם יש רק Candidate key יחיד והטבלה היא 3NF אז היא גם 3.5NF.
- אם תכונה מופיעה **<u>רק בצד ימין</u>** של הרלצייה אז היא בוודאות Non-Prime
- אם תכונה מופיעה **<u>רק בצד שמאל</u>** של הרלצייה אז היא בוודאות Prime

שלב ראשון בפתרון בעיות זה לרשום את כל הקשרים שקיימים בטבלה.

5-XmlAndJsonFile

XML הוא תקן לייצוג נתונים במחשבים. שימוש ב־XML מקל על החלפת נתונים בין מערכות שונות שפועלות על גבי תשתיות שונות. תקן ה-XML לא מגדיר איזה מידע יוצג אלא מגדיר כיצד לייצג מידע באופן כללי. תקן XML שייך למשפחת **שפות הסימון.**

מסמך XML נקרא **Well-formed** אם מתקיימים התנאים הבאים:

1. עבור כל תגית פתיחה יש תגית סגירה
2. יש רק תגית ראשית אחת (Root)
3. אין שני אלמנטים עם אותו ערך של תכונה
4. סדר סגירת התגיות חשוב
5. אין אלמנט עם יותר מתכונה אחת עם אותו שם.

דוגמא לקובץ XML :

Attribute - תכונה

<Element /> Empty Element

```
<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>
```

סימונים מיוחדים:

<	<
>	>
&	&
'	'
"	"

- אין קשר בין HTML לבין XML .

Xpath מאפשר לנו לבצע שאילתות על מסמך XML :
אתר: http://www.xpathtester.com/xpath

```
<?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>
</University>
```

/University/*

מחזיר את כלל הסטודנטים

/University/Student[1]/@degree

Attribute degree : PhD

/University/Student[./age = 21]/@degree

Attribute degree : PhD

University/Student[.//City = 'Ariel']/

```
Node Student :
      Chaya
      Glass
      111
      21

          Hatamr 5
          Ariel
          40792
```

/University/Student[(age = 21) | (FirstName = "Chaya")]/@degree

Attribute degree : PhD

הערה: Xpath מתחיל את האינדקסים שלו מ-1 ולא מ-0.

Example
//book | //cd
6 + 4
6 - 4
6 * 4
8 div 4
price=9.80
price!=9.80
price<9.80
price<=9.80
price>9.80
price>=9.80
price=9.80 or price=9.70
price>9.00 and price<9.90
5 mod 2

ניתן בעזרת xPath לבצע תנאים כמו > וכו'.

XQuery

קלט:

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

פלט:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

קלט:

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

פלט:

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

FLWOR (pronounced "flower") is an acronym for "For, Let, Where, Order by, Return".

- **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)

```
Let $x := avg(/Bookstore/Book/price)
Order by xs:int{$x/2};
```

קלט:

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

פלט:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

קלט:

```
for $x in /University/Student
  let $avg_age := avg(/University/Student/age)
  where $x/age < $avg_age + 1
  return $x/id
```

פלט:

```
<id>111</id>
```

```
<?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>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

וידוא תקינות של מסמך XML:

XSD:

הוא תקן המשמש להגדרת סכמה (מבנה) של מסמכים הכתובים בשפת XML. XSD הוא תקן של ארגון W3C שפותח במקור על ידי חברת מיקרוסופט.

דוגמא:

```
<?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"/>

```
<?xml version="1.0"?>
<University>
  <Student degree="PhD">
    <FirstName>Chaya</FirstName>
    <LastName>Glass</LastName>
    <Address>
      <Street>Hatamr 5</Street>
      <City>Ariel</City>
      <Zip>40792</Zip>
    </Address>
    <age>21</age>
  </Student>
  <Student>
    <FirstName>Tom</FirstName>
    <LastName>Glow</LastName>
    <Address>
      <Street>Mishmar 5</Street>
      <City>Ariel</City>
    </Address>
  </Student>
</University>
```

Not valid.
1. Invalid content was found starting with element 'Address'. One of '{age}' is expected.
2. Invalid content was found starting with element 'Zip'. No child element is expected at this point.
3. Invalid content was found starting with element 'Address'. One of '{age}' is expected.

על מנת לתקן:

נוסיף בסוף השורה של ה-age את minOccurs=0 ונוסיף שורה של Zip גם כן אופציונלית, על מנת **לתקן את הסדר** במקום <xs:sequence> נשנה ל <xs:all>

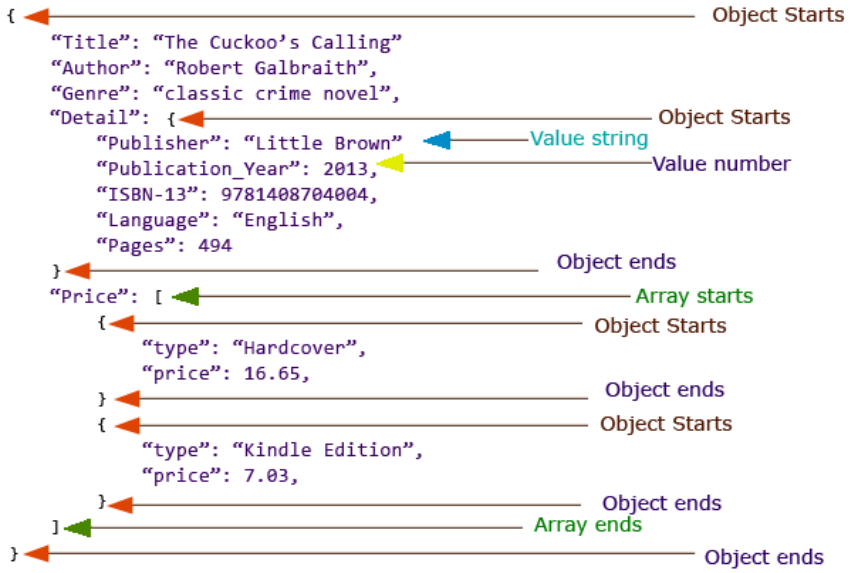
על מנת לאפשר String **לא** ריק:

```
<xs:simpleType name="NonEmptyString">
    <xs:restriction base="xs:string">
        <xs:minLength value="1" />
        <xs:pattern value=".*[^\s].*" />
    </xs:restriction>
</xs:simpleType>
```

\s is whitespace.
^\s means not whitespace

Using it:
<xs:element name="lastName" type="NonEmptyString" />

:Json



JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

The biggest difference is:

XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON is Better Than XML

XML is much more difficult to parse than JSON.
JSON is parsed into a ready-to-use JavaScript object.

6-NoSQLFile

NoSQL הוא קטגוריה חדשה יחסית של בסיסי נתונים, אשר נותנים פתרון אחסון וגישה למידע שאינו ממודל במבנה טבלאי יחסי אשר נפוץ בבסיסי נתונים יחסיים.

