מסדי נתונים

מחברת סיכומים לקורס מסדי נתונים 2021

מסדי נתונים - מחברת הרצאות

נערך ונכתב על-ידי דור עזריה

2021

ספר זה לא נבדק על ידי מרצה, יתכן שימצאו טעויות.

לסיכומים נוספים שלי במדעי המחשב ומתמטיקה:
https://dorazaria.github.io/

מוזמנים לעקוב אחרי 😊:
LinkedIn: https://www.linkedin.com/in/dor-azaria/
GitHub: https://github.com/DorAzaria

תוכן עניינים

מה זה מסד נתונים?

בסיס נתונים או **מסד נתונים** (באנגלית: **Database** - דאטה בייס, או בקיצור **DB**) הוא אמצעי המשמש לאחסון מסודר של נתונים במחשב, לשם אחזורם ועיבודם.
בסיס נתונים מאוחסן באמצעי אחסון נתונים, בדרך כלל על גבי דיסק קשיח, המאפשר גישה ישירה לנתונים.
הגישה לבסיס הנתונים נעשית באמצעות תוכנה ייעודית - מערכת לניהול בסיס נתונים.
בסיס הנתונים בנוי לפי מודל לאחסון הנתונים, כמו מנגנונים פנימיים למיון ולחיפוש.

המודל הנפוץ ביותר היום, הוא **המודל הטבלאי**, כל טבלה היא **ישות** כלשהי.
שליפת מידע ופעולות עדכון בבסיס נתונים טבלאי נעשות באמצעות **שפת SQL**, המהווה ממשק המאפשר גישה לנתונים מבלי להתייחס לאופן שמירתם בבסיס הנתונים.

מרכיבי בסיס הנתונים במודל של טבלה

יחידת הנתונים הקטנה ביותר בבסיס הנתונים היא **השדה - העמודה** בטבלה (field).
בנתוני אזרח במרשם התושבים, למשל, מספר הזהות, השם הפרטי, ושם המשפחה הם שדות בטבלה.
השדה חייב להיות מסוג מסוים (int, char וכו'..), שמגביל את הזנת הערכים לו לאותו סוג נתונים.
כך, למשל, שדה שהוגדר מספרי לא יוכל להכיל תווי אותיות.

אוסף כל השדות המתארים את האזרח הוא **הרשומה - השורה** בטבלה (record) של אותו אזרח.
לכל רשומה יש מפתח ראשי (primary key), המשמש לזיהוי חד-משמעי שלה.
ברשומת האזרח, מספר הזהות משמש כמפתח ראשי, ושם המשפחה והשם הפרטי משמשים כמפתח משני.

מסד נתונים רלציוני

מידע של יחסים - המשמעות זה בעצם טבלה.
יש לנו DB שמכיל את המידע של האוניברסיטה או של כל מיני הזמנות של חברה מסוימת וכו'...
לכל DB יש נושא מסוים.
ה-DB מורכב בטבלאות כך שכל טבלה היא **ישות** כלשהי ויש קשר בין הישויות (טבלה היא ישות).
למשל DB של אוניברסיטה יש ישויות של סטודנטים, מרצים, ציונים וכו'... ויש קשר ביניהם.
הקשר הוא תמיד דו-כיווני בין הטבלאות.

כל טבלה מכילה את **השדות** (עמודות), (למשל עבור הישות סטודנטים השדות יהיו תז, שנת לידה ממוצע וכו'..)
וב**רשומות** (שורות) יופיע לנו המידע (בהתאם לעמודות).

תז	שם	שנת לידה	ממוצע
123414	שימי תבורי	1832	100
41243121	בן אל תבורי	2018	80

ה-DB רנצליונים מבוססים על **שפה SQL**, השפה מבוססת על איזשהו תורה במתמטיקה שנקראת אלגברה רלציונית.
דוגמא לDB מפורסמים mySQL, SQLite ....

ה-mySQL הוא קוד פתוח ולו יש סביבת עבודה נוחה בשם GUI **workbench** .
בעזרת ה-GUI ניתן לכתוב באופן מאוד ידידותי את כל הפקודות ונוכל גם להריץ אותם, בריצה מתקיימת גם פנייה לשרת שם גם מתבצעים ונשמרים או נמשכים הנתונים.
ה-SQL Server הוא בתשלום, יש לו גם GUI ויש לו גישה בחינם מהאוניברסיטה.

במבט מלמעלה:
מסד נתונים (**DB**) הוא נושא מסוים, בתוכו יש אוסף של **ישויות** (טבלאות) שקשורות אחת לשניה.
לכל טבלה יש **שדה** (העמודה) **ורשומה** (השורה).
בעזרת שפת ה-SQL אפשר להשתמש בנתונים מהמסד וכך נבצע עליהם פעולות כלשהן.

עקרונות ACID

טרנזקציה: זה **אוסף של פעולות** לווא דווקא פעולה אחת, יכול להיות אוסף פעולות מסוגים שונים.
יש לנו 4 פקודות בסיסיות: הכנסה, עדכון, מחיקה ושליפה של נתונים מתוך ה-DB .

דוגמה לטרנזקציה:
נניח נרצה להעביר כסף מחשבון אחד לאחר, אז מחשבון אחד נמחוק את הכסף ומחשבון האחר נוסיף את הכסף.
אנחנו צריכים לוודא ששניהם יתבצעו, למשל כדי שלא יווצר מצב שנוריד כסף מחשבון אחד ולא נכניס לחשבון השני.
טרנזקציה חייבת להתבצע **באופן מלא** בלבד.

ACID הם 4 עקרונות שכל DB **רלציוני** חייב לקיים.
בעזרת עקרונות אלה אפשר להבין איזה יתרונות נקבל או חסרונות באיבוד עקרונות האלה ומה הכוח שלהן.

1. **Atomicity** - כל טרנזקציה חייבת להתקיים או במלואה או לא להתקיים בכלל.

2. **Consistency- קביעות** , הכוונה היא שיש לנו איזשהו אוסף של פעולות (טרנזקציה), כל טרנזקציה שנעשה לא פוגעת בDB אם היא תתבצע, אם היא תשנה את DB (מה שאסור) אז היא פשוט לא תתבצע.
לדוגמה: ב-DB רלציוני אמרנו שמגדירים בטבלה עם כל מיני תכונות (סוגים\טיפוסי משתנים) שהם בעצם העמודות של הטבלה.
אז אם נכניס לעמודה מסוג int איזשהו string אז נקבל שגיאה, זה פעולה לא חוקית.

3. **Isolation - נבדלות**, הכוונה היא כשאנחנו מדברים על שרת של DB בעולם האמיתי, המון בקשות ל-DB מתבצעות במקביל, נתאר למשל אתר אינטרנט של קופת חולים כללית ונרצה לראות את רשימת המקומות שניתן לקבוע בו תור לחיסון קורונה, אז יכול להיות שיהיו כמה פניות במקביל מכל מיני משתמשים ביחד לאותו DB ספציפי.
אז יכול להיות מצב שה-DB יעשה מלא טרנזקציות בצורה מקבילית אבל חייבים להבטיח שהתוצאות לא ישתנו.
לדוגמה: נגיד שיש חשבון משותף לזוג נשוי, לשניהם יש אמצעי תשלום למשל כרטיס אשראי שדרכו אפשר להוציא כסף ושניהם ירצו להוציא כסף (אסור להיכנס למינוס) ונניח שיש 800 שח בחשבון (ונרצה להוציא 500) אז אם האישה תוציא ראשונה ישאר 300 ואז בעלה שירצה להוציא אחריה לא יוכל להוציא 500 הוא לא יכול (הוא יקבל הודעת סירוב)
כלומר אם נעשה את הפעולות האלה בפעולה (תורית) זה ימנע את המצב הזה.
איך מממשים את זה בפועל? עושים כל מיני דברים שבעצם **ינעלו** את הטבלה.

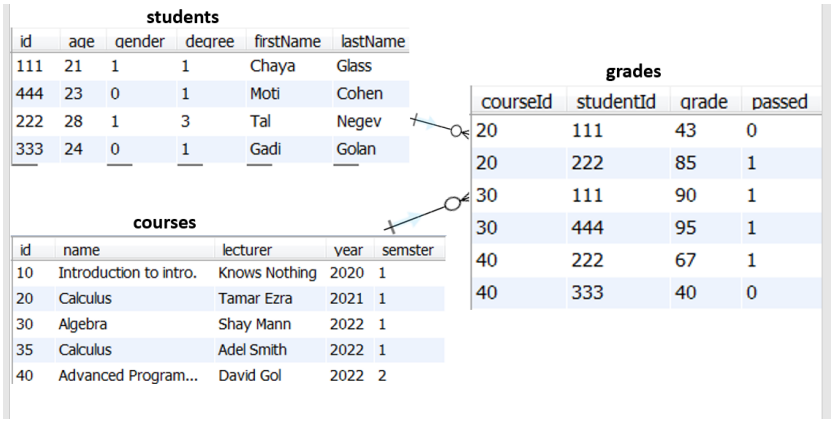
4. **Durability - המשכיות/יציבות** , הכוונה היא ש-DB מתחייב לסוג של זמינות וגם אם הוא לא זמין מסיבה מסוימת (תקלת חשמל) קרה איזשהו כשל והשרת לא יכול לעבוד, עדיין יש התחייבות ש-DB מתחייב לספק לנו את השירות שלו.

ארבעת העקרונות האלה מבטיחות DB מאוד אמין וחזק.

שפת SQL

שפת ה-SQL היא בעצם שפת הפקודות שנבצע על ה-DB. הפקודות מבוססות על אלגברה רלציונית.
אפשר לכתוב את השפה באותיות קטנות או גדולות, מילים ששמורות בשפה נכתוב באותיות גדולות לנוחות הכתיבה והמימוש.

בתוך ה-DB שלנו, יש 3 טבלאות שנעבוד עליהם:



students

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

grades

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

courses

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

החצים מראים את הקשר בין היישויות (הטבלאות) בדר"כ אמור להיות איזשהו קשר בין כל הטבלאות.
לא חייב להיות קשר ישיר אבל חייב להיות קשר מסוים.
כאן למשל יש את הסטודנטים והקורסים ושניהם מקושרים לטבלת הציונים.
אפשר לראות גם שהעמודה courseID מטבלת courses וה-studentID הגיע מטבלת students.
אז בעצם כבר מהסתכלות על העמודות, אפשר להבין בערך את הקשר בין הטבלאות.

פקודות בסיסיות

SQL

```
SELECT id
FROM students
```

זה יקבל את רשימת ה-id מטבלת ה-students, כלומר אנחנו נקבל רשימה המוצגת בטבלה שלא נשמרת בזיכרון.

id
111
444
222
333

SQL

```
SELECT id*2
FROM students
```

אפשר לכתוב לא רק שם של עמודה אלא אפילו פעולה חשבונית.

id*2
222
444
666
888

בהמשך נציג כאן פונקציות וכשנפעיל אותם נקבל את התוצאה שנרצה לקבל בהתאם לעמודה.

SQL

```
SELECT year, semester
FROM courses
```

נקבל עמודה של year ועמודה של semester ונקבל את כל הקורסים.

year	semster
2020	1
2021	1
2022	1
2022	1
2022	2

יש חשיבות לסדר כתיבת העמודות, כלומר אם נכתוב `SELECT semester, year` אז נקבל קודם עמודה של semester ואז עמודה של year.

SQL

```
SELECT *
FROM grades
```

שולף את הכל מהישות grades.

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

SQL

```
SELECT DISTINCT year,semester
FROM courses
```

הוא יחזיר רק את המקרים השונים, כלומר נקבל מקרים בלי חזרות.

year	semester
2020	1
2021	1
2022	1
2022	2

לעומת,

year	semster
2020	1
2021	1
2022	1
2022	1
2022	2

עד עכשיו, שלפנו את כל הרשומות בטבלה, יכולנו לבחור איזה עמודות לשלוף בטבלה, אבל לא נגענו ברשומות.
כל הרשומות שיש לנו קיבלנו, אבל בעולם האמיתי יהיו לנו המון המון רשומות (זה ממש אינסופי בסוגו).
לעומת זאת, העמודות הם סופיות (10 או 50 או 200) מישהו ב-DB החליט כמה עמודות יהיה הטבלה ולפי זה יכולות להיכנס כמה רשומות שצריך (אינסופי) למשל תמיד יכנסו עוד סטודנטים חדשים לאוניברסיטה, אז העמודות לא משתנות או נוספות אלא השורות, כל שורה (רשומה) זה סטודנט חדש.

SQL

```
SELECT id
FROM students
WHERE degree < 2
```

נקבל את כל הסטודנטים עם תואר ראשון.

id
111
333
444

SQL

```
SELECT age
FROM students
WHERE firstName LIKE ‘%i’
```

נשתמש ב-LIKE על מחרוזות.
הפקודה כאן בעצם מחפשת בשדה firstName , שם כלשהו שיגמר בתו-i בסוף המילה כמו למשל Ori.
והיא תחזיר לנו עמודה של כל הגילאים של סטודנטים שהשם הפרטי שלהם מסתיים ב-i.

age
23
24

יש משמעות למיקום של ה-%.
כלומר אם 'i%' אז מדובר במילה שמתחילה בתו i כמו למשל isaac.
ואם '%i%' אז הכוונה שפשוט התו i חייב להיות במילה (כולל ההתחלה והסוף).

אופרטורים - conditions

- >, < , = , => , =<
- הפעולה **<>** היא "לא-שווה" (בדומה ל-=! בשפות אחרות).
- **AND** , **OR**

דוגמה לשילוב ביניהם:

SQL

```
SELECT *
FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

- **NOT** , למשל:

SQL

```
SELECT firstName
FROM students
WHERE NOT firstName=’Moti’
```

אפשר גם לעשות <> במקום NOT כלומר firstName<>'Moti'

- **BETWEEN**, (זה כולל הקצוות) למשל:

SQL

```
SELECT *
FROM grades
WHERE grade BETWEEN 80 AND 90
```

courseId	studentId	grade	passed
30	111	90	1
20	222	85	1

- **IN**, למשל:

SQL

```
SELECT *
FROM grades
WHERE studentId IN (111,444)
```

courseId	studentId	grade	passed
20	111	43	0
30	111	90	1
30	444	95	1

- **LIKE** (רק במחרוזות).

Nested Queries

במילים אחרות, nested SELECT הוא query שבתוך query.
למשל כאשר יש לך SELECT בתוך ה-SELECT הראשי.

SQL

```
SELECT lecturer
FROM courses
WHERE id NOT IN (SELECT courseId
                 FROM grades)
```

תמיד נסתכל על הבפנים ונצא החוצה (כמו במתמטיקה).
ב-SELECT הפנימי, שלפנו מטבלת הציונים את הקורסים לכן קיבלנו רשימה של courseId.
ואז ב-SELECT החיצוני נלך לטבלת הקורסים ונשלוף את המרצים אבל בתנאי ש-id לא ברשימה הפנימית.
במילים אחרות, נקבל את רשימת המרצים (עבור קורס מסוים) שעוד לא הכניסו ציונים לסטודנטים.

כלומר, אם יש לנו את 2 היישויות הבאות (מימין זה טבלת grades ומשמאל זה טבלת courses):

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

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

אז נקבל:

lecturer
Knows Nothing
Adel Smith

פעולות על קבוצות

1. איחוד **UNION** - נרצה לאחד נתונים בין 2 טבלאות.

SQL

```
(SELECT id,lastName
FROM students
WHERE id < 200)
UNION
(SELECT id,lastName
FROM students WHERE id BETWEEN 300 AND 400)
```

ה-UNION הוא distinct, זה אומר שלא יהיה כפילות (כמו בתורת הקבוצות).

אם נרצה יהיו כפילויות נעשה **UNION ALL**.

איזה שמות של עמודות יראו? את העמודות של הselect הראשון.

למשל עבור הטבלה הבאה:

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

נקבל:

id	lastName
111	Glass
333	Golan

כללים לאיחוד:

1. חייב להיות אותו מספר של עמודות שאותם אנחנו שולפים.
2. לא חייב שיהיה אותם שמות של עמודות, העיקר שיהיה אותו מספר עמודות.
3. המיקום (הסדר) כן חשוב עבור המשמעות המשותפת של העמודות, זה חשוב גם מבחינה שאם לא נתייחס לסדר יכול גם לגרום למצב שנבקש למשוך מטיפוס int אבל בעצם נגענו בטיפוס מחרוזת.

2. **חיתוך** - ב-SQL אין פעולה ספציפית לחיתוך כי הפעולה **IN** פשוט מבצעת את אותו המשמעות.

מה זה NULL

נועד כדי לציין ערך שחסר.

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan
444	23	0	1	Moti	Cohen
555	24	0	NULL	NULL	NULL
666	27	1	NULL	Tamar	NULL

SQL

```
SELECT *
FROM students
WHERE lastName IS NULL
```

נקבל את כל העמודות מתוך students כך שהשם משפחה שלהם ריק.

id	age	gender	degree	firstName	lastName
555	24	0	NULL	NULL	NULL
666	27	1	NULL	Tamar	NULL

SQL

```
SELECT *
FROM students
WHERE degree = 1 OR degree <> 1
```

בפקודה הזאת יחזרו לנו על הערכים **חוץ** מ- NULL.

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass

בדוגמה הזאת, אם degree הוא גם 1 **וגם** לא 1 אז הוא NULL.
כלומר, כל מה שמבחינה לוגית עבור SQL הוא לא True ולא False הוא בעצם Unknown שמשמעו עבור NULL.

- כאשר מדובר ב-SQL אז קיים true, false ויש גם unknown כלומר יש ערכים שלא נדע אם נכון או לא, והערך null הוא לא true ולא false הוא בעצם **unknown**.
- אפילו NULL=NULL זה unknown.
- עבודה עם NULL היא בעזרת **IS**.

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

NOT	
True	False
False	True
Unknown	Unknown

הפונקציה COALESCE

אפשר לעשות select עם כל מיני פונקציות, יש ב-SQL לא מעט כאלה.
הפונקציה מקבלת כל מיני ערכים ומחזירה את הערך הראשון שהוא לא null.
כלומר אם ברשומה מסוימת יש שדה שהוא NULL (אחד מהשדות שנציין בתוך הפונקציה הוא NULL) אז נשלוף את השדה הבא אחריו שהוא לא NULL, במידה והשדה הראשון שציינו בפונקציה לא NULL, אז נשלוף אותו מיידית.

SQL

```
SELECT id , COALESCE(lastName, firstName, 'unknown')
FROM students
```

נקבל:

id	COALESCE(lastName, firstName,
111	Glass
222	Negev
333	Golan
444	Cohen
555	unknown
666	Tamar

בהמשך נלמד לשנות את שמות העמודות לצורה יפה יותר.

פקודת נוספות

SQL

```
INSERT INTO courses (id,name,lecturer,year,semester)
VALUES (66, 'databases', null, 2025, 1)
```

בפקודה זו נכניס רשומה חדשה ליישות courses.

id	name	lecturer	year	semester
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 Programming	David Gol	2022	2
66	databases	NULL	2025	1

SQL

```
UPDATE Products
SET UnitPrice = 1200
WHERE ProductID = 3
```

הפקודה UPDATE תעדכן את המחיר של מוצר 3.

SQL

```
DELETE FROM `order details`
WHERE OrderId = 10261;
```

הפקודה **DELETE** תמחק את ההזמנה מטבלת פירוט הזמנות.

SQL

```
SELECT id,firstName
FROM students
ORDER BY lastName
```

פקודה **ORDER BY** תסדר את הנתונים בצורה מסוימת (מיון).
אין התחייבות ב-SQL לשלוף בסדר מסוים חוץ מהמילה השמורה **ORDER BY**.
סדר עולה **ASC (דיפולטיבי)** , סדר יורד **DESC.**

id	firstName
444	Moti
111	Chaya
333	Gadi
222	Tal

SQL

```
SELECT gender,age,lastName
FROM students
ORDER BY gender ASC, age DESC
```

כלומר סדר המיון הראשון, קובע את המיון השני.
קודם נמיין את gender ומתוך המיון הזה נמיין את הגילאים בהתאם.

gender	age	lastName
0	24	Golan
0	23	Cohen
1	28	Negev
1	21	Glass

SQL

```
SELECT * FROM students LIMIT 2
```

ה-LIMIT מאפשר לנו לשלוף מספר מסוים של רשומות בלי WHERE, לפעמים הוא מאוד שימושי לעומת WHERE בדוגמה זו הוא יחזיר לנו 2 רשומות.

id	age	gender	degree	firstName	lastName
111	21	1	1	Chava	Glass
222	28	1	3	Tal	Negev
NULL	NULL	NULL	NULL	NULL	NULL

הבעיה היא שאין אחיזה בטוחה בשביל להגיד איזה 2 רשומות נקבל.
כלומר במקרה זה, נבחרו 2 רשימות ללא תנאי מסוים.