מבנה המידע שונה ממערכות בסיסי נתונים יחסיים, ולכן ישנן פעולות שמהירות יותר ב-NoSQL וישנן פעולות שמהירות יותר בבסיס נתונים יחסי. בסיסי נתונים מסוג NoSQL הופכים נפוצים יותר במערכות Big Data וכן במערכות זמן אמת.

מערכות אלו נקראות לעיתים "Not Only SQL", כדי להדגיש שלחלקן תמיכה בשפת השאילתות SQL.

Type	Example
Key-Value Store	redis, riak
Wide Column Store	HBASE, cassandra
Document Store	mongoDB, CouchDB relax
Graph Store	Neo4j, InfiniteGraph The Distributed Graph Database
RDB	Apache Jena
Search-Engine DB	Elastic Search, Apache Solr

Big Data - הוא מונח המתייחס למאגר מידע הכולל נתונים מבוזרים, שאינם מאורגנים לפי שיטה כלשהי, שמגיעים ממקורות רבים, בכמויות גדולות, בפורמטים מגוונים, ובאיכויות שונות.

יתרונות של NoSql על מסד נתונים רלציונאלי:

1. מותאם להתעסקות עם Big Data
2. גמיש, ללא הסתמכות על תבנית מסויימת.
3. מבנה יכול להשתנות עם הזמן.
4. יותר זול לתחזוק.

תזכורת MySQL :

במודל זה בסיס הנתונים בנוי מטבלאות, כאשר כל טבלה מכילה מידע על ישות מסוימת (לדוגמה, לקוחות במערכת בנקאית).
בסיסי נתונים נפוצים בשימוש :

- קוד פתוח: MySQL, SQLite, PostgreSQL
- בתשלום: SQL Server & Oracle

ACID

המונח ACID הוא ראשי תיבות של Atomicity, Consistency, Isolation, ו-Durability. תרגום המונחים לעברית הוא **אַטוֹמִיוּת, עִקְבִיוּת, בִּידוּד וְעֲמִידוּת**. תכונות אלה הן אבן הפינה של מסדי נתונים ומערכות לניהול תנועות, ובלעדיהן לא ניתן להבטיח את שלמות הנתונים במערכות אלה.
בפועל, תכונות ה-ACID נאכפות במידה רופפת יותר כדי לשפר את ביצועי המערכת.

- **Atomicity**
  כל פעולה מתבצעת במלואה או לא מתבצעת בכלל
- **Consistency**
  הבסיס נתונים תמיד נשאר עקבי, אם פעולה לא חוקית (השמה של מחרוזת במקום מספר, השמה של מספר שלא זהה לחוק שהוגדר עבור עמודה) מתבצעת, אז בסיס נתונים לא מאפשר ביצוע של אותה הפעולה.
- **Isolation**
  בסיס הנתונים מאפשר לבצע פעולות רבות במקביל, כל עוד התוצאה זהה לביצוע הפעולות באופן טורי.

- **Durability**
  פעולה שמתבצעת תמיד נשארת במסד נתונים, אפילו אם היה תקלה באמצע, אם זה הפסקת חשמל וכו'. פעולות לא הולכות לאיבוד.

SQL או Seek Well
שפה לביצוע פעולות של בסיס נתונים , פקודות מבוססות על Relational Algebra.
שפה שהינה Insenstive language, כלומר ניתן להשתמש גם באותיות גדולות וגם באותיות קטנות.

תו מחיר

1. בד"כ לא יכול לספק את כל **ACID**
2. אין סטנדארט, כל אחד יכול לממש אחרת.
3. פחות טוב עבור מידע רלציואני.

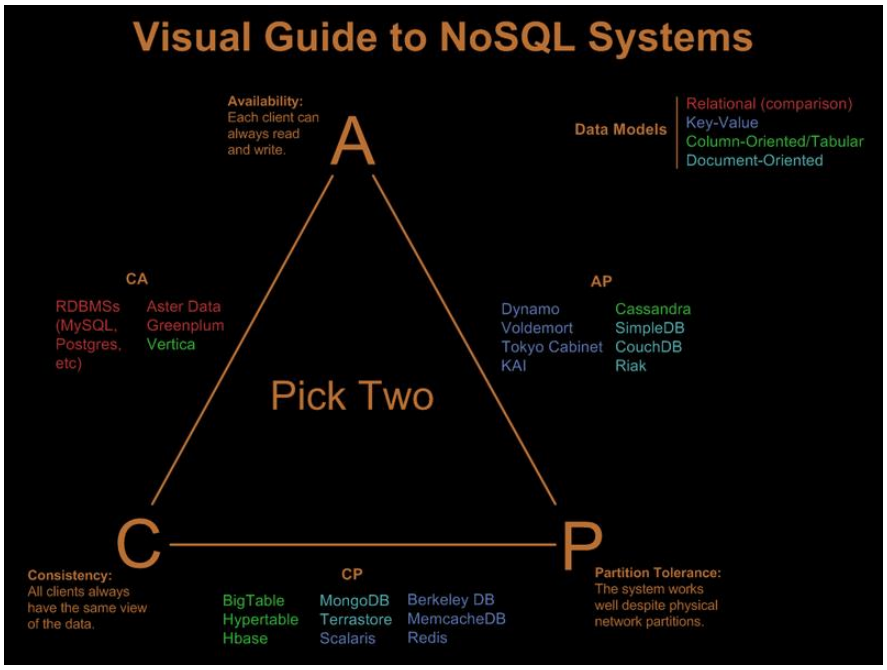
CAP Theorem :

1. Consistency
כל פעם שאנחנו מצביעים איזשהי קריאה אנחנו תמיד מקבלים את המידע הכי עדכני או שנקבל שגיאה

2. Availability
כל פעם שאנחנו מצביעים קריאה אנחנו נקבל איזשהי תשובה, בין אם היא מעדוכנת ובין אם לא

3. Partition tolerance
המערכת יכולה להתנהל גם על כמה מחשבים שונים, כשיש רשת בינהם שהיא לא מאבוטחת ולא מובטח לנו עליה שום דבר. יכול להיות שיש הודעות שיגיעו באיחור וכו', שהמערכת תעבוד כשהיא מבוזרת.

שפות של NoSQL יכולות לקיים רק 2 מהדברים בו זמנית.



ב-NoSQL נקבל BASE

1. Basically Avaliable
המידע בד"כ זמין.
1. Soft State
המצב יכול להשתנות גם ללא עדכונים בגלל עדכונים ישנים.
1. Eventual consistency
אם ניתן מספיק זמן, בסוף המידע יהיה עקבי.

Relational Database	NoSql Database
Supports powerful query language.	Supports very simple query language.
It has a fixed schema.	No fixed schema.
Follows ACID (Atomicity, Consistency, Isolation, and Durability).	It is only "eventually consistent".

Radis
Redis מבוסס על שליפת הנתונים על בסיס "**מפתחות**" המאפשר שליפת נתונים באופן מהיר ביותר מתוך מאגרי מידע ענקיים (Big data). תומך במידע **זמני**. מאפשר לשלוף **רק** לפי המפתח, בלי אפשרות לשאול על הערך.

SET, GET, DEL

- SET hello "world"
OK
- GET hello
"world"
- GET "world"
(nil)
- SET name1 Yossi
- GET name1
"Yossi"
- 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>"
- DEL user:1001

We usually store the id as part of the key, so we can fetch it easily

The value can be anything (e.g. XML)

בתוך Hashmap הגדול אפשר לעשות עוד Hashmap אבל **לא** עוד רמה. אין אפשרות לעשות עוד Hashmap

Hashes(HSET, HGET, HMSET, HGETALL

- HSET user:302 first_name "Tamar"
- HSET user:302 last_name "Cohen"
- HGET user:302 first_name
"Tamar"
- HMSET user:302 degree 3 gender "female"
- HGET user:302 degree
"3"
- HGETALL user:302
1) "first_name"
2) "Tamar"
3) "last_name"
4) "Cohen"
5) "degree"
6) "3"
7) "gender"
8) "female"

HMSET: for setting multiple attributes in a single command

It looks as if HGETALL returns a list, but it really returns a set

INCR and INCRBY

- SET bottles 5
- INCR bottles
(integer) 6
- INCRBY bottles -3
(integer) 3
- INCR name1
(error) ERR value is not an integer or out of range

INCR x is equivalent to x++ (in JAVA), and is guaranteed to be atomic

INCRBY x y is equivalent to x+=y (in JAVA).

הערך (Value) יכול להיות קצת יותר מורכב, הוא יכול להיות בעצמו **רשימה** או **HashMap**

Lists (RPUSH, LPUSH, LRANGE)

- RPUSH user:571:items "chair"
- RPUSH user:571:items "table"
- RPUSH user:571:items "water"
- LPUSH user:571:items "cup"
- LRANGE user:571:items 1 2
1) "chair"
2) "table"
- LRANGE user:571:items 0 -1
1) "cup"
2) "chair"
3) "table"
4) "water"
- RPUSH user:573:items "bed" "chair" "table" "can"

RPUSH (Right Push) puts the new item last (on the very right)

LPUSH (Left Push) puts the new item first (on the very left)

LRANGE key x y
Returns items from x to y

-1 means until the end of the list

You can provide the full list in a single command

הספירה מתחילה מ0.

Sets and Sorted Sets

Redis also supports sets:
- SADD x y (add item y to set x)
- SREM x y (remove item y from set x)
- SISMEMBER x y (is y a member of x)
- SUNIOUN x q (returns the union of sets x and q)

And Sorted sets:
- 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)

EXPIRE

- The EXPIRE commands sets a time in seconds to which a value will expire.

➢ EXPIRE user:302 100
➢ TTL user:302
(integer) 92

Get KEYS with pattern

➢ SET user:1000 "Adam"
➢ SET user:1001 "Tammy"
➢ SET user:1002 "EVE"
➢ SET user:1010 "APPLE"
➢ RPUSH user:1003:items "chair"

➢ KEYS user:100?
1) "user:1000"
2) "user:1001"
3) "user:1002"

➢ KEYS user:*
1) "user:1000"
2) "user:1010"
3) "user:1003:items"
4) "user:1001"
5) "user:1002"

Cassandra

מערכת קסנדרה היא מערכת NoSQL. היא מספקת ממשק פשוט יותר של חיפוש ערך לפי מפתח.
מאפשר שליפה מהירה של המידע, **מאורגן כמו מערכת קבצים**, מאפשר שאילתות כמו מסד נתונים רלאציוני.
כל רשומה (שורה) עומדת בפני עצמה. (לכל שורה יש את העמודות שלה)

ליצור בסיס נתונים:
יותר 2 כפליות שנשמרות באותה מרכז מידע.

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

אם במקום SimpleStrategy היינו רושמים NetworkTopolgyStrategy אז זה היה נפרס על פני כמה מרכזי מידע.

ליצור טבלה:

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

תנאי:

ה-id חייב להיות **Primary Key**

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

הכנסה:

```
CREATE TABLE grades (studentId INT, course TEXT, grade FLOAT, PRIMARY KEY(studentId,
course));
INSERT INTO grades(studentId, course, grade) values(111, 'into to intro', 95);
INSERT INTO grades(studentId, course, grade) values(111, 'calculus', 78);
INSERT INTO grades(studentId, course, grade) values(111, 'Algebra', 81);
INSERT INTO grades(studentId, course, grade) values(222, 'Algebra', 51);
INSERT INTO grades(studentId, course, grade) values(222, 'Algebra', 61);
```

שליפה:

```
SELECT grade from grades WHERE studentid=111 AND course > 'b';
```

Partion Key

- The partition key can include more than a single key.
- When data is inserted into the cluster, the first step is to apply a hash function to the partition key. The output is used to determine what node (and replicas) will get the data.
- The whole partition key must be specified (with equality sign) every query! (unless we request the whole table)
- This is because Cassandra must know where to find the requested data.
- E.g. CREATE TABLE grades (studentId INT, course TEXT, grade FLOAT, passed BIT, PRIMARY KEY((studentId, course), grade));
- SELECT * FROM grades WHERE studentId=111 AND course=20 ✔
- SELECT * FROM grades WHERE studentId=11 ✘

Clustering Key

סדר על מפחות:

- Table T with Primary keys:

  x as the partition key and y, z as clustering keys

SELECT * FROM T WHERE x = 5; ✔

SELECT * FROM T WHERE y = 5; ✘ — לא ניתן לגשת ל clustering key ללא partion key

SELECT * FROM T WHERE x = 5 AND z = 7; ✘ — לא לפי הסדר

SELECT * FROM T WHERE x = 5 AND y < 3 AND z > 0; ✘ — השאילתות זה כמו ספריות, צריך לתת בדיוק ערך, לדוגמא y=3

SELECT * FROM T WHERE x = 5 AND y < 6 AND y > 0; ✔ — אין חשיבות לסדר במידה והשאילתה עצמה נכונה

SELECT * FROM T WHERE x = 5 AND z = 4 AND y > 0; ✘ — Y חייב להיות שווה כי הוא מכיל את z

SELECT * FROM T WHERE x < 10; ✘ — חייב להית שווה

SELECT * FROM T WHERE x =2 AND z <4 AND y = 3; ✔

SELECT * FROM T WHERE y = 2 AND z < 8; ✘ — חייב להיות parrtion key (=)

:MongoDB

בסיס הנתונים המוביל בעולם בקטגוריית NoSQL, ובין חמשת המובילים בכל הקטגוריות.
מאחסן את הנתונים גם כן בKey-Value, מאפשר שאילתות גם על הנתונים בValue ומאפשר לעשות הרבה מניפולציות על המידע – Map Reduce .