```sql
SELECT *
FROM students
ORDER BY firstName LIMIT 2, 2
```

במקרה זה, הפקודה תצביע על הרשומה השניה ומשם היא תמשוך 2 רשומות.

id	age	gender	degree	firstName	lastName
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
NULL	NULL	NULL	NULL	NULL	NULL

למשל בתיבת מייל כשנרצה לדפדף במיילים שלנו ולהמשיך לעמוד הבא, אפשר להציג כל עמוד של 20 מיילים כל פעם.. ככה אפשר לקבל בקלות את הדפדוף הזה.

פונקציות אגרגטיביות

עד עכשיו שלפנו את הנתונים כמו שהם, לא המצאנו נתונים אלא הצגנו אותם כמו שהם (חוץ coalesce שהצגנו) הפונקציות האלה לוקחות את הנתונים שנשלוף ועושות איזושהי פעולה ומחזירה רשימה עבור הפעולה.

```sql
SELECT COUNT(*)
FROM students
```

הפונקציה מחזירה את מספר הרשומות

```sql
SELECT AVG(grade)
FROM grades
```

הפונקציה מחזירה את הממוצע ציונים בטבלה

```sql
SELECT SUM(passed)
FROM grades
```

הפונקציה סוכמת כמה עברו

```sql
SELECT MAX(grade)
FROM grades
```

הפונקציה מחזירה את הציון הגדול ביותר

SQL

```
SELECT MIN(grade)
FROM grades
```

הפונקציה מחזירה את הציון הנמוך ביותר

כדי לקבץ ערכים ספציפיים יחדיו ולחשב עליהם פונקציה מסוימת נשתמש ב- **GROUP BY .**

נתבונן בטבלה הבאה:

id	name	lecturer	year	semester
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 Programming	David Gol	2022	2
66	databases	NULL	2025	1

SQL

```
SELECT year, COUNT(year)
FROM courses
GROUP BY year
```

תחזיר כמה קורסים נלמדים בכל שנה:

year	count(year)
2020	1
2021	1
2022	3
2025	1

SQL

```
SELECT courseId, AVG(grade)
FROM grades
GROUP BY courseId
```

נסתכל על :

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

תחזיר לנו את ממוצע של כל קורס.

	courseId	AVG(grade)
	20	64.0000
	30	92.5000
	40	53.5000

SQL

```
SELECT studentId, MAX(grade)
FROM grades
GROUP BY studentId
```

תחזיר לנו את הציון הגבוה ביותר של כל סטודנט:

	studentId	MAX(grade)
	111	90
	222	85
	333	40
	444	95

הפקודה WHERE לא עובדת על GROUP BY, אז איך נגדיר תנאים או נסנן תוצאות בקבוצה?
שימוש ב-**HAVING** נותן לנו מענה למקרים אלה.
בדוגמה הבאה, כל קבוצה היא שנה כלשהי אבל רק השנים שמופיעות יותר מפעם אחת בטבלה:

id	name	lecturer	year	semester
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 Programming	David Gol	2022	2
66	databases	NULL	2025	1

SQL

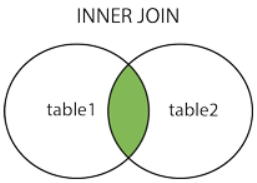
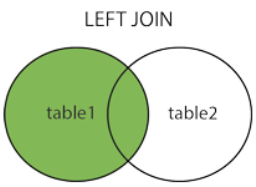
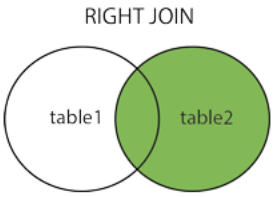
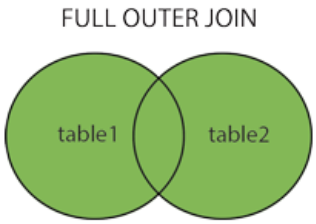
```
SELECT year, COUNT(year)
FROM courses
GROUP BY year
HAVING COUNT(year) > 1
```

ולכן נקבל את:

year	count(year)
2022	3

פקודות JOIN

- משתמשים ב- join בשביל לבצע שליפה מכמה טבלאות.
- השליפה מבוססת על העמודות הזהות בכל הטבלאות.

סוגי JOIN		
סוג	**הסבר**	**המחשה**
INNER JOIN	מבצע שליפה של כל הרשומות מ-2 הטבלאות כל עוד יש התאמה בין העמודות.	
LEFT JOIN	מבצע שליפה של כל הרשומות מהטבלה השמאלית (table 1) עם ההתאמה של השורות בטבלה הימנית (table2). אם אין התאמה יופיע בתוצאות הערך null.	
RIGHT JOIN	מבצע שליפה של כל הרשומות מהטבלה הימנית (table 2) עם ההתאמה של השורות בטבלה השמאלית (table1). אם אין התאמה יופיע בתוצאות הערך null.	
FULL JOIN	מבצע שליפה של כל הרשומות מהטבלה השמאלית (table 1) ואת כל הרשומות מהטבלה הימנית (table 2) **mySql לא תומך ב-Full Join**	

דוגמאות ל-JOIN

נתונות 2 טבלאות, משמאל טבלת "*Students*" ומימין טבלת "*StudentCourse*":

Students

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	xxxxxxxxxx	18
2	PRATIK	BIHAR	xxxxxxxxxx	19
3	RIYANKA	SILIGURI	xxxxxxxxxx	20
4	DEEP	RAMNAGAR	xxxxxxxxxx	18
5	SAPTARHI	KOLKATA	xxxxxxxxxx	19
6	DHANRAJ	BARABAJAR	xxxxxxxxxx	20
7	ROHIT	BALURGHAT	xxxxxxxxxx	18
8	NIRAJ	ALIPUR	xxxxxxxxxx	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

SQL

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE
FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

ונקבל:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

SQL

```
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

ונקבל:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

SQL

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

ונקבל:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

SQL **חשוב!- mySQL לא תומך בפקודה הזאת!**

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

ונקבל:

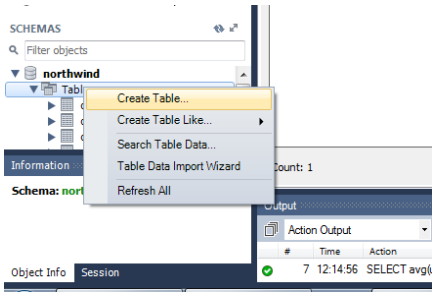
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

יצירת טבלה

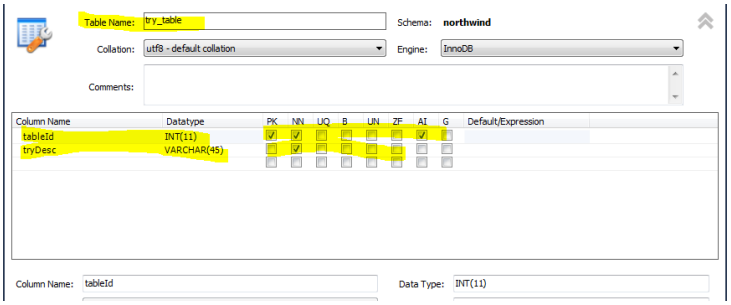
SQL

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

דרך ה-GUI



לאחר מכן יפתח חלון להגדרות הטבלה החדשה, פעולת הכנסת הנתונים לפי תנאים נקראת **Integrity Constraints:**



PK	ה-primary key, הוא המפתח הראשי לטבלה
NN	ה-Not Null מחייב את השדות להיות לא ערך ריק
UQ	ה-Unique Key מונע כפילויות של ערכים בעמודה
B	מקבל ערכים בינארים בלבד
UN	ה-unsigned number מקבל ערכים מספרים חיוביים בלבד בעמודה
ZF	ה-zero field ממלא את העמודה ב-0
AI	ה-Auto increment מכניס מספר בסדר עולה וסדרתי ללא כפילויות
G	ה-Generate מייצר ערך מספרי כלשהו (אקראי) וייתכן כפילויות

SQL	יצירת טבלה

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

פקודת ALTER TABLE

הם אוסף פקודות עבור טבלה (יצירה, מחיקה ושינוי).

SQL	הוספת עמודה

```
ALTER TABLE table_name
ADD column_name datatype;
```

SQL	מחיקת עמודה

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

SQL	עדכון עמודה

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

פקודת DROP TABLE

הם אוסף פקודות עבור מחיקת טבלה בכמה צורות.

SQL	מחיקת כל הטבלה

DROP TABLE table_name;
e.g- drop table try_table;

SQL	מחיקת נתוני טבלה, ללא מחיקת הגדרת הטבלה (תכונות הטבלה)

TRUNCATE table_name;
e.g- **TRUNCATE** try_table;

- כלומר, משאיר את הטבלה ריקה.

מילת המפתח CHECK

אילוץ ה- CHECK מגביל את הערך שניתן להציב בעמודה.

ה- SQL הבא יוצר אילוץ CHECK בעמודה "Age" בעת יצירת הטבלה "Persons". מגבלת ה- CHECK מבטיחה שלא יהיה לך אדם מתחת לגיל 18:

SQL	שימוש ב-CHECK על CREATE TABLE

```
CREATE TABLE Persons (
      Age int,
    CHECK (Age>=18)
);
```

כדי לאפשר מתן שמות לאילוץ CHECK ולהגדרת אילוץ CHECK במספר עמודות, נשתמש בתחביר SQL הבא:

SQL	שימוש ב-CHECK על CREATE TABLE

```
CREATE TABLE Persons (
      Age int,
      City varchar(255),
    CONSTRAINT CHECK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

כדי ליצור אילוץ CHECK בעמודה "Age" כאשר הטבלה כבר נוצרה, השתמש ב- SQL הבא:

SQL	שימוש ב-CHECK על ALTER TABLE

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

כדי להוריד מגבלת CHECK, השתמש ב- SQL הבא:

SQL	שימוש ב-CHECK על ALTER TABLE בשביל DROP

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

שאלת מבחן בהינתן טבלה:

```
ProductPrice(p_id, site_id, price, to_date)
```

SQL	מה תחזיר השאילתה הבאה?

```
SELECT MAX(p1.price - p2.price)
FROM ProductPrice as p1
JOIN ProductPrice as p2
ON p1.p_id = p2.p_id AND p1.site_id <> p2.site_id
WHERE p1.p_id = 18 AND p1.to_date is NULL
```

תשובה: הפרש המחירים של מוצר מס' 18 שלא פג תוקף באתרים השונים. בוחר את ההפרש הגדול ביותר.

שאלה

בהינתן 2 טבלאות:

```
Employee(EmpId, FullName, ManagerId)
EmployeeSalary(EmpId, Project, Salary)
```

SQL	מה תחזיר השאילתה הבאה?

```
SELECT  Salary
FROM EmployeeSalary
ORDER BY Salary DESC LIMIT 1, N
```

תשובה: תחזיר לנו את המשכורת ה-nth הגבוהה ביותר בטבלה.

קשר בין טבלאות - Foreign Key

ה-Foreign Key מגדיר את הקשר בין 2 רלציות (טבלאות).
אם נתבונן באיור משמאל אפשר לראות שהעמודה של studentId בטבלה grades זה בעצם ערכים שמגיעים מהטבלה students אז בעצם העמודה הזאת היא מהווה את הקשר בין הטבלאות grades ו-students.

אז Foreign Key זה בעצם שדה (לפחות אחד) בתוך טבלה מסוימת שבעצם מהווה Primary Key לטבלה אחרת.

students

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

courses

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

grades

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

כלומר, studentID הוא Foreign Key בטבלה grades שבעצם מהווה Primary Key לטבלה students.

ב-mySQL יש בתוך ההגדרות של הטבלה לשונית של 'Foreign Key' ובה אנחנו יכולים להגדיר אותו כרצוננו.

פקודת התנאי CASE

בדומה לשפות תכנות אחרות גם ב-SQL יש תנאים (כמו if, else…)

מבנה הקוד נראה כך:

SQL

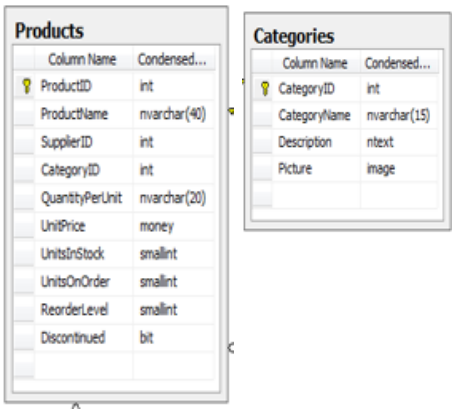
```
SELECT
       CASE
          WHEN condition_1 THEN value1
          WHEN condition_2 THEN value2
          ELSE value3
        END as alias
FROM table;
```

- אם לא נכתוב את ה- alias ב-END אז נקבל ששם העמודה תיהיה בעצם כל הקוד שתבנו ל-CASE (לא קריא).
- הפקודה CASE נכתבת אך ורק בתוך ה-SELECT .
- היא לצורך תצוגה בלבד, כלומר היא לא נשמרת בזיכרון כך אם במקרה נרצה להשתמש באיברים שקיבלנו בתנאי בהמשך הקוד אנחנו לא נוכל לגשת אליהם כי הם לא באמת קיימים.

כלומר בצורה כזאת:

SQL

```
SELECT A, B, CASE … END , C , D, CASE … END, E , F
FROM table;
```



שאלה

בהינתן 2 הטבלאות הבאות:

כתבו שאילתה המחזירה את כל המוצרים – Id, שם, ובמקום הקטגוריה יציג Yes אם המוצר שייך ל Beverages או Condiments ו No אחרת.

SQL

```
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
WHERE p.CategoryID = c.CategoryID
```

טבלאות זמניות- Temporary Table

טבלאות אלה נשמרות בזיכרון המערכת באופן זמני, כלומר בעת יציאה מהתוכנה (mySQL במקרה שלנו), הטבלה תימחק ולא יהיה ניתן לגשת אליה יותר.

שאלה

רשמו את חמש ההזמנות ששילמו בהם הכי הרבה כסף.

לכל הזמנה את שם הלקוח, תאריך ההזמנה ומי הספק והסכום ששילמו מסודרים מהסכום הגדול לקטן.

השתמשו ב- temporary table

SQL

```
CREATE TEMPORARY TABLE top_order AS
 (SELECT OrderID as ids, sum(UnitPrice*Quantity) as totalSum From `order details`
od GROUP BY OrderID ORDER BY totalSum desc  LIMIT 5 );
SELECT c.CompanyName, c.ContactName, o.OrderDate, s.CompanyName, t.totalSum
FROM orders o, shippers s, customers c, top_order t
 WHERE  c.CustomerID = o.CustomerID
AND s.ShipperID = o.ShipVia
AND o.OrderID= t.ids
ORDER BY totalSum desc;
```

משתנים - Variables

מסמנים משתנה בסימון @.

אנחנו יכולים להגדיר משתנה שנשמור אותו ואז יהיה אפשר להשתמש בו מאוחר יותר.

יש 2 אפשרויות נפוצות להגדרת משתנים:

SQL — **הצבה של ערך לתוך משתנה באמצעות SET:**

```
SET @passGrade = 60
```

SQL — **הצבה של משתנה מתוך SELECT, להכניס את הערך שחוזר מתוך ה-SELECT ישירות לתוך שדה:**

```
SELECT @avgGrade := AVG(grade)
FROM grades
```

שימוש ב- Aliases

השימוש ב-Alias מאפשר לתת כינוי (שם זמני) לטבלה או לכל גוף בטבלה באמצעות הפקודה "AS".
חשוב לשים לב כי אם אנחנו מתעסקים NESTED אז הגוף החיצוני לא יכיר את ה-Alias הפנימי ויכול לגרום לשגיאות.
הפתרון לבעיה זאת היא לתת Alias לגוף הפנימי וככה יהיה היכרות מלאה בין החיצוני לפנימי.

בקוד הזה, יהיה שגיאה כי ה-SELECT החיצוני לא מכיר את av (ה-Alias של הפנימי).

```sql
SELECT MAX(av)
FROM (
SELECT studentId, AVG(grade) AS av
FROM grades
GROUP BY studentId
);
```

הפתרון:

נשים AS t בסוף (לא חייבים להשתמש בו), זה מספיק כדי לקשר בין החיצוני לפנימי ולהכיר הכל:

```sql
SELECT MAX(av)
FROM (
SELECT studentId, AVG(grade) AS av
FROM grades
GROUP BY studentId
) AS t;
```

המושג TRANSACTIONS

תנועה (באנגלית: **transaction**) היא פעולה לוגית לשינוי נתונים המורכבת מסדרת פעולות בדידות.
כל הפעולות הבדידות המרכיבות אותה חייבות להתבצע כיחידה אחת, או לא להתבצע כלל. תכונה זו נקראת **אטומיות**.
היכולת להבטיח את שלמות הנתונים מכונה **ACID**.

לדוגמא, בפעולה של העברת כספים בין חשבונות בנק, חשוב שאם סכום כלשהו הוחסר מהחשבון המשלם הוא יופקד בחשבון הנמען. אם, בשל כשל כלשהו, תהליך העברת הכספים נקטע לפני שהושלם, המערכת עלולה להיוותר במצב לא עקבי.

לדוגמה, הסכום הוחסר אך לא הועבר. מטרת התנועה היא להבטיח את שלמות הנתונים, גם לנוכח מקרי כשל.

תנועה שלא בוצעה בשלמותה מבוטלת באמצעות הסגה (**Rollback**), כדי להחזיר את המערכת והנתונים למצב יציב שקדם לפעולה.
בסיום התנועה, היא הופכת לתנועה מקובעת (**Commited Transaction**).
תנועה יכולה להתבטל באמצעות הסגה (**Rollback**) כל עוד לא קובעה.

תנועת SQL פשוטה במסד נתונים מתבצעת לרוב באופן הבא:

1. התחלת התנועה. (**START TRANSACTION**).
2. הרצת מספר שאילתות. עדכוני נתונים שמבצעת התנועה אינם נראים עדיין לתנועות אחרות.
3. קיבוע התנועה (**COMMIT**) . אם התנועה הצליחה, עדכונים שנעשו במהלכה נראים כעת לתנועות אחרות.
4. במקרה של כשל באחת מהשאילתות, עשוי מסד הנתונים להסיג (**Rollback**) את התנועה כולה או רק את השאילתה שנכשלה. ההתנהגות המדויקת תלויה בהגדרות נוספות המתלוות לתנועה.
5. כל עוד התנועה לא התקבעה, ניתן להסיגה.

דוגמה ל-TRANSACTION | **SQL**

```
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;
```

בדוגמה מלמעלה, נשים לב לפקודות המסומנות באדום.

מה שקורה בדוגמה הוא שמהרגע שיש את START TRANSACTION הקוד מחשב טרנזקציה, הוא מבצע את כל הפקודות מתחתיו, שום דבר אחר לא יכול לקרות בטבלאות האלה באותו הזמן כי אחרת זה עלול לפגוע בנתונים.

ברגע שמגיע את ה-COMMIT זה ישחרר את כל הטבלאות וגם מבצע את כל השינויים שעשינו ביחד.

פקודת STORED PROCEDURE

אפשר לשמור פקודות SQL שכתבנו בעזרת פקודת STORED PROCEDURE ובהרצת הפקודה הזאת שנגדיר נקבל את מה ששמרנו בעבר (כמו סוג של פונקציה משלנו).
הם פונקציות ששמורות בתוך ה-DATABASE שלנו.

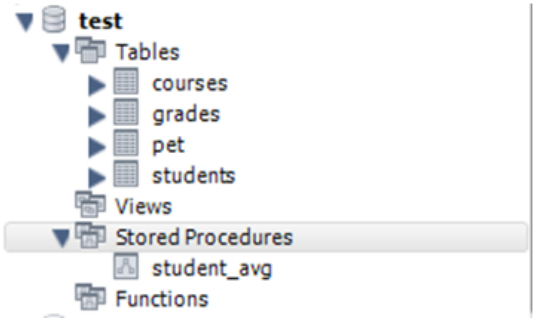
SQL

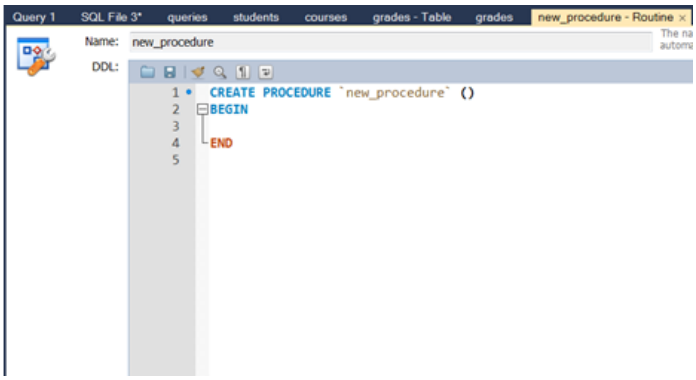
```
DELIMITER $$
CREATE PROCEDURE student_avg (IN stId INT)

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

DELIMITER ;
```

ובעת הרצת ה-"פונקציה" `procedure_name` נקבל את מה שכתבנו בין ה-BEGIN לבין ה-END.

התפקיד של **ה-DELIMITER** הוא לשנות את הסימון סיומת הפקודה כדי למנוע שגיאות.
בשפת ה-SQL הדיפולט לסיומת פקודה היא מסומנת ב-' ; ' .
כאשר נעבוד בפקודות גדולות אנחנו יכולים להגיע למצב שאנחנו רוצים לסגור פקודה אבל בטעות סגרנו פקודה אחרת שמוכלת בתוכה.
למשל בדוגמה מלמעלה, אם לא היינו מגדירים את הסיומת הזמנית להיות ' $$ ' אז הקוד היה מסתיים איפה שמסומן בכחול, זאת שגיאה מכיוון שאנחנו אמורים לסיים ב-END.
לכן נשנה זמנית את הסיומת להיות ' $$ ' , אפשר לשנות גם לאיזה סימן אחר שנרצה כל עוד הוא לא שמור במערכת.
ואז בסוף הקוד נחזיר את סימון הסיומת חזרה לדיפולט ' ; ' כדי להמשיך לעבוד בשפה הרשמית והמקובלת כדי להימנע משגיאות נוספות.

אפשר ליצור בקלות STORED PROCEDURE ב-mySQL:





תפיסת שגיאות ב-SQL

כדי לתפוס שגיאות ב-SQL נסתכל בדוגמה מלמטה והעיקר על הסימון האדום:

SQL

```
CREATE PROCEDURE `transfer_balance`(
       IN sender INT,
       IN receiver INT,
       IN trAmount REAL)
BEGIN

   DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING
   BEGIN
       ROLLBACK;
       SELECT 'Error occurred' as Message;
   END;

   START TRANSACTION;
   SELECT @sBl = amount FROM bankBalances WHERE userId = sender;
   UPDATE bankBalances SET amount = @sBl - trAmount WHERE userId = sender;
   SELECT @rBl = amount FROM bankBalances WHERE userId = receiver;
   UPDATE bankBalances SET amount = @rBl  + trAmount WHERE userId = receiver;
   COMMIT;

END;
```

לגבי הקוד מלמעלה:

החלק של START TRANSACTION הוא בעצם העברת כספים באופן בטוח מחשבון אחד לאחר.

יש לנו CREATE PROCEDURE בשם 'transfer_balace' והוא מקבל 3 פרמטרים (ממש כמו פונקציה בשפות אחרות).

החלק באדום מטפל בשגיאות, אם קרתה איזושהי שגיאה נרצה לתפוס את השגיאה, אפשר להגיד HANDLER שתפקידו לתפוס את השגיאות ונגדיר שיתפוס שגיאות מסוג SQLEXCEPTION ו-SQLWARNING , כלומר שיתפוס את כל סוגי שגיאות ואזהרות של שפת ה-SQL.

תפקיד ה-ROLLBACK קשור ל-START TRANSACTION ותפקידו לגלגל אחורה את כל השינויים במקרה של תקלה כלשהי כדי למנוע מצב של אי-אחידות זאת כדי להבטיח את קיום עקרון הטרנזקציה והאמינות שלה במקרה הגרוע.

תפקיד ה-SELECT 'ERROR OCCURRED' הוא להדפיס את המשפט הזה ברגע השגיאה.

(פשוט נקבל עמודה אחת ורשומה אחד כך שבתוך הרשומה יופיע את המשפט שהגדרנו).

פקודת TRIGGER

Triggers הם תוכניות מאוחסנות, אשר מבוצעות אוטומטית או מופעלות כאשר אירועים מסוימים מתרחשים.
הטריגר בעצם מעדכן נתונים בצורה אוטומטית.
הטריגר קורה כל הזמן, זה סוג של תזמון של פונקציה, כלומר בשונה מ-Stored Procedure בו אנחנו משתמשים בעת הצורך, הטריגר עובד בתזמון אוטומטי ורץ לבד בלי לקרוא לו.
את הטריגר אפשר ליצור ב-workbench ב-generator פשוט וקל או במימוש כתיבת קוד ואותו נשמור בלשונית שלו במסד הנתונים בשם triggers.

SQL | **דוגמה ליצירת Trigger חדש - מעדכן את ממוצע הציונים של כל סטודנט לאחר הכנסת ציון חדש**

```
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 ;
```

אפשר להשתמש ב-BEFORE במקום ה-AFTER בהתאם לצורך.

SQL | **מחיקת ה-Trigger**

```
DROP TRIGGER new_grade_received;
```

שימוש ב-Views

ה-View היא לא באמת טבלה שנשמרים בה נתונים, היא בעצם "טבלת ראי" שמשקפת טבלה אחרת עם נתונים אמיתיים.
ה-View שימושי למידור של נתונים למשתמש, כלומר הוא יכול להגביל גישה של נתונים עבור משתמש.

SQL | **דוגמה ליצירת View חדש**

```
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;
```

עדכון ה-VIEW

לעדכן View זה דבר מורכב ויכול לגרום לשגיאות רבות.
בפועל, אי אפשר לעדכן אותו אבל קיימים מקרים בעזרת מניפולציות מסוימות.
ה-View הוא בעצם מעין מצביע לנתון אמיתי שלא ניתן לשנות אותו מה-View, כלומר זה כמו שנפתח אתר באינטרנט ונקבל טקסט, אנחנו לא יכולים לשנות אותו, הוא פשוט מופיע שם לקריאה וזה אותו הדבר.
לכן, כדי לעדכן את ה-View יש לגשת "לאבא" שלו, במקרה הזה מדובר בטבלה ממה הוא משתמש .
רק אחרי שנשנה את הטבלה של האב, ה-View "יתעדכן" כלומר יראה את השינוי שבוצע מעליו.
ל-mySQL יש יכולת לעקוף את זה כך שממש אפשר לעדכן את ה-View אך זה יקרה אך ורק במידה ול-View יש את אותו השדה בעל אותו השם שיש לאבא שלו, אחרת נקבל שגיאה.

אלגברה רלציונית

- אלגברה יחסית (רלציונית) מגדירה פקודות דומות לאלו שנמצאות ב- SQL (לצערנו עם שמות שונים).
- לכל פקודה יש סמל.
- זה התאוריה שמאחורי ה-SQL, אלגברה רלציונית מגדירה את הפקודות שנכתבו ב-SQL לפי סימנים ולוגיקה.

Relational Algebra	SQL
σ	Where
π	Select
ρ	As
$\cup$	Union
$\cap$	In
$---$	Not In
$\div$	Division
X	Outer Join
$\bowtie$	Inner Join
$\bowtie$	Full Outer Join
$\bowtie$	Right Outer Join
$\bowtie$	Left Outer Join
λ	Group By

צירוף טבעי (Natural Join)

טבלת R		
id	**lastName**	**firstName**
12345	Smith	Jane
67899	Cohen	Tamar

טבלת S		
id	**Street**	**City**
12345	35 Elm	Chicago
67899	15 Herzl	Haifa

נציג בצורה הבאה:

$$\Pi_{(id,\, lastName,\, firstName,\, Street,City)}(\sigma_{(id)}(S) \bowtie R)$$

$S \bowtie R$				
id	**lastName**	**firstName**	**Street**	**City**
12345	Smith	Jane	35 Elm	Chicago
67899	Cohen	Tamar	15 Herzl	Haifa

SQL | **דוגמה לייצוג SQL באלגברה רלציונית**

```
SELECT firstName, id
FROM students
WHERE degree > 1
```

נציג בצורה הבאה:

$$\Pi_{(firstName,\, id)}(\sigma_{(degree>1)}(students))$$

SQL