יצירת מסד נתונים

use MyDB (Database Name)

מחיקת מסד נתונים

db.dropDatabase()

יצירת Collection (שורש שמכיל את כל הרשומות)

db.createCollection("movies")
db.movies.drop()

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

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a collection fixed size collecction that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexID	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection. 37

יצירת רשומה Document

Db.movies.insert(JSON STYLE DOCUMENT)

```
db.movies.insert({"movieName": "The Zookeepers Wife", "id": "111",
                  "desc": "bla blab la bla",  "stage manager":"Nici Karu",
                  "genre" : "drama",
                  "production": {
                     "script": "someone",
                     "music": "somebody",
                     "Photography": "din charles"}
                  })
```

```
db.movies.insert( [ {"movieName": " The SpongeBob Movie", "id": "222",     "desc": "bla blab la bla", "stage manager":"Paul Tibbit", "genre" : "children",   "production": { "script": "someone",  "music": "somebody",  "Photography": "din charles"} }, {"movieName": "Baby Boss",  "id": "333",
"desc": "bla blab la bla",  "stage manager":"Tom makarts", "genre""comedy",
"production": {  "script": "someone",  "music": "somebody",   "Photography": "din charles"}
}, {"movieName": "Ben Gurion Epilogue",      "id": "444",     "desc": "bla blab la bla",  "stage manager":"Yariv muzar", "genre" : "israeli",   "production": { "script": "someone", "music": "somebody",  "Photography": "din charles"} } ] )
```

שליפת נתונים / שליפת נתונים בצורה מסודרת

- במידה ויש תנאי של אי שיוון כלשהו, אז זה יהיה בתוך סוגריים {} ולפני $

לדוגמא:

{"key":"value"} מול { "key":{$lt:value} }

```
db.movies.find()
db.mycol.find().pretty()
db.COLLECTION_NAME.find({ "stage manager" :  "Paul Tibbit" })
```

שליפה עם "או" / "וגם" – ברירת מחדל זה "וגם"

```
db.mycol.find( { $and: [ {key1: value1}, {key2:value2} ] } ).pretty()
db.mycol.find( { key1: value1, key2:value2} ).pretty()
db.mycol.find( { $or: [ {key1: value1}, {key2:value2} ] } ).pretty()
```

הסרה

```
db.COLLECTION_NAME.remove(DELETION_CRITTERIA)
```

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

```
> db.movies.remove({"movie name":"baby boss"})
```

עדכון

```
> db.movies.update({"id":"111"},{$set:{"id":"444"}})
```

אם ID היה שווה ל111 אז הוא יעדכן אותו ל444
Multi:true – במקרה של ריבוי התאמות ישנה את כולם.
ללא המולטי - ישנה רק אחד. (הראשון)

שליפת חלק מהנתונים

```
> db.movies.find({},{"movie name":1})
{ "_id" : ObjectId("592415ed6a7dcadde62f8c2a"), "movie name" : "the zookeeper wife" }
{ "_id" : ObjectId("592416466a7dcadde62f8c2b"), "movie name" : "the spongeBob movie" }
```

```
> db.movies.find({},{"movie name":1, _id:0})
{ "movie name" : "the zookeeper wife" }
{ "movie name" : "the spongeBob movie" }
```

הגבלה על מס' המסמכים לשליפה

```
> db.movies.find({},{"movie name":1, _id:0}).limit(1)
{ "movie name" : "the zookeeper wife" }
```

דילוג על תוצאות

```
> db.movies.find({},{"movie name":1, _id:0}).limit(1).skip(1)
{ "movie name" : "the spongeBob movie" }
```

מיון

```
> db.movies.find({},{"movie name":1,"genre":1, _id:0}).sort({"genre":1})
{ "movie name" : "the spongeBob movie", "genre" : "animation" }
{ "movie name" : "the zookeeper wife", "genre" : "drama" }
> db.movies.find({},{"movie name":1,"genre":1, _id:0}).sort({"genre":-1})
{ "movie name" : "the zookeeper wife", "genre" : "drama" }
{ "movie name" : "the spongeBob movie", "genre" : "animation" }
```

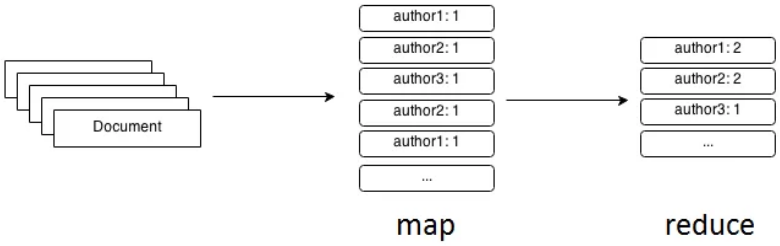
פונקציות Aggregation

db.movies.aggregate([{$group:{MyID:"$id",MyMin:{$min:"$time"}}}])

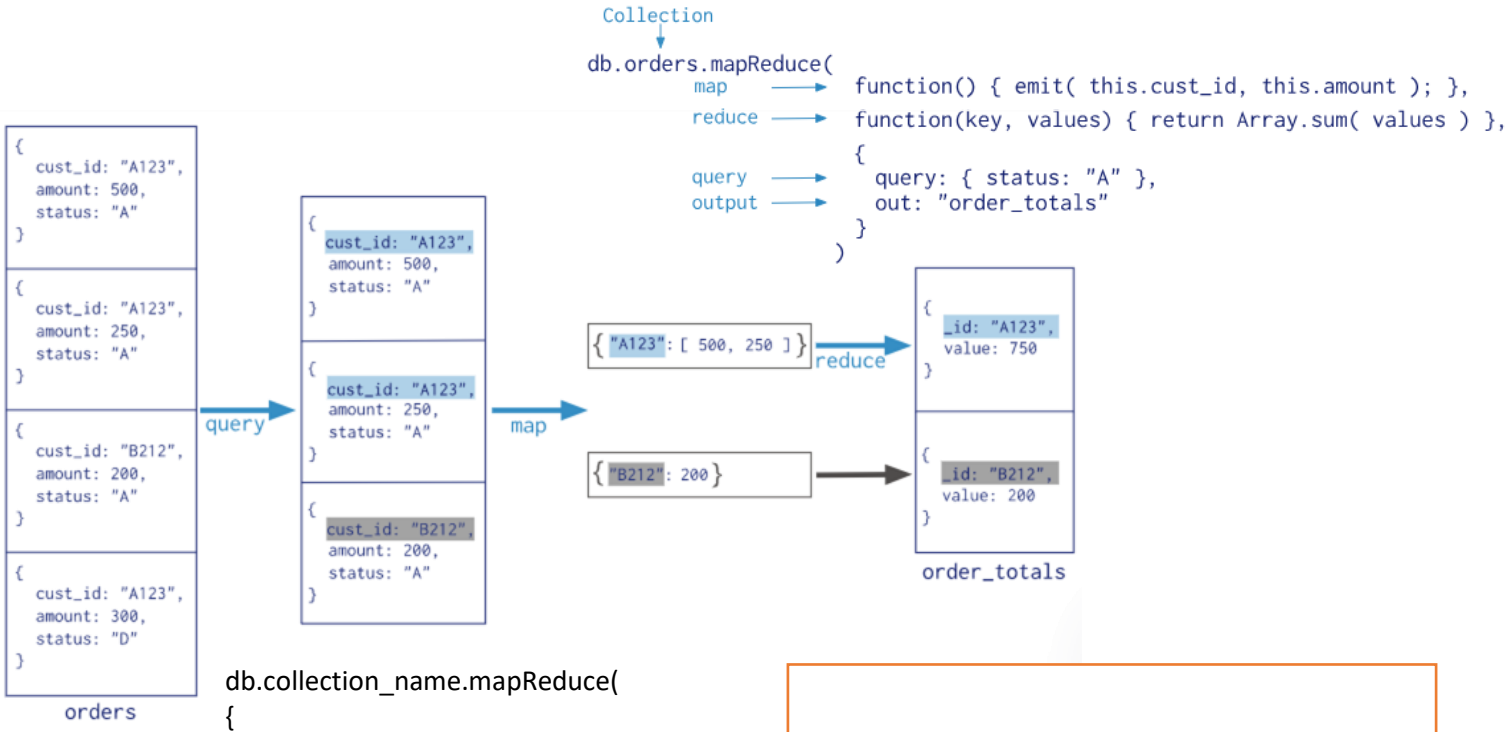
```
> db.movies.aggregate([{$group:{_id:"$movie name",min_age:{$min:"$age"}}}])
{ "_id" : "The jungel book", "min_age" : "10" }
{ "_id" : "The Zookeeper wife", "min_age" : "16" }
{ "_id" : "The Lion King", "min_age" : "4" }
```