```
SELECT students.lastName, students.id, grades.grade
FROM students RIGHT JOIN grades
ON students.Id=grades.studentId
WHERE grades.courseId=20;
```

נציג בצורה הבאה:

$$\Pi_{(lastName,\, id,\, grade)}(\sigma_{(courseId=20)}\,(students \ltimes_{(id=studentId)} grades)\,)$$

דוגמאות

נתונות הטבלאות (ישויות הבאות):

טבלה	מפתח
$Person(name,\ age,\ gender)$	$name\ is\ a\ key$
$Serves(pizzeria\ ,\ pizza\ ,\ price)$	$(pizzeria,\ pizza)\ is\ a\ key$
$Frequents(name\ ,\ pizzeria)$	$(name,\ pizzeria)\ is\ a\ key$
$Eats(name\ ,\ pizza)$	$(name,\ pizza)\ is\ a\ key$

RA	מצא פיצריות המבוקרות לפחות ע"י מבקר אחד בן פחות מ-18.
$\Pi_{(\ pizzeria)}(\sigma_{(age < 18)}(Person) \bowtie (Frequents)\ )$	

RA	מצאו את שמות כל הנשים שאכלו פיצה פטריות או פפרוני (או שניהם)
$\Pi_{(name)}(\ \sigma_{gender = 'female' \wedge (pizza='mushroom' \vee\ pizza='pepperoni')}(Person \bowtie Eats)\ )$	

RA	מצאו את שמות כל הנשים שאכלו פיצה פטריות <u>וגם</u> פפרוני
$\Pi_{(name)}(\ \sigma_{gender = 'female' \wedge pizza='mushroom'}(Person \bowtie Eats)\ )\ \cap$ $\Pi_{(name)}(\ \sigma_{gender = 'female' \wedge pizza='pepperoni'}(Person \bowtie Eats)\ )$	

RA	מצאו את כל האנשים שהולכים לכל הפיצריות שמגישים בהם לפחות פיצה אחת שהם אוהבים
$\Pi_{(name)}(Person)\ \ ---\ \Pi_{(name)}(\ \Pi_{(name.pizzeria)}(Eats \bowtie Serves)\ ---\ Frequents)$	

כאשר הכיוון הימני הוא כל האנשים שלא הלכו לפיצריות בהם מגישים פיצות שהם אוהבים לאכול.
הכיוון השמאלי הוא כל האנשים בכללי.
לכן עבור כל האנשים נוריד את כל המקרים בהם לא הלכו לפיצריות בהם מגישים פיצות שהם אוהבים לאכול.

נתונות הטבלאות (ישויות הבאות):

טבלה	מפתח
$Serves\ (S\_Name,\ L\_Num,\ Direction,\ Km)$	$(S\_Name,\ L\_Num,\ Direction)$
$Arrives\ (T\_Num,\ S\_Name,\ L\_Num,\ Direction\ Platform,\ D\_Time\ ,A\_Time)$	$(T\_Num,\ S\_Name,\ L\_Num,\ Direction)$
$Station\ (S\_Name,\ Height)$	(S_Name)
$Station\_Type\ (S\_Name,\ S\_Type)$	$(S\_Name,\ S\_Type)$

שמות התחנות המשרתות יותר מקו אחד	RA
$\pi_{(S\_Name)}(\sigma_{(S\_Name=S)\ \wedge\ ((L\_Num\neq L)\vee(Direction\neq D))}(\rho_{S\_Name\to S,\ L\_Num\to L,Direction\to D}(Serves)\ X\ Serves$	

לא נרצה להחשיב כיוונים שונים של אותו קו	RA
$\pi_{(S\_Name)}(\sigma_{(S\_Name=S)\ \wedge\ (L\_Num\neq L)}(\rho_{S\_Name\to S,\ L\_Num\to L,Direction\to D}(Serves)\ X\ Serves))$	

נורמליזציה

נורמליזציה היא התהליך של ארגון נתונים במסד נתונים ומטרתה:

- ארגון הנתונים בצורה נכונה ויעילה ככל היותר
- להפוך את מסד הנתונים לגמיש יותר
- כללים שנועדו להגן על הנתונים
- נרצה לדרג את מסד הנתונים לפי איכות
- ניצול וניהול טוב לפי סולם מוכר לעולם

כדי שנוכל לדבר על רמת נורמליזציה צריך להכיר 3 מושגים:

1. **Candidate key** - זה אוסף של שדות שברגע שיש לנו אותם אנחנו יכולים לגשת לכל שאר השדות בבסיס הנתונים. זה קבוצה מינימלית (לא ניתן לצמצם אותה) ואפשר עדיין להגיע ממנה לכל השדות.
2. **Super key** - זה קבוצה שמכילה את **Candidate key**, גם בהינתן קבוצה זו נוכל להגיע לכל השדות.
   ה-**Super key** יכול להכיל ערכים נוספים לעומת ה-**Candidate key** ולכן היא לאו דווקא מינימלית.
3. **None prime** - הם כל השדות שלא שייכים לאף **Candidate key**.

הגדרות לנרמול

1. **תלות פונקציונלית בין תכונות:** חלק מהמאפיינים תלויים אחד בשני.
   נסמן $A \rightarrow B$ נאמר ש-B תלוי ב-A או נאמר ש-A קובע את B.
   אם $B \subseteq A$ אז ברור ש-B תלוי ב-A כי הוא תלוי בתוכו.
2. **מפתחות - Candidate Key**
   - אוסף מינימלי של תכונות שיכולות להגדיר לי כניסה ליישות.
     הכוונה למינימליות אומר שלא הכנסנו נתון מיותר לתוך הטבלה שלנו.
   - יכול להיות שבטבלה מסוימת יהיה יותר מ **Candidate Key** אחד.
3. **Prime or Non-Prime**
   - **Prime** - תכונה ששייכת לקבוצה מה-**Candidate Key**
   - **Non-Prime** - תכונה שלא שייכת לקבוצה מה- **Candidate Key**
4. **Super Key**
   - סט של תכונות שמגדיר לי כניסה ליישות
   - דומה ל-**Candidate** רק שכאן אין התייחסות למינימליות.

בעיות ביחסים

1. **Searchability** - נגישות לא נכונה למידע בטבלה
2. **Redundancy** - עודף של מידע

הנירמול מטפל בבעיות היחסים האלה ובעיות נוספות.

דוגמה להבדלים בין Prime לבין Non-Prime

בהינתן הטבלה הבאה:

ID	NAME	AGE	CITY	PHONE_ NO
66	Alex	21	Noida	9535653232
67	Luke	22	Delhi	8353355366
68	Haley	19	Noida	7323598665
69	Luke	21	Mumbai	8656864365

נבדוק עבור כל עמודה/אטריביוט אם היא **Prime** או **Non Prime**?

- עבור **ID** - כל רשומה יהיה **ID ייחודי** משלו לכן זה **Prime attribute**.
- עבור **NAME** - לשני אנשים שונים **יכול להיות את אותו השם הפרטי**, לכן זה **Non-Prime attribute**.
- עבור **AGE** - לשני אנשים שונים **יכול להיות את אותו הגיל**, לכן זה **Non-Prime attribute**.
- עבור **CITY** - שני אנשים שונים **יכולים לגור באותו העיר**, לכן זה **Non-Prime attribute**.
- עבור **PHONE_NO** - לשני אנשים שונים **לא יכול להיות את אותו מספר הטלפון**, לכן זה **Prime attribute**.

רמות נרמול

לנורמליזציה יש כמה רמות נרמול $NF = Normalized\ Form$:

1. **1NF** - כל תכונה (כל עמודה) צריכה להכיל ערך **אטומי** ויחיד (Searchability).
   - הרמה הכי בסיסית שטבלה צריכה לעמוד בה, כל שדה בטבלה חייב להיות שדה אטומי ולא מחושב.
   - לכל שדה (כל עמודה) יש תפקיד אחד ויחיד.
   - ללא שדות מחושבים, כלומר שלא יהיה בטבלה גם שנת לידה וגם גיל - אחד מהם מיותר כי לאחד מהם אפשר להגיע על ידי חישוב של השני.
2. **2NF** - הטבלה חייבת להיות 1NF , בנוסף נרצה שכל התכונות שהן **Non-Prime** אסור שהם יהיו תלויות ממש בתת הקבוצה של ה-**Candidate Key**.
3. **3NF** - הטבלה חייבת להיות 2NF. שכל ה-**Non-Prime** יהיו תלויים רק ב-**Super Key**.
   בנוסף, **אסור** שתהיה תלות טרנזיטיבית.
4. **BCNF (3.5NF)** - כל 2 סטים X , Y כשאחד תלוי בשני, אם $X \rightarrow Y$ אז X חייב להיות **Super Key**.
   תכונה שהיא **Prime** חייבת להיות תלויה ב-**Super Key**.
5. **4NF** - הטבלה חייבת להיות BCNF וגם אסור שיהיו כפילויות של נתונים בין כמה רשומות, לכן במידה ויש כפילות ניקח את התכונה שגורמת לכפילות וניצור לה טבלה חדשה ונפרדת.

נסביר בדוגמא את כל רמות הנרמול, נתונה טבלת מכירות של מוצרי Xbox ו- Playstation:

Cust Name	Item	Shipping Address	Newsletter	Supplier	Supplier Phone	Price
Alan Smith	Xbox One	35 Palm St, Miami	Xbox News	Microsoft	(800) BUY-XBOX	250
Roger Banks	PlayStation 4	47 Campus Rd, Boston	PlayStation News	Sony	(800) BUY-SONY	300
Evan Wilson	Xbox One, PS Vita	28 Rock Av, Denver	Xbox News, PlayStation News	Wholesale	Toll Free	450
Alan Smith	PlayStation 4	47 Campus Rd, Boston	PlayStation News	Sony	(800) BUY-SONY	300

כללים לרמת נרמול NF1

1. כל תא חייב להכיל ערך אחד בלבד.
2. הערכים בעמודה זהים.
3. השורות חייבות להיות בעל מזהה ייחודי - הוספת ID ייחודי, או הוספת עמודות נוספות כדי ליצור ייחודיות.

תיקון לפי כל סעיף ברמה

1. הבעיה: נשים לב שעבור Evan Wilson יש 2 ערכים בתא אחד בעמודה **Item** ו-**Newsletter**.
   הפתרון: נפריד ביניהם באמצעות הוספה של רשומה נוספת עבור Even Wilson כדי לאפשר ערך אחד בכל תא.
2. הבעיה: עבור Evan Wilson נראה כי בעמודה **Supplier** ו- **Supplier Phone** נמצאים ערכים רנדומלים.
   הפתרון: נחליף את הערכים Wholesale ו- Toll Free בערכים תקניים ורלוונטים.
3. הבעיה: אין לכל לקוח/שורה מספר מזהה ייחודי משלו.
   הפתרון: נוסיף עמודה חדשה שתכיל את כל ה-**Primary Key** שהם יהיו ה-ID של כל לקוח.

Primary Key

Cust ID	Cust Name	Item	Shipping Address	Newsletter	Supplier	Supplier Phone	Price
at_smith	Alan Smith	Xbox One	35 Palm St, Miami	Xbox News	Microsoft	(800) BUY-XBOX	250
roger25	Roger Banks	PlayStation 4	47 Campus Rd, Boston	PlayStation News	Sony	(800) BUY-SONY	300
wilson44	Evan Wilson	Xbox One	28 Rock Av, Denver	Xbox News	Microsoft	(800) BUY-XBOX	250
wilson44	Evan Wilson	PS Vita	28 Rock Av, Denver	PlayStation News	Sony	(800) BUY-SONY	200
am_smith	Alan Smith	PlayStation 4	47 Campus Rd, Boston	PlayStation News	Sony	(800) BUY-SONY	300

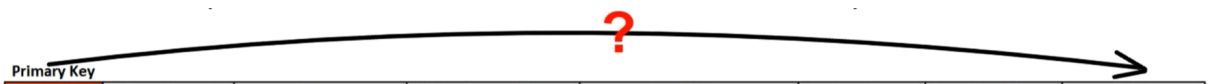
נמשיך לרמת נרמול הבאה עם הטבלה החדשה...

כללים לרמת נרמול NF2

חייבת להיות NF1 ושכל התכונות/העמודות (עמודות שהן לא מפתח כלומר האפורות) תלויות **במפתח** (כלומר העמודה הכתומה).
נרצה שכל התכונות שהן **Non-Prime** אסור שהם יהיו תלויות ממש בתת הקבוצה של ה-**Candidate Key**.

תיקון לפי הכללים של הרמה

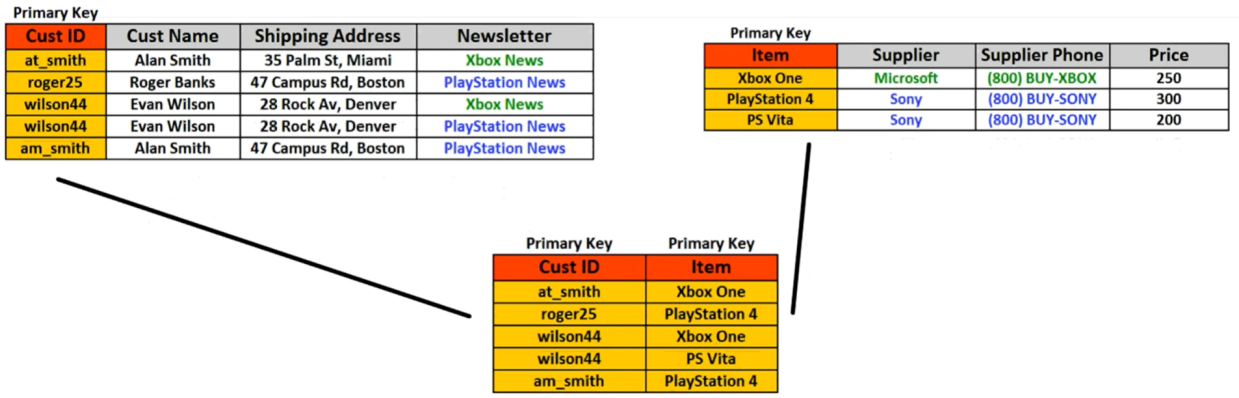
בעיה: המחיר לא תלוי במי שקנה את המוצר, כלומר ה-**Price** לא תלוי ב- **Cust ID**, אלא ב-**Item**.
או במילים אחרות, ה-**Price** שהוא **Non-Prime** לא תלוי ממש בתת הקבוצה של ה-**Candidate Key** שהוא **Cust ID** כפי שמופיע בטבלה.



Primary Key

Cust ID	Cust Name	Item	Shipping Address	Newsletter	Supplier	Supplier Phone	Price
at_smith	Alan Smith	Xbox One	35 Palm St, Miami	Xbox News	Microsoft	(800) BUY-XBOX	250
roger25	Roger Banks	PlayStation 4	47 Campus Rd, Boston	PlayStation News	Sony	(800) BUY-SONY	300
wilson44	Evan Wilson	Xbox One	28 Rock Av, Denver	Xbox News	Microsoft	(800) BUY-XBOX	250
wilson44	Evan Wilson	PS Vita	28 Rock Av, Denver	PlayStation News	Sony	(800) BUY-SONY	200
am_smith	Alan Smith	PlayStation 4	47 Campus Rd, Boston	PlayStation News	Sony	(800) BUY-SONY	300

פתרון: יש להפריד ל-2 טבלאות שונות, טבלת פרטי לקוח וטבלת פרטי מוצר, בנוסף צריך טבלה נוספת שהמטרה שלה היא "צומת הדרכים" או "הטבלה המקשרת" בין 2 הטבלאות, שם ישבו ה-**Primary Key** של 2 הקבוצות.



נמשיך לרמת נרמול הבאה עם הטבלאות החדשות...

כללים לרמת נרמול NF3

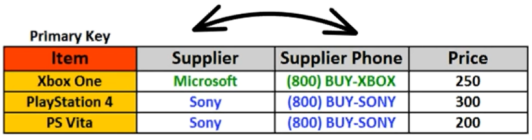
חייב להיות 2NF וניתן לקבוע את כל העמודות רק על ידי המפתח בטבלה ולא על ידי עמודה אחרת.

נרצה שכל ה-**Non-Prime** יהיו תלויים רק ב-**Super Key**.

תיקון לפי הכללים של הרמה

<u>בעיה</u>:

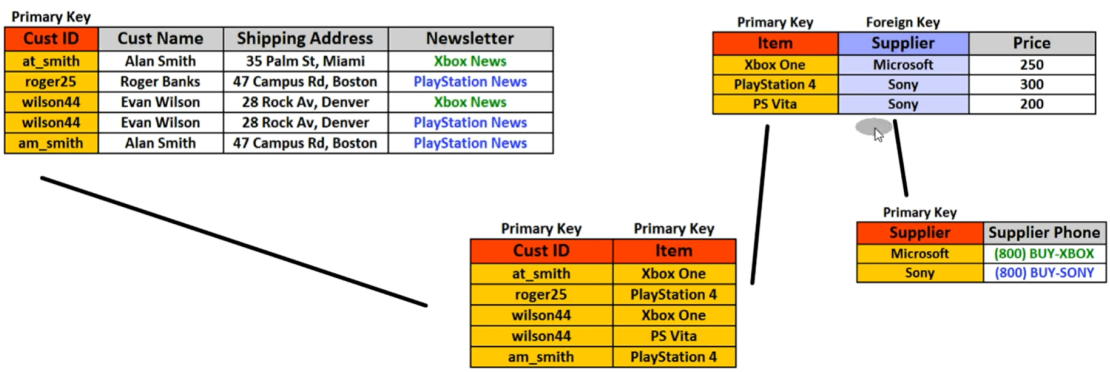
1. הספק ומספר הטלפון של הספק תמיד מגיעים ביחד בטבלה.
2. המספר של Sony לא יהיה קשור לעולם לספק של Microsoft וגם להפך.
3. יש חזרות של אותם מספרי טלפון וזה לא טוב, כלומר אם נוסיף עוד מוצרים של Sony החזרות רק יגדלו לרעתנו.
4. מה שבעיקר לא מתקיים כאן זה ש-**Supplier Phone** שהוא **Non-Prime** לא תלוי ב-**Super Key** אלא תלוי ב-**Non-Prime** אחר שהוא בעצם ה-**Supplier**.



<u>פתרון</u>: נרצה להפוך את **Supplier** ל-**Super Key** כך ש-**Supplier Phone** שהוא **Non-Prime** יהיה תלוי בו.

לכן נפריד את הטבלה הימנית למעלה ל-2 טבלאות שונות: טבלת מספרי טלפון של הספקים וטבלת מוצרים.

בטבלת מספרי הטלפון ה-**Primary Key** שלו יהיה ה-**Foreign Key** של טבלת המוצרים הכוונה שבעמודה **Supplier** בטבלת המוצרים יהיה אפשר להכניס רק ערכים שהם ה-**Primary Key** בטבלת מספרי הטלפון.



נמשיך לרמת נרמול הבאה עם הטבלאות החדשות...

כללים לרמת נרמול NF3.5

תכונה/עמודה שהיא **Prime** חייבת להיות תלויה ב-**Super Key**, כלומר לכל $X \rightarrow Y$, אז Xהוא **Super Key**.

תיקון לפי הכללים של הרמה

נשים לב שאנחנו עומדים בתנאים.

בכל הטבלאות אין שדות שהן **Prime** מלבד המפתחות עצמם ולכן מתקיים.

הטבלה היחידה שיש בה **Prime** שהיא לא מפתח זאת טבלת מספרי הטלפון, **Supplier Phone** הוא **Prime** כי לכל חברה יש מספר טלפון משלה, והיא תלויה ב-**Supplier** שהוא **Super Key** ולכן הכל תקין.

כללים לרמת נרמול NF4

ללא תלות רב ערכית

תיקון לפי הכללים של הרמה

<u>בעיה</u>: ה-**Newsletter** , **Shipping Address** , **Cust Name** תלויים כל אחד מהם בפרט ב- **Cust ID**.

כלומר הם כולם תלוים במפתח הראשי.

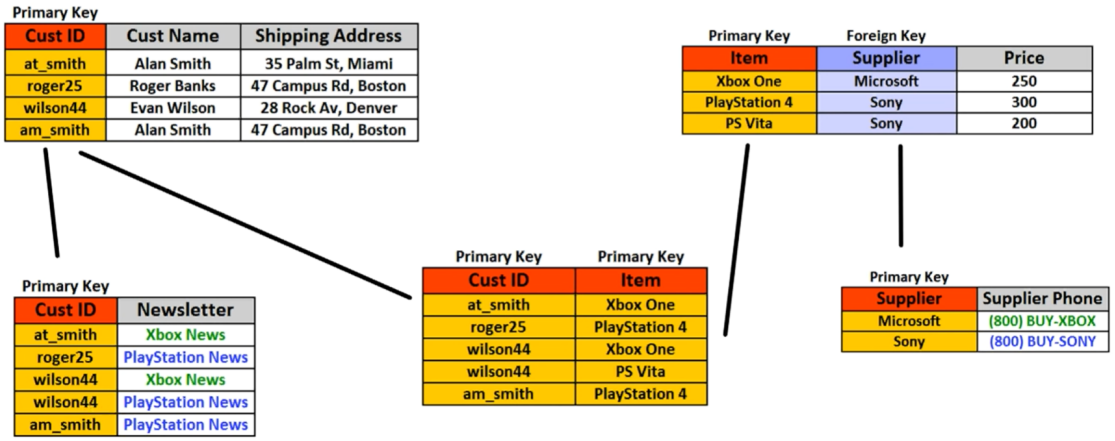
למרות זאת עדיין יש כאן בעיה:

1. עבור Evan Wilson מופיע כאן פעמיים עם אותו כתובת משלוח כי הוא קנה 2 מוצרים שונים לכן, אם הוא היה משנה את הכתובת שלו אנחנו נצטרך לשנות את הכתובת שלו ב-2 רשומות שונות.
2. עבור המשתמש am_smith למשל אם הוא יקנה מוצר נוסף של Xbox אז אנחנו ניצור רשומה חדשה עם אותו המפתח והשם של הלקוח והכתובת שלו יופיעו פעמיים.

- תלות רב ערכית הוא מצב בו עמודה אחת למשל **Newsletter** יכולה להכיל כמות שונה של ערכים לעומת שאר העמודות בטבלה.

<u>פתרון</u>: נפריד לטבלאות: טבלת פרטי לקוח וטבלת הזמנות.

עכשיו אפשר לראות כי בטבלת פרטי הלקוח, כל לקוח מופיע רק פעם אחת בלבד, וגם למשל wilson44 מופיע פעמיים בטבלת ההזמנות וזה בסדר כי לקוח יכול להזמין כמה מוצרים.



תרגיל בסיסי לנורמליזציה

עבור $R(A, B, C, D, E)$ עם היחסים הבאים:

$$AB \rightarrow CD$$
$$D \rightarrow A$$
$$BC \rightarrow DE$$

מצאו את ה-Candidate Key ואת רמת הנרמול:

$A \rightarrow$ ✘
$AB \rightarrow ABCD \rightarrow ABCDE \rightarrow$ ✔

כלומר, דרך AB הגענו לכל שאר הגופים.

$AC \rightarrow$ ✘
....
$BC \rightarrow BCDE \rightarrow BCDEA \rightarrow$ ✔
$BD \rightarrow BDA \rightarrow BDAC \rightarrow BDACE \rightarrow$ ✔
....

ולכן ה- Candidate Key הוא: $\{AB\}$, $\{BC\}$, $\{BD\}$

מכיוון ש-E לא נמצא כלל בצד ימין של היחסים נציין כי הוא Non-prime.

עבור רמת הנרמול נניח מראש כי מתקיים $1NF$ מראש.
כדי לבדוק האם $2NF$ ניגש אל ה-non prime שלנו E, ונבדוק שאין non-prime לתת קבוצה של candidate key.
לכן נחפש איפה מופיע E כלומר: $BC \rightarrow DE$ ונשאל האם הוא נמצא גם מימין לתת הקבוצה של ה-candidate key?
מאחר ו-BC הם לא תת-קבוצה של ה-candidate key ולכן E לא נמצא מימין ולכן $2NF$ מתקיים.
עבור $3NF$ נבדוק האם כל ה-non prime נמצאים מימין ל-super key, לכן נשאל האם E נמצא מימין ל-super key והתשובה היא כן כיוון ש-BC הוא super key.

לכן רמת נרמול הסופית עבור התרגיל היא $3NF$.

סיכום רמות נרמול

רמה	צריך להתקיים ש:	דוגמה - לא ייתכן ש:
NF1	יש רק ערך אחד בכל תא וגם אין 2 ערכים שונים באותה העמודה.	1. באותה העמודה יהיה הגיל וגם הכתובת. 2. יהיו 2 כתובות מגורים באותו התא.
NF2	$A \rightarrow NonPrime$ כאשר A **לא** תת קבוצה של ה-$Candidate$. **או** שאין לנו בכלל $NonPrime$ ולכן מתקיים באופן ריק.	נתונה קבוצת ה- Candidate: $\{U, Z, X\}$ אז עבור V $\{U, Z\}$ כאשר (V הוא $NonPrime$) נראה ש-$\{U, Z\}$ הוא תת קבוצה של המפתחות.
NF3	$A \rightarrow NonPrime$ כאשר A **כן** תת קבוצה של ה-$Super\ Keys$. **או** שאין לנו בכלל $NonPrime$ ולכן מתקיים באופן ריק.	עבור V $\{U, Z\}$ כאשר (V הוא $NonPrime$) לא ניתן לגרור מ-$\{U, Z\}$ לכל שאר התכונות - כלומר $\{U, Z\}$ הוא לא $Super\ Key$.
NF3.5	$A \rightarrow Prime$ כאשר A **כן** תת קבוצה של ה-$Super\ Keys$.	עבור V $\{U, Z\}$ כאשר (V הוא $Prime$) לא ניתן לגרור מ-$\{U, Z\}$ לכל שאר התכונות - כלומר $\{U, Z\}$ הוא לא $Super\ Key$.
NF4	**אסור** שיהיה תלות רב-ערכית	נתונות הרשומות של שם, מאכל אהוב ותחביב: בני | המבורגר | כדורגל בני | פיצה | גיימר אז התלות רב ערכית היא שאפשר ליצור רשומות: בני | פיצה | כדורגל בני | המבורגר | גיימר (כי למשל אם בני אוהב כדורגל וגם פיצה אז אפשר ליצור מזה רשומה שכזאת שנוצרה בגלל תלות לפי רשומות קודמות).

התחברות ל-mySQL בשפת JAVA

ה-mySQL הוא בעצם שרת (server) ולו יש מלא לקוחות (clients) שיכולים לגשת אליו באופן כמעט אוניברסלי כמו למשל JAVA, PYTHON, WORKBENCH ועוד כמה..

JAVA	דוגמה לשימוש בסיסי ב-sql בשפת JAVA

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

משמעויות המימוש

- `import java.sql.*;` תפקידו לקרוא לספרייה
- `Class.forName("com.mysql.jdbc.Driver");` קריאה דינמית לדרייבר וחובה למימוש
- `Connection` - התחברות לשרת שמקבל את שם המשתמש, הסיסמה וכתובת השרת המתאים
- `Statement` - זה כמו ליצור תיבת חדשה לשאילתה
- `executeQuery` - מקבל את השאילתה ומחזירה את הבא
- ResultSet - שהתקבל בחזרה של השאילתה הוא בעצם מחזיק את הישות שהתקבלה מהשאילתה
    - התוכנית תדפיס את כל הישות במעבר על מטריצה 2D כמו 2 לולאות for רגילות בשפת JAVA.

המימוש שהוצג כאן לא מספיק לקימפול

כדי לקמפל צריך קודם להתקין את ה-Connector לפרויקט ב-JAVA ב-2 אפשרויות:

1. התקנות ה-Connector בעזרת Gradle (מאוד קל וידידותי).
2. הורדת קובץ JAR וטעינה שלו לספריית הפרויקט.

Prepare Statement

זה בעצם יכול ליצור אינטראקציה בין המשתמש ל-DB.
נניח וקיימת תוכנה/אפליקציה ידידותית בבית החולים בה אחות מכניסה מספר תעודת זהות של מטופל כלשהו,
התוכנה תחזיר פלט של כל הנתונים הדרושים עבור המטופל לפי תעודת הזהות שלו.
ה-Prepare Statement הוא אחד השיטות לאפשר פעולה שכזאת ועוד הרבה אחרות.
הוא יכול לספק שכבת הגנה על הנתונים כיוון שהמתכנת יגדיר איזה נתונים להחזיר למשתמש בעת הצורך.

דוגמה לשימוש ב-Prepare Statement — **JAVA**

```
String query = "DELETE FROM students WHERE studentId=?"
try (PreparedStatement pstmt = con.prepareStatement(query)){
      pstmt.setString(1, userId);
      pstmt.executeUpdate();
}
```

משמעויות המימוש

- הסימן שאלה במחרוזת (השאילתה) מגדיר שאליו אמור להיכנס נתון מסוים.
- לאחר מכן נכניס את השאילתה ל-prepareStatement והוא יחזיר לנו `PreparedStatement` במידה והכל התנהל באופן תקין (בגלל זה ה-try).
- השיטה setString בעצם מקבלת את מספר הארגומנט (מספר הסימן שאלה) שאליו יכנס הערך userID. כלומר ניתן להכניס אפילו כמה סימני שאלה וכל אחד מהם יש לו מספר סידורי בהתאם ובסדר עולה מ-1.
- ה-executeUpdate נועד לעדכן את הטבלה במסד הנתונים בעת הצורך - כאן לדוגמה היינו צריכים אותו כי השתמשנו בפקודה DELETE.

executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex: SELECT	Ex: DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

שימוש ב-Stored Procedure בשפת Java

דוגמה לשימוש ב-Stored Procedure — JAVA

```
String query = "{CALL student_avg(?)}";
CallableStatement stmt = con.prepareCall(query);
int studentId = 222;
stmt.setInt(1, studentId);
ResultSet rs = stmt.executeQuery();
```

היעילות בשימוש ב-Stored Procedure היא שהוא סגור בלוגיקה הממומשת בתוך השרת כך שגם אם ניגשים למסד הנתונים דרך אפליקציה וגם דרך אתר אינטרנט במקביל לדוגמה, אנחנו לא צריכים לחשוש מלשנות את המימוש בעת שימוש בקוד ה-JAVA או ה-HTML וכו'.. כי השינוי מתבצע בשרת בלבד וזה יכול למנוע שגיאות רבות.

דוגמה לשימוש רב פרמטרי ב-Stored Procedure — JAVA

```
CallableStatement stmt = con.prepareCall("CALL student_avg_2(?,?,?)");
stmt.setInt(1, n);
stmt.registerOutParameter (2, Types.DOUBLE);
stmt.registerOutParameter(3, Types.INTEGER);
stmt.execute();

Double avgGrade = stmt.getDouble(2);
Integer maxGrade = stmt.getInt(3);

System.out.println("the average grade is: "+avgGrade+".");
System.out.println("the maximum grade is: "+maxGrade+".");
```

- במימוש כאן אנחנו צריכים קודם להגדיר איזה טיפוסים כל שדה מקבל מהפרמטר ולאחר ה-exec יהיה אפשר להשתמש בנתונים כרצוננו בשפת ה-JAVA.

שימוש ב-XML עבור מסדי נתונים

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.

- ייצוג המידע באופן טקסטואלי.
- נתונים בצורה היררכית.
- נועד לעטוף מידע ולשמור עליו.
- נועד גם להציג מידע (אפליקציות לאנדרואיד ו-web).
- שמירת המטא-מידע ביחד עם המידע עצמו. כלומר, שמירת תיאור הנתונים עם הנתונים עצמם.
- סידור המידע במגוון צורות ולא רק בטבלה כמו ברוב הפורמטים האחרים.
- ב-XML טבעי מאוד לסדר מידע באופן היררכי (עץ).

דוגמה לקובץ XML (נקרא לקובץ זה, קובץ הדוגמה) | **XML**

```
<?xml version="1.0"?> ← כותרת חובה ולא בעלת משמעות כלשהי מלבד הגרסה הקבועה
<University> ← חייב להכיל את השורש של כל הקובץ, שורש אחד ויחיד
  <Student degree="PhD"> ← כאן יש תכונה שנשמרת כ-מטא-דאטא
      <FirstName>Chaya</FirstName>
      <LastName>Glass</LastName>
      <id>111</id>
      <age>21</age>
      <Address> ← אפשר לראות כי ניתן לפתוח באופן היררכי עוד ענף נתונים בתוך הסטודנט
          <Street>Hatamr 5</Street>
          <City>Ariel</City>
          <Zip>40792</Zip>
      </Address>
  </Student> ← סגירת הטאג עבור הסטודנט הספציפי הזה
  <Student> ← פתיחת טאג לסטודנט חדש
      <FirstName>Tal</FirstName>
      <LastName>Negev</LastName>
      <id>222</id>
      <Address>
          <Street>Rotem 53</Street>
          <City>Jerusalem</City>
          <Zip>54287</Zip>
      </Address>
  </Student>
  <Student/> ← אפשר גם לפתוח טאג ריק של סטודנט
<!-- A comment about the file... --> ← ככה כותבים הערות בקובץ
</University> ← סגירת השורש
```

אופרטורים וסימני כתיבה

מה יקרה אם נרצה להשתמש בסימן '<' ב- XML לא כחלק מתג?
נניח שיש לנו תג של "failures" ותלמיד נכשל פחות משלוש פעמים:

```
<failures> < 3 </failures>
```

אנחנו נקבל שגיאה מכיוון שב-XML מבצעים אופרטורים בצורת כתיבה שונה מהרגיל:

```
<failures> &lt; 3 </failures>
```

ועבור הבאים:

<	<
>	>
&	&
'	'
"	"

דוגמה לשימוש ב-XML עם שפת JAVA בשילוב "קובץ הדוגמה" — JAVA

```
File inputFile = new File("student.xml");
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(inputFile);
System.out.println("Root element :" + doc.getDocumentElement().getNodeName());
//Just print root (university)
NodeList nodeList = doc.getDocumentElement().getElementsByTagName("Student");
for (int studentIdx = 0; studentIdx < nodeList.getLength(); studentIdx++){
       Node studentNode = nodeList.item(studentIdx);
       if (studentNode.getNodeType() == Node.ELEMENT_NODE){
       Element element = (Element) studentNode;
         Student student = new Student();
       studentList.add(student);
       System.out.println("Degree : " + element.getAttribute("degree")); //just
print degree ("PhD" when studentIdx=0)
       NodeList studentAllNodes = studentNode.getChildNodes();
       for (int stIdx = 0; stIdx < studentAllNodes.getLength(); stIdx++){
              Node stInnerNode = studentAllNodes.item(stIdx);
              switch (stInnerNode.getNodeName()){
              case "FirstName": student.firstName = stInnerNode.getTextContent();
              break;
              case "LastName": student.lastName = stInnerNode.getTextContent();
```

```
            break;
            case "id": student.id =
Integer.parseInt(stInnerNode.getTextContent()); break;
            case "age": student.age =
Integer.parseInt(stInnerNode.getTextContent()); break;
            case "Address":
                  Address address = new Address();
                  student.address = address;
                  NodeList addressAllNodes = stInnerNode.getChildNodes();
                  for (int adIdx = 0; adIdx < addressAllNodes.getLength();
adIdx++) {
                        Node adInnerNode = addressAllNodes.item(adIdx);
                        switch (adInnerNode.getNodeName()) {
                        case "Street": address.street =
adInnerNode.getTextContent(); break;
                        case "City": address.city = adInnerNode.getTextContent();
                        break;
                        case "Zip": address.zip = adInnerNode.getTextContent();
                        break;
                        }
                  }
                  break;
            }
      }
      }
}
```

שימוש ב-XPATH

- יותר מזכיר את SQL
- נועד לחיפוש ערכים
- ניתן להצביע/לשלוף ערך ספציפי מבלי לעבור על כל העץ
- מתחיל ספירה סדרתית מ-1 ולא מ-0
- אפשר להציב תנאים לשליפת הנתונים
- ה-XPATH מחזיר XML!
- [אתר לבדיקת פקודות XPATH]

דוגמה

נניח שנרצה להשיג את העיר של הסטודנט השני:

`University/Student[2]/Address/City`

JAVA	פתרון לדוגמה בשפת ג'אווה עם שילוב "קובץ הדוגמה"

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

שימוש ב-attributes

נתבונן בקטע מקובץ XML

```
<University> ← חייב להכיל את השורש של כל הקובץ, שורש אחד ויחיד
  <Student degree="PhD"> ← כאן יש תכונה שנשמרת כ-מטא-דאטא
      <FirstName>Chaya</FirstName>
```

עבור ה- "PhD" נקרא attribute וכדי למשוך אותו באמצעות XPATH נשתמש ב-@.

```
University/Student[1]/@degree
```

זה יחזיר לנו **מחרוזת** ולא XML!

שימוש בתנאים

```
doc("students.xml")/University/Student[age<25]
```

אופרטורים נוספים:

OR	|	University/Student/FirstName | /University/Student/Address
All descendants	//	University//Street
All elements	*	University/Student/*
Parent path	/..	/University/Student/Address[Zip=40792]/../FirstName

שימוש ב-XQUERY

- שימוש ב-XPATH כלול בתוכו

ראשי תיבות FLWOR

From

Let - זה הגדרה של משתנה כמו ב-SQL לדוגמה: $SET\ @variable$

Where - זה אותו where של SQL

Order By - אותו הדבר כמו ב-SQL

Return - זה כמו SELECT ב-SQL.

דוגמה

עבור "קובץ הדוגמה" נשאל ב-XQUERY:

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

ונקבל את התשובה הבאה:

הקובץ שקיבלנו: XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

ועבור השאילתה הבאה:

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

נקבל: XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<id>111</id>
```

שימוש ב-XSD

- ה-XML מאוד גמיש, אין לו חוקיות ברורה, הוא ללא מגבלות.
- ה-XSD הוא תקן המשמש להגדרת סכמה (מבנה) של מסמכים הכתובים בשפת XML.
- בעזרת XSD ניתן להגדיר באופן בלתי תלוי בפלטפורמה ובמימוש מבנה לייצוג מידע ב-XML ולבדוק את תקינותו (התאמתו לתקן) של XML .
- הוא מאפשר לנו לקחת XML מסוים ולהגדיר ולחייב עליו חוקים מסוימים.

דוגמה ל-XSD

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> ← מכאן אנחנו מקבלים את כל הפונקציות
   <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" /> ← תומך במלא טיפוסים
                    <xs:element name="Address"> ← סוג-מורכב הנמצא בתוך הסטודנט
                           <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>
```

מערכת טיפוסים ב-XSD	
sequence	אלמנט המאפשר יצרת מופעים נוספים של תת-אלמנטים בסדרת מופעים כאשר יש לפחות מופע אחד יכול להיות מוכל ב group, choice, sequence, complexType, או בהורשת טיפוסי נתונים. תג squence מאפשר יצירת מערכי פתרון לרשימות מקושרות על ידי יצרת שימוש בתגיות: • **maxOccurs**= מספר שלם חיובי או **unbounded** (כמות הופעות לפי מספר שלם או כל מספר) • **minOccurs**=מספר שלם חיובי הערך הדיפולטיבי שלהם אם לא נציין במפורש יהיה 1. מערך סטטי (גודל ידוע בזמן יצירת ה XML) ייצוג על ידי הגדרת macOccurs למספר שלם חיובי. הרשימה תיוצג על ידי שימוש בהגדרה: unbound עבור כמות מרבית של הופעות.

עבור attributes

```
<xs:attribute name="xxx" type="yyy"/>
```

כאשר type יכול להיות כל טיפוס מוכר כמו int או string

ה attribute לא חייב להופיע ב XML מתוך ברירת המחדל שלו.

אם רוצים לחייב שימוש של attribute, מוסיפים את המאפיין use.

```
<xs:attribute name="ID" type="xs:string" use="required"/>
```

אינדיקטורים

אינדיקטורים מסמנים כיצד אנו רוצים שהאלמנטים יסודרו בתוך מסמך ה XML.

<u>מסמני סדר (תגים העוטפים את האלמנטים):</u>

1. **all** - כל הילדים חייבים להופיע פעם אחת בלבד, לא משנה הסדר. (*)
2. **choice** - רק ילד אחד מתוך הרשימה יכול להופיע.
3. **sequence** - כל הילדים חייבים להופיע, לפי הסדר בו הם נכתבו.

<u>מסמני כמות (מאפיין לאלמנט):</u>

4. **minOccurs** - כמות הפעמים המינימלית לאלמנט, ברירת המחדל היא 1, מקבל מספרים חיוביים בלבד. (*)
5. **maxOccurs** - כמות הפעמים המקסימלית לאלמנט, ברירת המחדל היא 1, מקבל מספרים חיוביים בלבד, מלבד לאינסוף שבו נכתוב "**unbounded**". (*)

הגבלת מחרוזות

XSD

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

בדוגמה השתמשנו ב-**pattern** שמגדיר את הצירוף המדויק של תווים שמאופשר לערך.
במקרה זה, אפשרנו כל מחרוזת שאין בה רווחים.

שימוש ב-JSON עבור מסדי נתונים

Java**S**cript **O**bject **N**otation - **JSON** הוא פורמט החלפת נתונים, מבוסס טקסט, שאינו תלוי בשפה, שקל לקריאה ולכתיבה לבני אדם ולמכונות.
JSON יכול לייצג שני סוגים:

אובייקט הוא אוסף לא מסודר של אפס או יותר זוגות שם / ערך.

JSON

```
{
   "Class":{ "name":"Peter", "age":20, "score": 70.05}
}
```

מערך הוא רצף מסודר של אפס או יותר ערכים. הערכים יכולים להיות מחרוזות, מספרים, בוליאנים, null או מערכים נוספים ואובייקטים.

JSON

```
{
   "arr" : ["black", "red", " Green"]
}

{
   "collection" : [
            {"one" : 100},
            {"two" : 200},
            {"three" : 300}
    ]
 }
```

מחלקות עיקריות ב-JSON	
JSONParser	נועד לניתוח מחרוזת JSON, הבנאי מקבל String או java.io.Reader
JSONObject	יורשת מ-HashMap ומאחסנת (ערך,מפתח).
JSONArray	יורשת מ-ArrayList ומייצגת אוסף
JSONValue	נועד להמרת מחרוזות JSON באובייקטים של Java.

JAVA | **כתיבה (write) לקובץ JSON**

```java
public class JSONWriteExample {
    public static void main(String[] args) throws FileNotFoundException {
        // creating JSONObject
        JSONObject jo = new JSONObject();
        // putting data to JSONObject
        jo.put("firstName", "John");
        jo.put("lastName", "Smith");
        jo.put("age", 25);

        // for address data, first create LinkedHashMap
        Map m = new LinkedHashMap(4);
        m.put("streetAddress", "21 2nd Street");
        m.put("city", "New York");
        m.put("state", "NY");
        m.put("postalCode", 10021);

        // putting address to JSONObject
        jo.put("address", m);

        // for phone numbers, first create JSONArray
        JSONArray ja = new JSONArray();
        m = new LinkedHashMap(2);
        m.put("type", "home");
        m.put("number", "212 555-1234");