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Map-Reduce :
עיבוד מידע



- A set of algorithms allowing parallel execution on massive amounts of data.
- **Mapper:** splits data, filters and runs a process.
- **Shuffle and Sort / Grouping:** ensures that all worker nodes have all data required for reduce.
- **Reduce:** worker nodes process each group of output data and build the output.
- We will come back to this paradigm when we learn Spark.




```
db.collection_name.mapReduce(
{
    mapper,
    reducer,
    {
        out : “out_name”,
        finalize: finalizeFunction
    }
    }
);
db.collection_name.find(out_name);
```

Finalize מבצע פעולה על התוצאה הסופית, יכול להיות Aggregation (Sum,Avg..).

ElastricSearch

מחזיר תוצאות לפי רלוונטיות התשובה, מאפשר חיפוש לפי טקסטים.
מדרג תוצאות לפי TF-IDF

TF מחפש את כמות החזרות (את המילים שהיו בשאילתה) בתוך כל אחת מהתשובות, וככל שיש יותר מילים נקבל דירוג יותר גבוהה.

IDF – מסדרים מילים שהם יותר נפוצים ופחות, בנוסף זה גם מוריד מילים שחוזרות בעצמם הרבה מקבלות משקל גדול יותר במכנה.

- **נשים לב כי הסכימה היא פר מילה ואז בסוף סוכמים הכל.**

כמה פעמיים המילה מופיעה במסמך

כמה מסמכים סה"כ יש

כמה מילים יש במסמך

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

d: current document
D: full corpus (of documents)
k: word in document/query
Q: set of words in query

TF

IDF

כמה מסמכים מכילים את המילה הרצויה

דוגמא:

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?

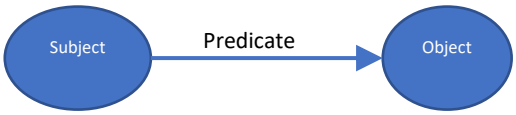
Doc	Tf-Idf score
D1	(1/6)*log(4/2)+(1/6)*log(4/1)+(1/6)*log(4/3)+(1/6)*log(4/2)=0.736
D2	(3/15)*log(4/3)+(2/15)*log(4/3)+(1/15)*log(4/3)=0.166
D3	(1/14)*log(4/2)+(1/14)*log(4/2)+(1/14)*log(4/3)+(1/14)*log(4/3)+(3/14)*log(4/3)+(1/14)*log(4/2)=0.363
D4	(log(4/2)+2*log(4/3)+log(4/3))/11=0.204

נשתמש ב-Curl בשביל פקודות ב-http שהבסיסים זה Get, Post, Put, Head, Delete.
Curl -XPUT – להכניס ערך.
Curl -XPOST – להכניס מסמך בלי id (יתעדכן אוטומטית)
Curl -XGET – להוציא מידע.
Curl -XHEAD – בודק האם מסמך קיים (להחליף את שאילתת XGET עם XHEAD)

ARQ / SPARQL

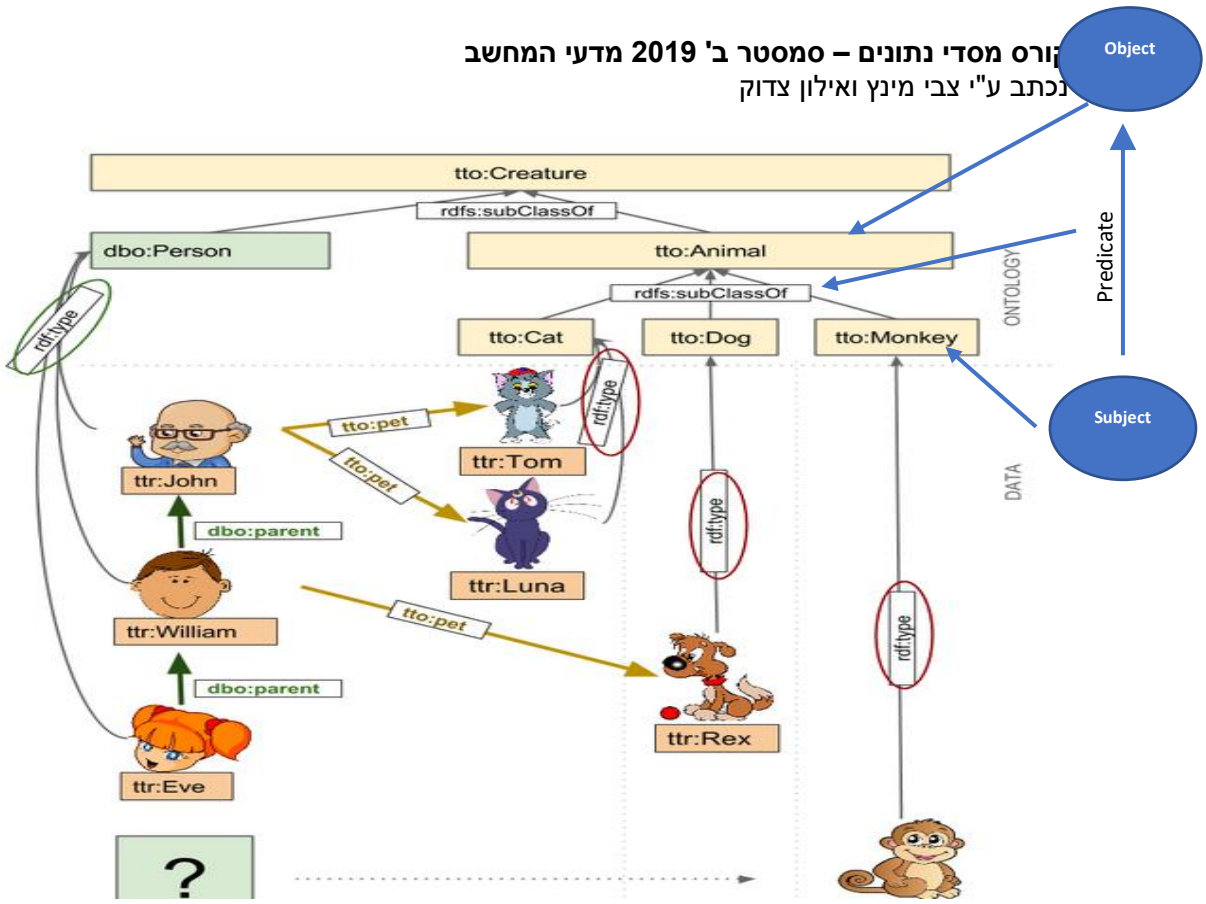
יתרון: מאפשר חיפוש נוח, לדוגמא כל הבנים של .. וכו', אפשר לגשת לתת גרף.

מבנה של שלשות, יש לו :



נושא, תיאור, אובייקט

לדוגמא: צבי אוהב פיצה



שליפות:

SELECT * WHERE {?s ?p ?o}

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

שאילתה:

נקבל **Subject** כי הוא הראשון ב Where

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

ה-Something זה משתנה.

```
SELECT ?thing WHERE {
    ?thing rdf:type dbo:Person .
    ?thing tto:sex "female" .
    }
```

"rdf:type" can be replaced by "a"

Select Persons That Have **Cats**

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

person
ttr:John

Select Persons That do **not** Have any Pets

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

person
ttr:Eve

Select Persons That do **not** Have any **Cats**

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

/ Concatenates relations

person
ttr:Eve
ttr:William

ה-/ חוסך את ?cat ו?cat ב2 השורות

Select all Creatures (using UNION)

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

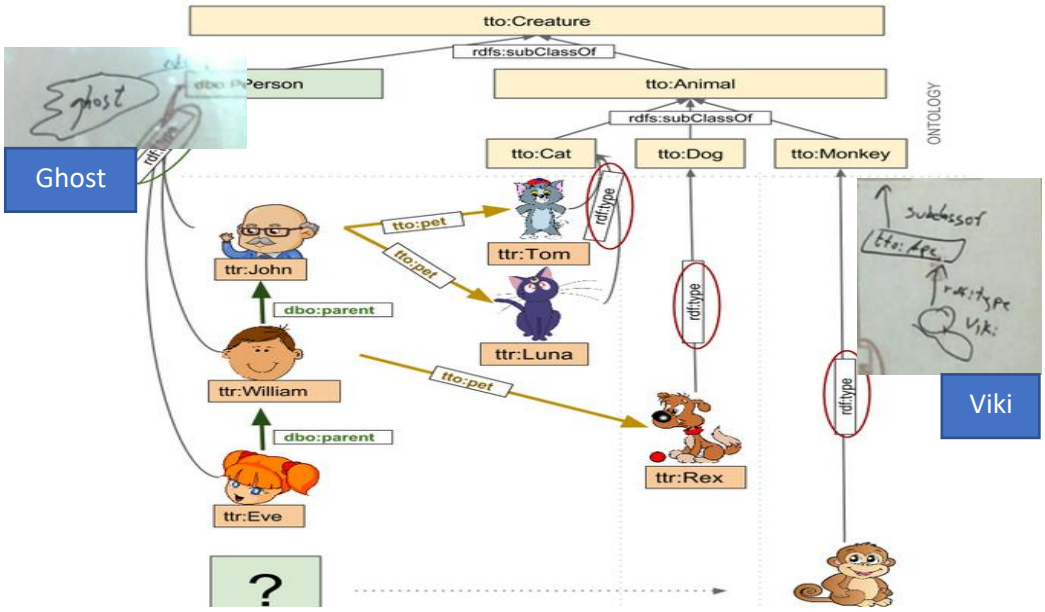
thing
ttr:Eve
ttr:John
ttr:William
ttr:LunaCat
ttr:TomCat
ttr:RexDog
ttr:SnuffMonkey

Select all Creatures (simpler and more correct)

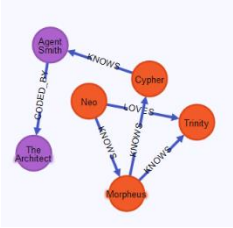
- SELECT ?thing WHERE {
  ?thing a / rdfs:subClassOf+ tto:Creature .
  }

+ is 1 or more of same predicate
(* is 0 or more)
(? Is 0 or 1)

thing



ההבדל בינהם הוא שעם ה+ שאם נוסיף Ghost אז זה לא יופיע בשינהם,
בגלל שצריך שה-type צריך להיות subclass של creature (צריך להיות משהו באמצע) ולכן זה לא מופיע באף אחד. אם נוסיף את Viki אז זה יופיע ב+ ולא בשני (יותר מנכד)



Neo4j

גרף מאוד שימושי בשביל למצוא שידוכים וחברים בגרף.
מבנה נתונים שבנוי בצורת גרף, יש בו קודקודים וקשתות שנקראים Relation


```
MATCH (a: person{name: "bob"})--(b) return b
```

שאילתא זו אומרת תתן את כל הקודקודים שיש להם קשת (אחת בלבד) בינם לבין קודקוד שה label שלו זה person (אפשר להסתכל על זה כמו על class שהוא person) .

אם נרצה את שמות האנשים שקשורים לבוב:

```
MATCH (a: person{name: "bob"})--(b: Person) return b.name
```

אם נרצה את שמות החיות שקשורות לבוב:

```
MATCH (a: person{name: "bob"})--(b: Pet) return b.name
```

אפשר לשאול על קודקודים שקשורים בקשר מסויים למשל:

```
MATCH (a: person{name: "bob"})-[: merriage]-(b) return b
```

יחזיר את כל הנשואים לבוב

אפשר לסמן שהקשר הוא בכיוון מסויים:

```
MATCH (a: person{name: "bob"})-[: like]->(b) return b
```

כל הקודקודים שבוב אוהב.