        // adding map to list
        ja.add(m);
        m = new LinkedHashMap(2);
        m.put("type", "fax");
        m.put("number", "212 555-1234");

        // adding map to list
        ja.add(m);
        // putting phoneNumbers to JSONObject
        jo.put("phoneNumbers", ja);

        // writing JSON to file:"JSONExample.json" in cwd
        PrintWriter pw = new PrintWriter("JSONExample.json");
        pw.write(jo.toJSONString());

        pw.flush();
        pw.close();
    }
}
```

JAVA | **קריאה (read) מקובץ JSON**

```java
public class JSONReadExample {
    public static void main(String[] args) throws Exception {
        // parsing file "JSONExample.json"
        Object obj = new JSONParser().parse(new FileReader("JSONExample.json"));

        // typecasting obj to JSONObject
        JSONObject jo = (JSONObject) obj;

        // getting firstName and lastName
        String firstName = (String) jo.get("firstName");
        String lastName = (String) jo.get("lastName");

        System.out.println(firstName);
        System.out.println(lastName);

        // getting age
        long age = (long) jo.get("age");
        System.out.println(age);

        // getting address
        Map address = ((Map)jo.get("address"));

        // iterating address Map
        Iterator<Map.Entry> itr1 = address.entrySet().iterator();
        while (itr1.hasNext()) {
            Map.Entry pair = itr1.next();
            System.out.println(pair.getKey() + " : " + pair.getValue());
        }

        // getting phoneNumbers
        JSONArray ja = (JSONArray) jo.get("phoneNumbers");
        // iterating phoneNumbers
        Iterator itr2 = ja.iterator();

        while (itr2.hasNext()) {
            itr1 = ((Map) itr2.next()).entrySet().iterator();
            while (itr1.hasNext()) {
                Map.Entry pair = itr1.next();
                System.out.println(pair.getKey() + " : " + pair.getValue());
            }
        }
    }
}
```

ניהול מסדי נתונים במערכות NoSQL

NoSQL הוא קטגוריה חדשה יחסית של בסיסי נתונים, אשר נותנים פתרון אחסון וגישה למידע שאינו ממודל במבנה טבלאי יחסי אשר נפוץ בבסיסי נתונים יחסיים (כלומר הם לא רציונלים).
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 "**N**ot **O**nly **SQL**", כדי להדגיש שלחלקן תמיכה בשפת השאילתות SQL.
הוא גם לא מבטיח את כל ה-ACID!

קטגוריות של NoSQL	
Document Databases	בסיס נתונים המצמד מפתח עם מבנה מורכב של מידע הנקרא "מסמך". מסמך יכול להכיל צמדים מורכבים של מפתח-ערך, מפתח-מערך, או אפילו מפתח-מסמך. דוגמה: **MongoDB**
Graph Stores	משמש לשמירת מידע הקשור לרשתות וקשרים חברתיים. דוגמה: **Neo4J**, HyperGraphDB
Key-Value Stores	משמש לבסיס נתונים פשוט ומהיר, התומך בצמדי מפתח-ערך בלבד. חלק מבסיסי Key-Value Stores מאפשרים להגדיר סוג (TYPE) לערכים (כגון Integer, String). דוגמה: **Redis**, Riak, Voldemort
Wide-Column Stores	בסיס נתונים מבוססי עמודות (במקום שורות), בו משמשים בעיקר לשאילתות על מערכי מידע גדולים במיוחד. דוגמה: **Cassandra**, HBase

במקום ACID יש את BASE

Basically **A**vailable - הנתונים זמינים רוב הזמן (יש להם מנגנוני שכפול נתונים כגיבוי במקרה של איבוד נתונים).
soft **S**tate - המצב עשוי להשנות גם ללא עדכונים (מכיוון שנתונים ישנים עדיין פועלים).
Eventual consistency - עקביות על הנתונים (השינוי יעודכן בכל השכפולים/שרתים שלנו כדי להבטיח עקביות).

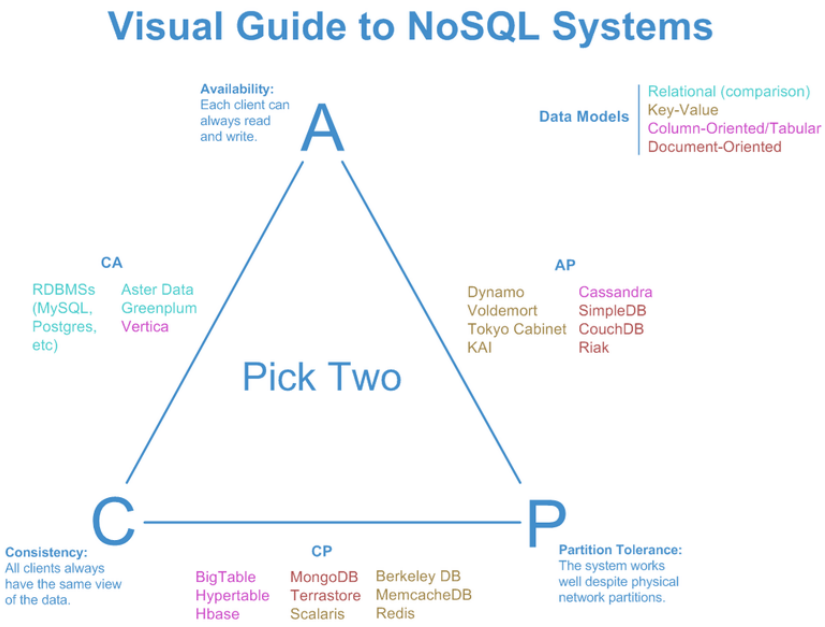
תיאוריית ה-CAP

לא ייתכן שמערכת מחשוב מבוזרת תספק בו זמנית יותר משתיים **מתוך** שלוש הערבויות הבאות:

Consistency - עקביות: בדומה ל- C ב- ACID, לכל הצמתים במערכת תהיה תצוגה זהה של הנתונים בכל עת.

Availability - זמינות: כל בקשה זוכה לתגובה (לאו דווקא הערך העדכני ביותר).

Partition tolerance: המערכת נשארת מקוונת אם מתרחשות בעיות רשת בין צמתי המערכת.



בסיס הנתונים Redis

Redis הוא בסיס נתונים מסוג **NoSQL** הפועל בזיכרון (In-Memory), ומבוסס קוד פתוח.
Redis מאפשר פתרון בעיות מורכבות באמצעות אופטימיזציה של מבנה הנתונים והפקודות המבוצעות באופן מהיר ובפשטות.
Redis מבוסס על שליפת הנתונים על בסיס "מפתחות" כלומר **Key-Value Store** המאפשר שליפת נתונים באופן מהיר ביותר מתוך מאגרי מידע ענקיים (Big data).
טרמינל לתרגול של רדיס: https://try.redis.io/

פקודות בסיסיות

Redis

```
SET hello "world"
```

Redis — **תחזיר לנו "world"**

```
GET hello
```

Redis — **תחזיר לנו nil (כלומר null)**

```
GET "world"
```

Redis — **תחזיר לנו "Yossi"**

```
SET name1 Yossi
GET name1
```

Redis — **תחזיר לנו את כל הסקריפט (אין משמעות לסימנים במפתח, אפשר לכתוב הכל כמו : ,^?*$@)**

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

Redis — **תמחק לנו את הערך במפתח שצוין**

```
DEL user:1001
```

פקודות עבור מספרים

Redis	ה-INCR יגדיל את bottles+1 כלומר הוא יהיה 6 בערך int

```
SET bottles 5
INCR bottles
```

Redis	ה-INCRBY יפחית את bottles עם 3- כלומר הוא יהיה bottles = 3

```
INCRBY bottles -3
```

אפשר גם להוסיף ככה יותר מ-1 למשל (INCRBY bottles 3) מוסיף 3 לערך bottles .

פקודות עבור רשימות

Redis	מכניס לנו מימין לרשימה את הערכים שכתובים במרכאות (אין חשיבות למספר 571 כמיקום כלשהו)

```
RPUSH user:571:items "chair"
RPUSH user:571:items "table"
RPUSH user:571:items "water"
```

Redis	מכניס משמאל לרשימה

```
LPUSH user:571:items "cup"
```

Redis	תשלוף את כל הערכים החל מ-1 עד 2 (אנחנו סופרים ב-Redis החל מ-0)

```
LRANGE user:571:items 1 2
```

נקבל:

```
1) "chair"
2) "table"
```

Redis	תשלוף את כל הערכים החל מ-0 עד 1- (כלומר עד הערך האחרון)

```
LRANGE user:571:items 0 -1
```

נקבל:

```
1) "cup"
2) "chair"
3) "table"
4) "water"
```

Redis	תכניס כמה ערכים לרשימה מימין במכה אחת

```
RPUSH user:573:items "bed" "chair" "table" "can"
```

פקודות עבור Hashes

Redis	מכניס 2 צמדים למפה לפי הסדר של (מפתח וערך)

```
HSET user:302 first_name "Tamar"
HSET user:302 last_name "Cohen"
```

Redis	שולף את "Tamar"

```
HGET user:302 first_name
```

Redis	מכניס כמה צמדים במכה ואז שולף את 3 מתוך degree

```
HMSET user:302 degree 3 gender "female"
HGET user:302 degree
```

Redis	מציג את כל הנתונים במפה (בלי קשר למפתח או ערך)

```
HGETALL user:302
```

נקבל:

```
1) "first_name"
2) "Tamar"
3) "last_name"
4) "Cohen"
5) "degree"
6) "3"
7) "gender"
8) "female"
```

קבלת מפתחות לפי תבנית מסוימת

Redis	עבור הנתונים הבאים:

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

כדי לעבור על המפתחות שלהם כמו לולאה נבצע בצורה הבאה:

Redis	

```
KEYS user:100?
```

ונקבל:

```
1) "user:1000"
2) "user:1001"
3) "user:1002"
```

אפשר גם לקבל את כל המפתחות כך:

Redis	

```
KEYS user:*
```

ונקבל:

```
1) "user:1000"
2) "user:1010"
3) "user:1003:items"
4) "user:1001"
5) "user:1002"
```

פקודות על Set

- **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)

פקודות על Set מסודר

- **ZADD** x val y (add item y with score value to sorted set x)
- **ZRANGE** x y z (return z items from x, starting at y)

שימוש ב-Expire

הפקודות EXPIRE מגדירות זמן בשניות שבסופן יפוג הערך.

Redis	נגדיר 100 שניות תקפות עבור הערך במפתח user:302
`EXPIRE user:302 100`	

Redis	יחזיר לנו ב-int את מספר השניות שנותרו עבור תוקף הערך במפתח הזה (TTL = Time To Live)
`TTL user:302`	

נקודה חשובה - ב-Redis לא ניתן לתשאל על הנתונים עצמם.

בסיס הנתונים Cassandra

נרצה להשתמש ב-Cassandra כאשר יותר חשוב לנו לקרוא מידע מהר, מאשר לכתוב אותו מהר.
קסנדרה (באנגלית: **Cassandra**) הוא בסיס נתונים מבוזר מסוג NoSQL בקוד פתוח שנכתב במקור בחברת פייסבוק, יצא לאור כתוכנה חופשית בשנת 2010 ומפותח מאז במסגרת מוסד התוכנה אפאצ'י.
השימוש העיקרי של מסד הנתונים הוא בעולם ה-Big Data, המיועד לטיפול בכמות גדולה מאוד של נתונים בנפח שמעל מאות טרה-בית, בקצבי הגעה מהירים מאוד וממקורות רבים ושונים.
היא מספקת ממשק פשוט יותר של חיפוש ערך לפי מפתח כלומר Wide-Column-Store.
טרמינל אונליין של קסנדרה לתרגול: https://www.datastax.com/try-it-out
ה-CQL - Cassandra Query Language בשונה מ-SQL היא שפת שאילתה:

- ללא שימוש ב-JOIN.
- ללא שימוש בתתי-שאילתות.

תכונות של קסנדרה

- לקסנדרה יש מנגנון replication (שכפול) - שמגבה לנו את המידע בכמה מקומות באופן מבוזר.
- חיפוש שלא לפי מפתח מצריך בקשת allow ומאוד לא מומלץ להשתמש בזה במערכת Big Data שכזאת.
- לקסנדרה אין Schema ולכן אורך השורות בטבלה לא חייב להיות זהה.
- בזמן נתון יכול להיות שהמידע לא יהיה זהה בכל שכפול של הטבלאות.
- ה-partition key מגדיר את ה-node ב-cluster כאשר ניגשים ל-"שורה".
- השרת ב-cluster נקרא node.

קסנדרה מול מסד נתונים רלציוני

- **שימוש בזיכרון**:
  ברלציוני נרצה שבמסד שלו נכתוב כמה שפחות (ונחסוך בזיכרון).
  בקסנדרה נעדיף לכתוב יותר על מנת לשלוף את הנתונים מהר יותר.
- **כפילויות**:
  ברלציוני לא נרצה כפילות של נתונים בכלל.
  בקסנדרה אנחנו בכוונה משכפלים את הנתונים כדי לשלוף אותם מהר יותר.

Partition key

- יכול לכלול יותר מעמודה אחת
- בחיפוש חובה לציין את כל ה-partition key בשביל שקסנדרה תדע היכן לחפש.

Clustering key

- כל מי שמגיע אחרי ה-partition ב-primary key
- אחראים לסידור המידע "בשורה"
- ברירת המחדל של הסידור הוא בסדר עולה.

KeySpace

הוא בעצם ה-top level space.

- כאשר נגדיר את ה-KeySpace נצטרך להגדיר גם את אסטרטגיית השכפול (replication) ומספר ההעתקים.
- ה-SimpleStrategy יוצר עותקים כמספר שצוין, עם מרכז אחד.
- ה-NetworkTopologyStrategy יוצר העתקים כמספר שצוין ואחר כך מחלק אותם במספר מרכזים.
- ה- replication_factor הוא בעצם מספר השכפולים של המסד.

Cassandra — **יצירת מסד נתונים**

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

- לא ניתן למיין את המידע בשאילתה כלשהי, לכן יש להגדיר את הסדר של המידע ביצירת הטבלה ע"י:
  `WITH CLUSTERING ORDER BY (col ASC|DESC)`

Cassandra — **יצירת טבלת students**

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

- חשוב להגדיר את ה-key!
- בשונה מ-SQL עבור ה-VARCHAR אנחנו לא צריכים להגדיר את האורך שלו.

Cassandra — **הכנסת נתונים לטבלת students**

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

INSERT INTO students (id, firstName)
VALUES (222, 'Only First');
```

- חובה להשתמש ב- גרש אחד כלומר ' '.

Cassandra — **שליפת מידע**

```
SELECT * FROM students;
```

פלט:

```
 id  | age  | firstname  | lastname
-----+------+------------+----------
 111 |   21 |      Chaya |    Glass
 222 | null | Only First |     null
```

Cassandra	שימוש ב-WHERE

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

נקבל:

```
 id  | age | firstname | lastname
-----+-----+-----------+----------
 111 |  21 |     Chaya |    Glass
```

Cassandra	אבל אם נבצע שימוש ב-WHERE בצורה הבאה:

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

- נקבל שגיאה כיוון שקסנדרה לא מסננת נתונים כמו SQL.
- אז למה זה עבד לנו עם ה-id? - נשים לב שבהגדרת הטבלה הגדרנו את ה-id להיות primary key!
- אפשר לעקוף את זה בשימוש Allow Filtering בסוף השאילתה (אבל לא מומלץ כי הפעולה מאוד איטית).

Cassandra	שגיאה נוספת

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

- אי אפשר לבצע פעולות השוואה על ה-partition key.

שיטת החיפוש הנכון בקסנדרה

- ניתן לחשוב על חיפוש כחיפוש path של קובץ.
- עבור ההגדרה הבאה נראה תקינות של חיפוש:

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

- **שאילתות תקינות**:
  - הרעיון הוא שפעולות השוואה וטווח הם 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;
```

- **שאילתות לא תקינות**:
  - לא שומרות של סדר חיפוש תקין.