אפשר לשאול את כל החברים או החברים של החברים וכו (מותר עד 10 קשתות)

לדוגמא:

```
MATCH (a: person{name: "bob"})-[*]-(b) return b
```

יחזיר לנו את כל הקודקודים שקשורים לבוב עד 10 קשתות.

```
MATCH (a: person{name: "bob"})-[: friend*..3]-(b) return b
```

יחזיר לנו את כל הקודקודים שקשורים לבוב עד 3 קשתות.

```
MATCH (a: person{name: "bob"})-[: friend*2..3]-(b) return b
```

יחזיר לנו את כל הקודקודים שקשורים לבוב 2 קשתות ל3.

```
MATCH (a: person{name: "bob"})-[: friend*2..3]-(b)  where b.age > 100 return b
```

אפשר להחזיר פונקציה:

```
MATCH (a: person{name: "bob"})-[: friend*2..3]-(b) return max(b.age)
```

אפשר שאילתא יותר מורכבת:

```
MATCH (a),(b),(c) where (a: person{name: "Yoav"})-[: friend*2..3]-(b) and (b)-[: like]-(c: Movie)
return c.name
```

מחזיר את שמות הסרטים שחברים של בוב אהבו

- מוסיפים צומת ע"י Create, ניתן גם כמה כאשר כל אחד בתוך סוגריים ()
- לפי נקודתיים : זה שם הצומת ואחרי זה התווית (רפרנס)

- ניתן להוסיף בCreate קשרים ע"י (negev)-[r1:teaches]->(glass) כאשר negev זה רק רפרנס לשם.

➢ CREATE (n)

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

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

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

Properties

Can use an empty reference

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

Can create many nodes at once

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

- הסוגריים [] רקים זה סוג של קשר לא משנה איזה.
- **Find זה Match**

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

:Paths

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

- Will return all paths of length 2 to 4 of type KNOWS, between Gadi Golan and others.

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

- Will return the shortest path (using any type of relation, and in any direction) between Gadi Golan and Tal Negev (of type students).

With, All / Any, In, Collect, Count, And , Or

ALL – כולם, Any – מספיק אחד

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

- Returns all students that study all courses.

מאפשר לשרשר Match

➢ MATCH (s:student)-[:studies]->(c:course) **WITH** s, COUNT(c) as num_courses **WHERE** num_courses <= 4 RETURN s.name

- Returns all students that study at most 4 courses.

➢ MATCH (negev { name:"Tal Negev" })-[:friends]->(frOfNegev:student)-[:knows]->(n:student) WITH frOfNegev, COUNT(frOfNegev) AS frCount WHERE frCount > 3 RETURN frOfNegev

- Returns all students that are Tal Negev's friend and know more 3 students.

s must appear again in WITH part, **so we can return s.name**

- Write a query that returns all the nodes that have any connectivity (of any length) with 'Tal Negev'
  - ➢ MATCH (a {name:'Tal Negev'})-[*]-(b) RETURN DISTINCT b
- Write a query that returns all students that study all courses that 'Tal Negev' learns, but are under the age of 30.
  - ➢ MATCH (a {name:'Tal Negev'})-[:studies]->(c:course) **WITH** COLLECT(c) AS negev_courses MATCH (s:student) WHERE s.age < 30 AND ALL (x IN negev_courses WHERE (s)-[:studies]->(x)) RETURN s

7-JavaStreamsFile

לפעמיים לא ניתן לעשות את כל החישובים במסד נתונים היות ואנחנו לא רוצים לשמור את המידע שנקבל, אם נרצה לעשות את כל החישובים בקוד נשתמש ב-Java Streams, אם נרצה לעשות את החישובים במסד נתונים נוכל להשתמש Stored procedure

נעזר בLambda Expressions :

```
Arrays.sort(dogArray, new Comparator<Dog>() {
    @Override
    public int compare(Dog o1, Dog o2) {
        return o1.getWeight() - o2.getWeight();
    }
});
```

Lambda Expression

```
Arrays.sort(dogArray, (m, n) -> m.getWeight() - n.getWeight());
```

```
(int arg1, String arg2) -> {System.out.println("Two arguments "+arg1+" and "+arg2);}
```

Argument List — Arrow token — Body of lambda expression

דוגמא:

```
List<String> myList =
   Arrays.asList("a1", "a2", "b1", "c2", "c1");

myList
   .stream()
   .filter(s -> s.startsWith("c"))
   .map(String::toUpperCase)
   .sorted()
   .forEach(System.out::println);
```

Function – פונקצייה שמקבל קלט.

Callable – פונקצייה שלא מקבל קלט.

קוראים לCallable ע"י ()call.

Stream – סדרה מופשטת של אלמנטים התומכים בביצוע פעולות צבירה (Aggregation), באופן סדרתי או מקבילי.

```
List<Integer> even = numbers.stream()
                            .map(s -> Integer.valueOf(s))
                            .filter(number -> number % 2 == 0)
                            .collect(Collectors.toList());
```

מחזיר כרשימה

דוגמא ל-anyMatch :

```
public class Main
{
    public static void main(String[] args)
    {
        Stream<String> stream = Stream.of("one", "two", "three", "four");

        boolean match = stream.anyMatch(s -> s.contains("four"));

        System.out.println(match);      //true
    }
}
```

```
List<String> roomsReqAccess = classrooms.entrySet()
            .stream()
            .filter(a->a.getValue()
                    .stream()
                    .anyMatch(x->x.reqAccesability))
            .collect(Collectors.toList(c->c.getKey()));
```

ניתן במקום Stream להשתמש ב ParallelStream כאשר הסדר **לא** חשוב.

פונקציית Reduce :

חותמת של Reduce : **reduce([identity], accumulator, [combiner])**

לדוגמא:

Reduce("", (x,t) -> x+t.charAt(6), (x,y)-> x+y);

כאשר [identity] זה בעצם המצב ההתחלתי, accumulator היא הפונקצייה האוגרת כאשר x זה מה שהיה עד עכשיו ו-t זה המילה הנוכחית, ו-[combiner] כיצד לשלב את חלקי התוצאות.

דוגמא:

```
orders.stream()
        .filter(a->a.status.equals("A"))
        .map(a-> a.amount)
        .reduce(0, (x,y)-> x+y);
```

הערה: בגלל שלא כתבנו את ה-combiner אז הוא יקח כברירת מחדל את הפונקצייה של ה-accumulator

8-Python BasicsFile

Python מול Java

The pros of Java over Python are:

1. Java programs execute faster as compared to Python.
2. Java application are much more stable and secure as compared to Botha s they basically run in a virtual secluded environment which is the JVM.
3. It can easily handle very large data sets and work efficiently.
4. The java applications are much more scalable.
5. If you are working with databases then JDBC and ODBC in Java can easily help you out whereas on Python the database access layers are a tad bit underdeveloped.
6. This point might not be true for all but I personally find it extremely difficult to debug Python indentation errors. Whereas the java braces are very easy to spot and rectify. Basically moving the code from editing in one IDE to another is a nightmare.