```
SELECT * FROM T WHERE y = 5;
SELECT * FROM T WHERE x = 5 AND z = 7;
SELECT * FROM T WHERE x = 5 AND y < 3  AND z > 0;
SELECT * FROM T WHERE x = 5 AND z = 4  AND y > 0;
SELECT * FROM T WHERE x < 10;
SELECT * FROM T WHERE y = 2 AND z < 8;
```

בסיס הנתונים MongoDB

- בסיס הנתונים נשען על מבנה של מסמך (Document-Oriented Database) בניגוד לבסיסי נתונים טבלאיים כמו mySQL העובד מעל טבלאות מקושרות.
- הוא בסיס הנתונים הכי פופולארי מעולם ה-NoSQL.
- מבנה המסמכים עובד מעל מימוש של JSON.
- MongoDB תומך בחיפושי שדה, שאילתת טווח וביטוי רגולרי.
- שאילתות יכולות להחזיר שדות ספציפיים של מסמכים ולכלול גם פונקציות JavaScript המוגדרות על ידי המשתמש.
- ניתן גם להגדיר שאילתות להחזיר מדגם אקראי של תוצאות בגודל נתון.
- טרמינל אונליין של מונגו: https://www.mplay.run/mongodb-online-terminal

MongoDB	מסד נתונים רלציוני
Database	Database
Collection	Table
Document	Row
Field	Column
Embedded Documents	Table Join
Primary Key - _id מקבלים באופן אוטומטי מפתח Primary Key	Primary Key

השימוש ב-use נועד להצביע על ה-mydb שנרצה לגשת אליו (אם לא קיים יוצר חדש כזה)

MongoDB

```
USE University
```

- המילה db היא מילה שמורה הנועדה להצביע על המסד נתונים שבו אנחנו עוסקים באותו הרגע.

מחיקה

MongoDB

```
db.dropDatabase()
```

יצירת טבלה חדשה בשם students

MongoDB

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

- ב-MongoDB המילה collection היא בעצם כמו טבלה.
- ה-capped מגדיר שאם נכנסים נתונים חדשים לטבלה שכבר אין בה מקום, אז באופן אוטומטי, הנתון הכי ישן בטבלה ימחק ויפנה מקום לחדש.
- ה-size מוגדר בבתים.
- ה-max הוא מספר הרשומות שאפשר להכניס לטבלה (מספר הסטודנטים).
- ה-autoIndexID יוצר _id לכל רשומה באופן אוטומטי.

MongoDB	הכנסת רשומה לטבלה בפורמט של JSON

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

MongoDB	ניתן להכניס גם כמה רשומות במכה אחת כשהם נכנסות בתור רשימה של רשומות

```
db.students.insert([{"FirstName": "Tom", "LastName": "Glow", "Address": {"Street":
"Mishmar 5","City": "Ariel"}}, {"FirstName": "Tal", "LastName": "Negev", "Address":
{"Street": "Yarkon 26","City": "Jerusalem"}}])
```

MongoDB	קבלת כל הערכים

```
db.students.find()
```

נקבל:

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

MongoDB	קבלת כל הערכים באופן יפה וקריא יותר

```
db.students.find().pretty()
```

MongoDB	מציאת ערך מסוים
db.students.find({“FirstName”: “Tal”})	

נקבל:

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

MongoDB	מציאת רכיב בתוך רכיב
db.students.find({“Address.City”: “Ariel”})	

נקבל:

```
{ "_id" : ObjectId("589afa8c44a5653a862dd692"), "FirstName" : "Chaya", "LastName" :
"Glass", "id" : "111", "age" : "21", "Address" : { "Street" : "Hatamr 5", "City" :
"Ariel", "Zip" : "40792" } }
{ "_id" : ObjectId("589afa9244a5653a862dd693"), "FirstName" : "Tom", "LastName":
"Glow", "Address" : { "Street" : "Mishmar 5", "City" : "Ariel" } }
```

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

MongoDB	שימוש ב-and

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

- כלומר, ה-and מגיע לפני הכל, וכל פסיק בתוך הרשימה מפריד כ-"וגם".

נקבל:

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

MongoDB	שימוש ב-or

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

- באותו השיטה כמו במקודם, פסיק בתוך הרשימה מוגדר כ-"או".

MongoDB	בחירת שדות ספציפיים

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

נקבל:

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

MongoDB	בחירת שדות ספציפיים, בלי לקבל את ה-id כפלט:

```
db.students.find({"FirstName":"Tim"},{"FirstName":true, _id:false})
```

נקבל:

```
{"FirstName" : "Tim" }
```

MongoDB	שאילתה שמקבילה ברלציוני: SELECT FirstName FROM students

```
db.students.find({} , {"FirstName":true, _id:false} )
```

נקבל:

```
{ "FirstName" : "Chaya" }
{ "FirstName" : "Tim" }
{ "FirstName" : "Tal" }
```

MongoDB	עדכון רגיל מתבצע כך:

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

- מונגו יחפש עבור השם Tom ויחליף את כל הרשומה בגלל השינוי שם הזה.

MongoDB	עדכון קצר יותר

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

- השימוש ב-set לא ישנה את שאר השדות שאין בהן צורך לשינוי.
- אם לא נשתמש ב-multi המונגו יעדכן רק את הרשומה הראשונה שהוא יפגוש.

MongoDB	מחיקה

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

- ה-1 הוא משתנה בוליאני, האם למחוק רק את הראשון שמוצאים או את כולם.
- ה-DELETION_CRITERIA הוא מה בעצם אנחנו רוצים למחוק?

MongoDB	הפונקציה limit מגבילה את מספר המסמכים לשליפה

```
db.COLLECTION_NAME.find().limit(NUMBER)
```

MongoDB	הפונקציה skip מדלגת על מסמך במספר כלשהו

```
db.COLLECTION_NAME.find().skip(NUMBER)
```

MongoDB	הפונקציה sort שולפת לפי סדר מסוים

```
db.COLLECTION_NAME.find().sort({KEY_NAME:1})
```

- כאשר 1 זה ASC ו -1 זה DESC.

פונקציות Aggregation

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"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$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"}}}])

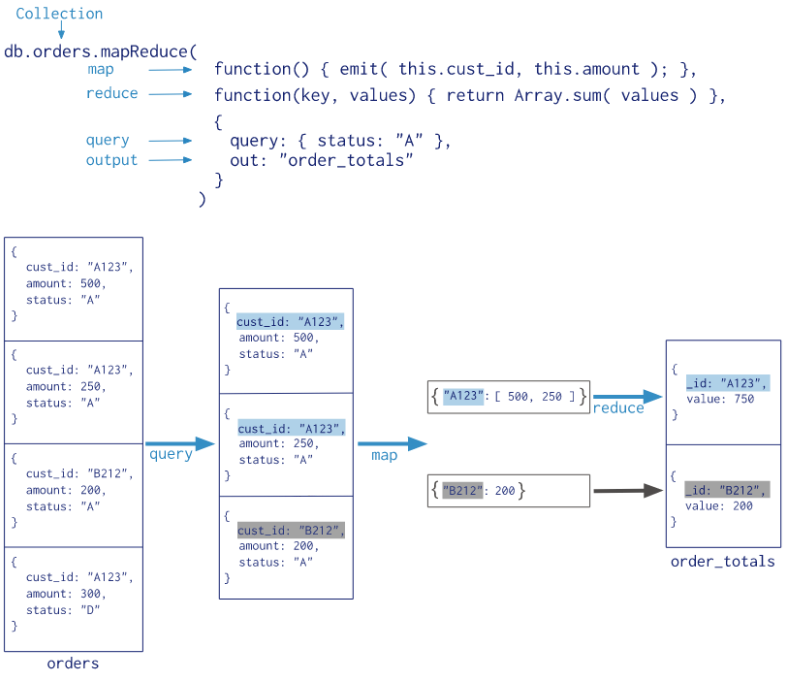
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Map-Reduce

- Mapper - מחלק את המידע, מסנן ומריץ את התהליך.
- Shuffle - מוודא שכל ה-workers קיבלו נתונים לעבוד עליהם.
- Reduce - עיבוד המידע שברשות ה-worker לקבלת התשובה הסופית.

MongoDB	דוגמה לתבנית map reduce

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



דוגמה

עבור המבנה הבא:

- Students:
  - gender
  - age
  - name
  - grades
    - subject
    - grade

נראה את השאילתה הבאה בשיטת map reduce:

```
db.students.aggregate([{$group:{_id:"$gender", count:{$sum:1 }}}])
```

MongoDB	מימוש הדוגמה בשיטת map reduce

```
mapper = function () {
    emit(this.gender, 1);
};

reducer = function(gender, count) {
    return Array.sum(count);
};

db.sourceData.mapReduce (
    mapper,
    reducer,
    {
        out : "example1_results"
    }
);

db.example1_results.find()
```

בסיס הנתונים Neo4j

- מערכת ניהול נתונים המבוססת על מבנה של גרף - **Graph Store**
- מורכב משני אלמנטים, קודקודים וקשתות
- משמש לשמירת מידע הקשור לרשתות וקשרים חברתיים.
- מאפשר למצוא חברים של קודקוד וחברים של חברים עד דור 10.
- סרטון הסבר בקישור וקונסולה להרצת שאילתות בדפדפן בקישור

Neo4j	רלציוני
Graph	Table
Node	Row
Property and its values	Column and Data
Relationship	Constraint
Traversal	Join

שפת השאילתות CQL - Cypher Query Language

- שפת השאילתות לבסיסי נתונים של Neo4j.
- זוהי שפה דקלרטיבית.
- התחביר פשוט וקריא.
- השפה משמשת כדי ליצור ולאחזר את היחסים בין הנתונים מבלי להשתמש בשאילתות מורכבות.

פקודה	שימוש
CREATE	יוצר צמתים, יחסים ותכונות
MATCH	שולף מידע על הצמתים, יחסים ותכונות
RETURN	מחזיר את תוצאת השאילתה
WHERE	מספק תנאים כדי "לפלטר" את המידע שהתקבל
DELETE	מוחק צמתים ויחסים
REMOVE	מוחק תכונות של צמתים ויחסים
ORDER BY	מסדר את המידע שהתקבל
SET	מוסיף או מעדכן

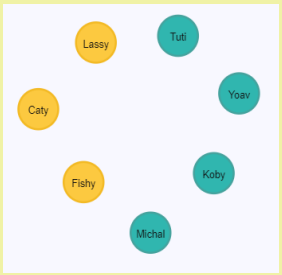
Neo4j	מחיקת כל הקודקודים והקשתות הקיימים

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

- אנו ניצור מבנה גרף חדש המתאר משפחה:
  - הורים: מיכל וקובי
    - ילדים: יואב ותותי
  - חיות מחמד: לאסי, קטי ופישי.

Neo4j	יצירת צמתים (לא מחוברים)

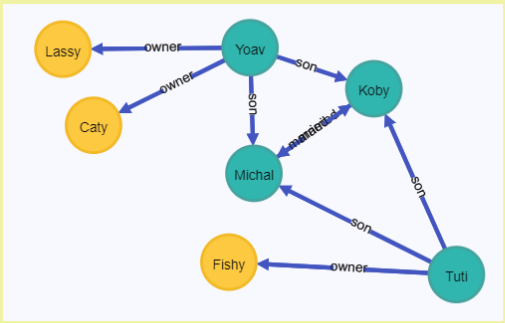
```
Create
(Michal:Person {name:'Michal', age: 36,gender:"female"}),
(Koby:Person {name:'Koby', age: 39,gender:"male"}),
(Yoav:Person {name:'Yoav', age: 10,gender:"male"}),
(Tuti:Person {name:'Tuti', age: 4,gender:"female"}),
(Lassy:Pet {name:'Lassy', age: 2,gender:"female" ,type: "dog"}),
(Caty:Pet {name:'Caty', age: 4,gender:"male", type: "cat"}),
(Fishy:Pet {name:'Fishy', age: 6,gender:"male", type: "fish"})
```



- בפקודת Create אחת ניתן ליצור רשימה של צמתים.

Neo4j	יצירת יחסים בין הצמתים

```
Create
(Koby)-[:married]->(Michal),
(Michal)-[:married]->(Koby),
(Yoav)-[:son]->(Michal),
(Tuti)-[:son]->(Michal),
(Yoav)-[:son]->(Koby),
(Tuti)-[:son]->(Koby),
(Yoav)-[:owner]->(Lassy),
(Yoav)-[:owner]->(Caty),
(Tuti)-[:owner]->(Fishy)
```



- בפקודת Create אחת ניתן ליצור רשימה של יחסים.

Neo4j | **בעזרת הפקודה Match מאחזרים מידע מבסיס הנתונים**

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

- בשאילתא הזאת, החזרנו את כל הצמתים שהם מסוג Person.

Neo4j | **ניתן להחזיר את כל הקודקודים לפי סוג היחס אליהם**

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

נקבל

n.name
Caty
Caty
Lassy
Lassy

Neo4j | **דוגמה נוספת**

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

נקבל

n.name
Michal
Koby
Yoav
Tuti

Neo4j | **דוגמה נוספת**

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

b.name
Lassy
Caty

Neo4j | **שליפת כל הקשרים (ולא רק קשרים ישירים)**

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

שימוש ב-Where

Neo4j	אותו WHERE מוכר, כאן מבצעים לפי תכונות של צמתים

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

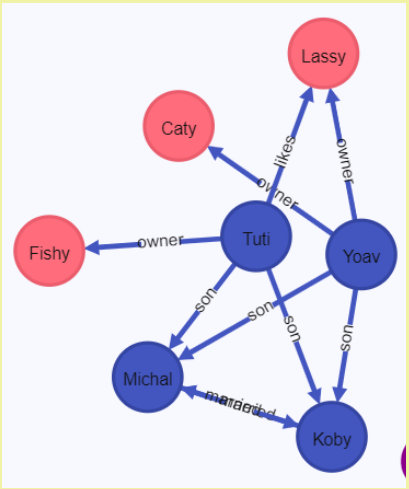
Neo4j	שאילתה בשילוב של Where עבור קשרים בין צמתים

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

יצירת רלציה (קשר) חדשה

Neo4j	יצירת רלציה חדשה בשם Likes

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



Neo4j	דוגמה נוספת

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

שימוש ב-Remove מאפשרת מחיקת תכונות של קודקוד

Neo4j	דוגמה לשימוש ב-Remove

```
MATCH (GoldenFishy:Pet {name: "GoldenFishy", age:0.5,gender:"male", type: "fish"})
REMOVE GoldenFishy.gender
```

שימוש ב-Delete

Neo4j	מחיקת כל הקודקודים

```
MATCH (n) detach
DELETE n
```

Neo4j	מחיקה של קודקוד ספציפי (כולל כל הקשרים שלו)

```
MATCH (GoldenFishy:Pet {name: "GoldenFishy", age:0.5,gender:"male", type: "fish"})
DETACH DELETE GoldenFishy
```

- המשמעות של DETACH זה מחיקת קודקוד ספציפי וכל הקשרים שלו ממנו ואליו.

Generation

המשמעות היא להוסיף עוד יחס, כלומר שאפשר להוסיף כמה שיותר קשרים.

ברגע שיוצרים רלציות אנחנו בעצם יוצרים generations לכן אין לזה משמעות מלבד היחס שהגדרנו לדור הספציפי.

Neo4j	יצירת יחס חדש (דור חדש) בין אבא לבן

```
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)
```

פונקציות אגרגטיביות	
Count	מחזיר את מספר השורות בהם בוצעה התאמה בשאילתה
Max	מחזיר את הערך המקסימלי מבין אוסף של רשומות
Min	מחזיר את הערך המינימלי מבין אוסף של רשומות
Sum	מחזיר את סכום של ערך הנמצא בין כל הרשומות באוסף
Avg	מחזיר את הממוצע של ערך הנמצא בין כל הרשומות באוסף
Collect	מחזיר את כל התוצאות ברשימה אחת במקום בטבלה

שימוש ב-With

תפקידו להגביל את מספר התוצאות אבל באופן כללי הוא שאנחנו מחזירים את התוצאות ואז אפשר לעשות עליהם מניפולציות לפי שמציגים אותם.

Neo4j	רשימת כל השמות בסדר יורד עם הגבלה ל-3 שמות

```
MATCH (n)
WITH n
ORDER BY n.name DESC LIMIT 3
RETURN collect(n.name)
```

collect(n.name)
[Yoav, Tuti, Michal]

מעבר בין יחסים

Neo4j	תחזיר את כל השמות של יחס של מורה עד אורך 2 (כלומר לא רק שכנים אלא שכנים של שכנים)

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

- כנ"ל לגבי [3 ... 1 * :relation]או [.... 5 * :relation]או פשוט [* :relation]
- ב-Neo4j אפשר ליצור גרפים העומדים עד אורך של 10 ולא יותר מזה.

מסלולים

Neo4j	מחזיר את המסלול הקצר ביותר בין 2 סטודנטים

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

p
[(8:student {age:24, degree:"1", id:333, name:"Gadi Golan"}), (6)-[8:in_class_with]->(8), (6:student {age:21, degree:"1", id:111, name:"Chaya Glass"}), (7)-[6:teaches]->(6), (7:student {age:28, degree:"3", id:222, name:"Tal Negev"})]

שימוש ב-ALL

נרצה להשתמש בו כאשר נעבור על אוסף של תוצאות ולוודא כי **כל** התוצאות באוסף עומדות בתנאי כלשהו.

Neo4j	שימוש ב-ALL מחזיר את כל 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
```

- שימוש ב-collect יכול להיות לא רק לתצוגה אלא כשימוש למבנה נתונים זמני לכל דבר ועניין.

שימוש ב-AND ו-OR

Neo4j	דוגמה

```
MATCH (c)
WHERE c.salary > 1000 AND c.address = 'Tel Aviv' OR c.address = 'Los Angeles'
```

תרגיל דוגמה

Neo4j	דוגמה

```
MATCH (a:Person{name: "Bob"})-[:watch]-(b:movie) WITH collect(b) as movies
MATCH (f:Person) WHERE ALL (x IN movies WHERE (f)-[:watch]-(x))
RETURN f.name
```

מחזיר את כל החברים של בוב שראו את כל הסרטים שבוב ראה.

כל פעם שרוצים להשתמש ב-MATCH נוסף, צריך להשתמש ב-WITH.

אם משתמשים ב-WITH, צריך לרשום תמיד אחריו את המשתנים שהיו לפני ה- WITH (למשל b) כדי שנשתמש בהם אחריו.

תרגיל מבחן

כתבו שאילתא שמחזירה את כל החברים של גל גזית, וגם את החברים של החברים שלו וגם החברים של החברים של החברים שלו שצפו בסרטי הארי פוטר.

Neo4j	פתרון

```
MATCH (a:Person)-[:friend*1..3]->(b:Person{name:'Gal'})
MATCH (d:Movie {name:'Harry Potter'}) WHERE (a)-[:watch]->(d) WITH COLLECT(DISTINCT a) as fr
RETURN fr
```

מצאנו את כל החברים של גל עד רמה 3, אחר כך רצינו לשאול על החברים שראו את סרטי הארי פוטר,

לכן, צריך לצרף עוד שאילתא ולכן נרשום WITH. אנחנו משתמשים ב-a ואחר כך לעשות עליו WHERE ALL ולכן חייבים לעשות אוסף collect של a. (כלומר זה אוסף כל החברים של גל עד רמה 3).

בסיס הנתונים Elastic Search

- בסיס נתונים מבוסס חיפוש מסוג Document Stores
- עובד לפי ניקוד ורלוונטיות של תוצאות אפשריות בחיפוש בבסיס הנתונים שלו
- דומה לחיפוש של גוגל או אתרים אחרים כאשר התוצאות הרלוונטיות ביותר מופיעות בראש העמוד.
- מבוסס לפי פקודות ה-HTTP כמו GET, POST PUT וכו'...
- כמובן שגם זה NoSQL

הוספת מסמך Document (פריט/רשומה).

אין צורך ליצור אינדקס (מסד נתונים) או סוג (טבלה), אנו יכולים מיד להוסיף מסמך.

דוגמה	elasticsearch

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

משיכת נתון מהמסד באמצעות GET

דוגמה	elasticsearch

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

- הפקודה **HEAD** תעבוד רק אם המסמך קיים (בשונה מ-GET).
- הפקודה **DELETE** תמחק את המסמך הרלוונטי.

שימוש בחיפוש Search כלומר SELECT

דוגמה	elasticsearch

```
curl –XGET "http://localhost:9200/university/students/_search"
```

תחזיר לנו כל תוצאה מתוך הסטודנטים ובנוסף תציג את הניקוד שלהם.

במקרה הזה נקבל בפלט ניקוד שווה בין כל התוצאות מכיוון שלא באמת חיפשנו משהו ספציפי.

דוגמה נוספת לחיפוש עבור משהו ספציפי יותר	elasticsearch

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

נקבל בפלט:

```
{"took":21,"timed_out":false,"_shards":{"total":5,"successful":5,"failed":0},
"hits":{"total":1,"max_score":0.2876821,"hits":[{"_index":"university","_type
":"students","_id":"222","_score":0.2876821,"_source":{"FirstName": "Tal",
"LastName": "Negev", "age": "28"}}]}}
```

חיפוש מתקדם בעזרת bool query

נרצה להשתמש בו עבור תוצאות שנרצה שבהכרח יופיעו בצורה במסוימת או בשילוב של כמה תוצאות ביחד.