Now for the pros of Python over Java:

1. Python codes are much compact and easier to comprehend as compared to Java codes.
2. The learning curve of Python is easy and it can deliver some pretty complex programs in a matter of lines.
3. Python is dynamically typed and there is no need to declare anything. An assignment statement directly binds a name to an object, and the object can be of any type.
4. Python has become the backbone of Internet of Things. Infact the Pi in the Raspberry Pi stands for Python.
5. A huge set of tools for Data Analysis, Scientific research are written in Python 2 which are currently being rewritten and upgraded to Python 3.
6. Python's feature of Integration makes it possible to call Python through java via Jython.

מפרש (אנגלית: **Interpreter**) הוא תוכנה הקוראת תוכנית מחשב הכתובה בשפת תכנות ומבצעת אותה ישירות, פקודה אחר פקודה.

מפרש לעומת מהדר (Compiler):

כל מחשב הוא מכונה, המסוגלת להריץ תוכניות הכתובות בשפה הייחודית לה. לכן, על מנת להריץ תוכנית מסוימת, הכתובה בשפה שאינה שפת המכונה של המחשב המריץ, ראשית יש לתרגמה לשפה זו. תרגום זה יכול להתבצע פעם אחת (ואז בסיום התרגום נוצר קובץ הכתוב בשפת מכונה, הניתן להרצה בכל עת) כפי שנעשה על ידי מהדר, או לפני כל פעם בה מריצים את התוכנית, כפי שנעשה על ידי מפרש. ההבדל בין השניים, איפוא, הוא מתי מבוצע התרגום. מהבדל זה נגזרים המאפיינים של כל אחד מהם.

תהליך ההידור הוא תהליך מורכב מאד, הספציפי למכונה מסוימת. קוד המכונה שנוצר מתחשב בחומרה הספציפית עליה הקוד עתיד לרוץ וכן במערכת ההפעלה המותקנת עליה. כלומר במידה והידרנו תוכנית מסוימת באמצעות מהדר המותאם למכונה מסוימת, התוכנית תרוץ על מכונה זו בלבד. מפרש, לעומת זאת, יאפשר הרצה של התוכנית כמעט בכל מכונה משום שהתוכנית תתורגם על כל מכונה מחדש.

מאידך, ברור כי שימוש במפרש עלול לצרוך משאבים רבים בשל העובדה כי התוכנית מתורגמת כל פעם מחדש. בנוסף, דרישות מסוימות של התוכנית (כמו הקצאת זיכרון) עלולים לקחת יותר זמן בעת שימוש במפרש, לאור העובדה שהתוכנית לא עובדת ישירות מול מערכת ההפעלה, אלא מול המפרש.

<u>שפה</u> – במצגות.

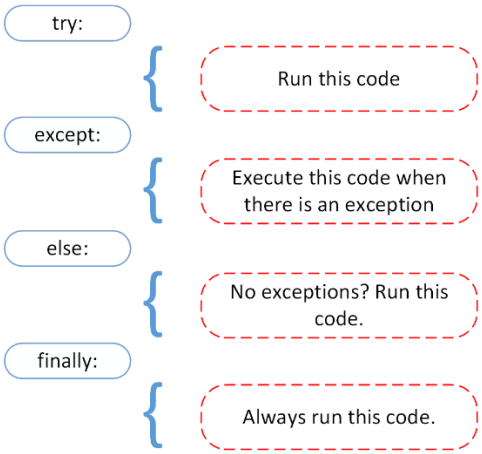
Lambda בפייטון:

```
x = lambda a, b : a if a>b else b

y = x(14,5)
print(y)
```

מחלקות:

```
class University():
    def __init__(self, name, rank):
        self.name = name
        self.rank = rank

    def name(self):
        print "The University name is %s" % (name)
        print "This University is ranked at %s" % (rank)
```

טיפול בחריגות:



Pip - pip is a package-management system used to install and manage software packages written in Python

NumPy - is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Import Numpy as np

התעסקות עם מערכים ב-NumPy, ממיר תמיד לטיפוס המורחב ביותר, לדוגמא עבור 2,2.0,4 אז הוא ימיר את המערך למספרים ממשים, בעוד שבמערך "Hello",2,2.2 נקבל מעבר לתווים.

9-SparkFile

Spark – זהו למעשה הדור הבא של עיבוד מידע, הרבה יותר יעיל מ-Map/Reduce, הוא יותר מהיר, יודע לנצל זכרון בצורה מיטבית, הרבה יותר קל ונוח לפתח בו, ניתן לעשות בו עיבודים של Machine Learning.

בשני הכלים עובדים עם אחת משפות פיתוח עיליות: Java, Scala או Python.

טכנולגיה שמאפשר להתעסק עם הרבה נתונים (לא מסד נתונים כי הוא לא מאחסן נתונים אלא רק מאפשר לבצע פעולות על הרבה נתונים)

יתרונות נוספים של Spark:

- גמישות – האפשרות לייצר מאפס פתרון מותאם צורך.
- תומך בשליפות SQL – Spark SQL.

- מספק עיבוד בזמן אמת – Spark Streaming – real-time streaming analytics.
- יכול לרוץ על Hadoop Cluster או בענן, או על Cluster משל עצמו.
- יכול לגשת למקורות מידע מאוד מגוונים: HDFS, Apache Cassandra, Apache HBase, Amazon S3 cloud-based storage

Spark עובד עם אובייקט מסוג **RDD** ומחזיר אובייקט **RDD**, סוג של Stream.

Resilient Distributed Dataset, the central data structure of Apache Spark

הקוד:

קורא טקסט מקובץ והופך לאוביקט RDD

קורא שורה שורה מפריד מילים ומשטח לרשימה אחת ולא רשימה לפי שורה

הופך כל מילה לאובייקט מורכב מkey שהוא המילה ומvalue שזה 1

מקבץ את האובייקטים מהשורה הקודמת לפי key דומים וסוכם את הvalue

הופך לlist

```
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))
```

: Sort By Value

```
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])) \
        #False is for descending order
        .sortByKey(False) \
        .map(lambda xy: (xy[1],xy[0])) \
        .collect())
```

כשקוראים מסמך גדול, ניתן להשתמש בCache() על מנת לטעון את המסמך לתוך הזכרון, כך שהפונקציות הבאות יופעלו במהירות.

:Bi-Grams

A bigram or digram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words

Bi-Grams זה בעצם הסתכלות על זוגות של מילים, בד"כ משומש בתהליכי עיבוד שפה כדי להבין תחביר של משפט באופן טוב יותר מאשר הסתכלות על מילה בודדת.

דוגמא:

"I did it you did it you did it"
[((I, did), 1), ((did, it), 3), ((it, you), 2), ((you, did), 2)]

Tri-Grams זה עם שלשות, ובכללי זה נקרא N-Gram.

Zip – מייצר זוגות של $(a[i], b[i])$ עבור כל $i \in a.length$

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

מימוש Bi-Gram ע"י Zip:

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

>>>pairs = text_file.flatMap(bigram)
>>>count = pairs.map(lambda x: (x, 1)).reduceByKey(lambda a, b: a + b)
```

Data-frame

יחזור על זה בהרצאה למה אין לי מושג.

10-NaiveBayesFile