דוגמה נוספת לחיפוש עבור משהו ספציפי יותר	elasticsearch

```
 curl -XGET "http://localhost:9200/university/students/_search" -d"
{ \"query\" :
  { \"bool\" :
    { \"filter\" :
      { \"match\" :
        { \"Address.City\" : \"Ariel\" } },
     \"filter\" :
      { \"range\" :
        { \"age\" :
          { \"lt\" : 30 }
} } } } } "
```

- ה-query יוצר שאילתת חיפוש.
- ה-filter הוא פשוט האופן שבו נרצה למצוא את התוצאות.
- ה-match נועד כדי להגדיר איזה התאמה בחיפוש נרצה (כמו למשל מילה כלשהי).
- ה-range נועד לאפשר טווח תוצאות אפשריות כמו למשל בדוגמה עבור כל גיל שקטן מ-30.
- ה-lt כלומר less than.

הערות

Elasticsearch מחזיר את המסמכים עם הציון הגבוה ביותר.

כפי שניתן לראות, כששאילתות עם "filter" אין ציון מצטבר. (גם ל- "must_not" אין ציון).

Elasticsearch תומך גם ב"must "ו"should". שניהם משפיעים על הציון.

"must" ו- "should" יכולים להופיע גם בתוך "filter", ובמקרה כזה "must" הוא "AND", ואילו "should" הוא "OR" . ציון הופך להיות מעניין כשאנחנו מתחילים להתאים מחרוזות.

שאילתות על מחרוזות

אם אנו רוצים את הביטוי המדויק, נוכל להשתמש ב" match_phrase "במקום ב"match".
אנחנו יכולים גם לערבב "match" ו- "match_phrase" ביחד.

elasticsearch	דוגמה לשאילתה פשוטה

```
-curl -XGET "http://localhost:9200/university/students/_search?
q=description:\"works hard\""
```

elasticsearch	דוגמה לשאילתה פשוטה

```
curl -XGET http://localhost:9200/university/students/_search -d "{ \"query\" : {
\"match\" : { \"description\" : \"doesn't show-up learns\" }    }}
```

שימוש ב-TF-IDF

נניח שיש לנו שאילתה (Q) ומסמכים אפשריים רבים $D = \{d_1, d_2, d_3, \ldots\}$.

כיצד נוכל לדרג את המסמכים הללו על סמך השאילתה?

נעזר בדוגמה הבא לצורך הסבר:

השאילתה:

Q: Who is the president of the united states?

המסמכים:

d_1: Donald Trump is United States' president.

d_2: We are the most united out of all the people and of all the places.

d_3: The United States of America is united again, who is more united than it?

d_4: Who would like to take the box out of the kitchen?

הנוסחא

$$TF\_IDF(d_i) = \sum_{k=0}^{|Q|} \frac{\#k \text{ in } d_i}{|d_i|} \cdot \log\left(\frac{|D|}{\#D \text{ with } k}\right)$$

TF: תדירות מונח. ספירה פשוטה של כמה פעמים מילת המפתח הנתונה מופיעה במסמך מנורמל לפי מספר המילים במסמך, כלומר $\frac{\#k \text{ in } d_i}{|d_i|}$.

IDF: תדירות מסמכים הפוכה: המסמכים הכוללים חלקי מספר המסמכים בהם מופיע המונח, כלומר $\log\left(\frac{|D|}{\#D \text{ with } k}\right)$.

ה-$|D|$ זה מספר כל המסמכים

ה-$|Q|$ זה אוסף כל המילים בשאילתה.

ה-d_i זה המסמך הנוכחי

וה-k היא המילה בתוך השאילתה או המסמך הנתון.

נחשב בטבלה לנוחות הפתרון

נאסוף את הנתונים (כתרגיל) ומספר המופעים של כל מילה במסמך לתוך הטבלה הבאה:

	Who	Is	The	President	Of	United	States	#words
d_1	0	1	0	1	0	1	1	6
d_2	0	0	3	0	2	1	0	15
d_3	1	2	1	0	1	3	1	14
d_4	1	0	2	0	1	0	0	11
#Doc	2	2	3	1	3	3	2	

ולפי הטבלה הקודמת ניצור טבלת חישוב עבור כל מסמך כדי לחשב את הרלוונטי ביותר. את החישוב נבצע לפי הנוסחא שהוצגה מקודם.

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

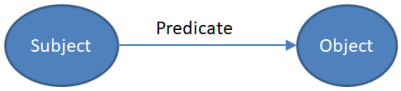
Doc	Tf-Idf score
d_1	(1/6)*log(4/2)+(1/6)*log(4/1)+(1/6)*log(4/3)+(1/6)*log(4/2)=0.736
d_2	(3/15)*log(4/3)+(2/15)*log(4/3)+(1/15)*log(4/3)=0.166
d_3	(1/14)*log(4/2)+(1/14)*log(4/2)+(1/14)*log(4/3)+(1/14)*log(4/3)+(3/14)*log(4/3)+(1/14)*log(4/2)=0.363
d_4	(log(4/2)+2*log(4/3)+log(4/3))/11=0.204

לכן התוצאות הם (מהרלוונטי ביותר):

1. d_1
2. d_3
3. d_4
4. d_2

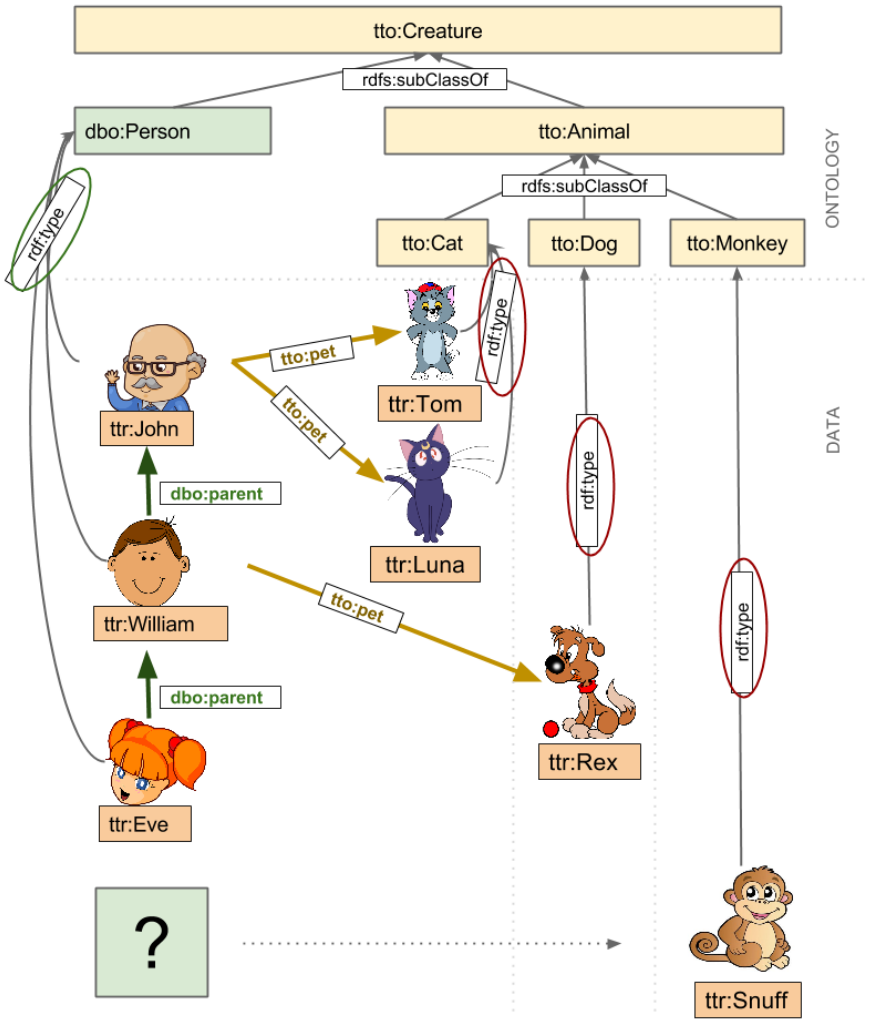
בסיס הנתונים RDF

- אחסון נתונים בצורה של גרף שעשוי משלשות.
- השלשה בנויה מsubject , predicate , objects.
  כלומר 1. מי, 2. באיזה אופן , 3. מה?
  כמו למשל "יוני אוהב פיצה", אז 1.יוני , 2. אוהב, 3. פיצה.



- קישור לסימולציה
- יש רק "טבלה" יחידה, המכילה את כל השלשות הללו.
- אנו נתמקד ב- **SPARQL** שהיא שפת שאילתות ה- RDF.
- נלמד רק בחירה.

נעזר באיור הבא לטובת הדוגמאות הבאות



SPARQL	דוגמה לשימוש בבחירה

```
SELECT * WHERE {?s ?p ?o}
```

נקבל כפלט:

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"
...	...	....

SPARQL	שליפת כל האנשים

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

נקבל:

something
ttr:Eve
ttr:John
ttr:William

שליפת כל הנשים — **SPARQL**

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

thing
ttr:Eve

שליפת כל האנשים שיש להם בעלי חיים — **SPARQL**

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

נשים לב שקיבלנו פעמיים John
זה מכיוון שיש לו 2 בעלי-חיים.

אפשר לבטל את הכפילות בעזרת DISTINCT.

person
ttr:John
ttr:John
ttr:William

שליפת כל האנשים שאין להם בעלי חיים בכלל — **SPARQL**

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

person
ttr:Eve

שליפת האנשים שאין להם חתולים	SPARQL

```
SELECT ?person
WHERE {
  ?person rdf:type dbo:Person .

  FILTER NOT EXISTS {
  ?person tto:pet / rdf:type tto:Cat .
  }
}
```

הסימון "/" מקשר/משרשר בין יחסים.

person
ttr:Eve
ttr:William

שימוש ב-UNION

שליפת כל דבר שהוא creature או תחת הקטגוריה הזאת.	SPARQL

```
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

הערה - הסימון "a" שמופיע במודגש הוא דרך קיצור ב-SPARQL עבור rdf:type

צורה תקינה ונכונה יותר עבור הדוגמה הקודמת

שליפת כל דבר שהוא creature או תחת הקטגוריה הזאת.	SPARQL

```
SELECT ?thing

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

כאשר הסימון "+" הכוונה לרמה אחד מעל בהיררכיה או יותר מזה.

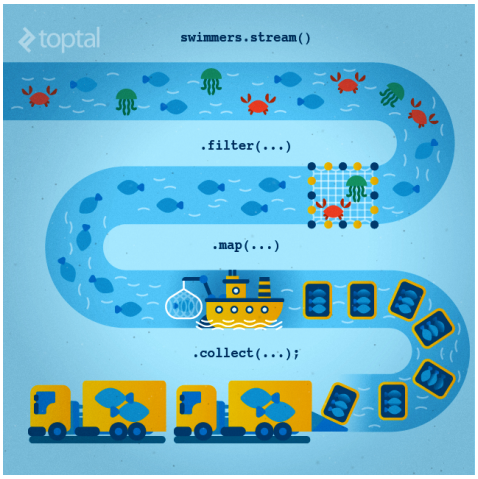
אפשר במקום להחליף באחד מהסימונים:

הסימון "*" אומר מ-0 או יותר מזה.

הסימון "?" אומר או 0 או 1 בלבד.

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

Java Stream



מה זה Stream

- Stream הוא כלי/ספרייה בג'אווה המאפשר לנו להתייחס לאוספים כאל רצפים ומאפשר לנו לבצע פעולות וחישובים על האלמנטים הנמצאים באוסף.
- אוסף יכול להיות מהצורה של מערך, רשימה, מפה..
- הפעולות יכולות להשפיע על כמות האלמנטים, על סוג האלמנטים או על הסדר שלהם.

איך Stream עובד?

1. **stream** - ניתן להפעיל על מבני נתונים מסוג מערך או רשימה, סטים ועוד...
   a. עבור רשימה פשוט נשתמש: myList.stream()
   b. עבור מערך נשתמש ב-Arrays כלומר: Arrays.stream(myArray)

2. **map** - פעולת ביניים המאפשרת לנו להמיר אובייקט למשהו אחר או לשנות את צורתם לפי הדרישות שלנו:
   a. קלט - פונקציית למבדה שעוברת על כל אלמנט ב-Stream ויכולה לבצע עליה שינויים.
   b. פלט - מחזיר Stream המורכב מתוצאות הפעלת הפונקציה הנתונה לאלמנטים ב-Stream הזה.

Java Stream

```
List number = Arrays.asList(2,3,4,5);
List square = number.stream().map(x->x*x).collect(Collectors.toList());
```

3. **collect** - משמשת להחזרת התוצאה של פעולות הביניים שבוצעו בזרם.
   - היא הופכת את כל האלמנטים שנמצאים ב-Stream לסוג אחר של נתונים כמו למשל List, Map וכו'.
   - ה-collect מקבל אובייקט שנקרא Collectors ולו יש כל מיני אופציות כמו: ToList, groupingBy וכו'.

Java Stream

```
List<Integer> number = Arrays.asList(2,3,4,5,3);
Set<Integer> square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

4. **filter** - פעולת ביניים המאפשרת לנו לסנן אלמנטים לפי הדרישות שלנו:
   a. קלט - פונקציית למבדה בוליאנית.
   b. פלט - מחזיר Stream המורכב מאלמנטים התואמים את הקביעה הנתונה.

Java Stream

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
```

5. **sorted** - פעולת ביניים/סופית המחזירה Stream המורכב מאלמנטים של זרם זה, ממוין לפי הסדר הטבעי.
עבור סדר הפוך אפשר להשתמש בקומפרטור כלומר list.stream().sorted(Comparator.reverseOrder())

Java Stream

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().sorted().collect(Collectors.toList());
```

6. **forEach** - פעולה המבוצעת בסוף לכל רכיב ב-Stream שלנו.

Java Stream

```
List number = Arrays.asList(2,3,4,5);
number.stream().map(x->x*x).forEach(y->System.out.println(y));
```

7. **reduce** - לוקחת רצף של אלמנטים ומאחדת אותם לתוצאה לפי הפעולה שנגדיר כמו: sum, max, count.

Java Stream

```
List number = Arrays.asList(2,3,4,5);
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);
```

8. **anyMatch** - פונקציה בוליאנית שנועדה לבדוק אם אלמנט **כלשהו** ברשימה עומד בתנאי נתון.

Java Stream

```
List<Integer> list = Arrays.asList(3, 4, 6, 12, 20);
boolean answer = list.stream().anyMatch(n -> n % 2 == 0);
```

9. **allMatch** - פונקציה בוליאנית שנועדה לבדוק אם **כל** האלמנטים ברשימה עומדים בתנאי הנתון.

דוגמאות

Java Stream

```
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);
```

Java Stream — **דוגמה להדפסת מערך**

```
int[] arr = {1, 2, 45, 78, 3, 48, 23, 105, 5, 15};
Arrays.stream(arr).forEach(x-> System.out.print(x + " "));
```

פלט:

```
1 2 45 78 3 48 23 105 5 15
```

Java Stream — **להחזיר את כל האיברים הקטנים מ-10**

```
int[] arr = {1, 2, 45, 78, 3, 48, 23, 105, 5, 15};
Arrays.stream(arr).filter(s -> (s < 10)).forEach(x-> System.out.print(x + " "));
```

פלט:

```
1 2 3 5
```

Java Stream — **מיון סדר לקסיקוגרפי**

```
String[] arr = {"e", "d", "c", "b", "a"};
Arrays.stream(arr).sorted().forEach(x-> System.out.print(x + " "));
```

פלט:

```
a b c d e
```

שימוש ב-Collect

- פעולה מאוד שימושית על Streams
- היא הופכת את כל האלמנטים שנמצאים ב-Stream לסוג אחר של נתונים כמו למשל map, list וכו'..
- ה-Collect מקבל אובייקט שנקרא collector ולו יש כל מיני אופציות כמו: ToList, groupingBy וכו'...

שימוש ב-collectors	Java Stream

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

בדוגמה הזאת השתמשנו ב-collector מסוג **summarizingInt**, תפקידו לעשות סיכום כללי על פעולות בסיסיות שנשאלות בדרך כלל עבור סדרת מספרים כלשהי, כמו למשל מהו האיבר המקסימלי, מהו הסכום הכולל, מה הממוצע של המערך וכו'...

פלט:

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

שימוש ב-collectors עם joining	Java Stream

```
Stream.of("Hello", "world") .collect(Collectors.joining(" - "));
```

פלט:

```
Hello - world
```

הערה

`anyMatch()` פונקציה כדי לבדוק אם רכיב כלשהו ברשימה עומד בתנאי נתון.

`allMatch()` פונקציה כדי לבדוק אם כל הרכיבים ברשימה עומדים בתנאי נתון.

`findFirst()` פונקציה שמחזירה את הרכיב הראשון ברשימה בתוך הסטרים.

`findAny()` פונקציה שמחזירה את כל הרכיבים בסטרים.

דוגמה	Java Stream

```
String[] arr = {"ee", "dd", "aaac", "db", "asa"};
Boolean result = Arrays.stream(arr).anyMatch(x->x.length()==2);
System.out.println(result);
```

פלט:

```
true
```

בדוגמה הבאה, אוספים את כל הציונים של כל סטודנט ומדפיסים את הממוצע של כל אחד	Java Stream

```
students.stream().forEach(s ->{ double average = s.getGrades().stream()
.mapToDouble(g ->g.getValue()) .average().getAsDouble();
System.out.println( s.getName() + " average: " + average); });
```

דוגמה - תדפיס את השרשור העצמי של כל מחרוזת רק אם השרשור ארוך יותר מ-7 כולל	Java Stream

```
List myList = Arrays.asList ("camel", "zebra", "you", "apple", "banana", "me");

myList.stream().map(a->a+a).filter(a->a.length()>=7)
.sorted().forEach(System.out::println);
```

פלט:

```
appleapple
bananabanana
camelcamel
zebrazebra
```

תכנות מקבילי

נרצה להשתמש כאשר:

1. לא אכפת לנו מסדר העיבוד.
2. עיבוד של כל פריט בקוד לוקח הרבה מאוד זמן
3. יש לנו threads זמינים לשימוש.

במקרה של עיבוד מקבילי נרצה להשתמש ב-**parallelStream** במקום ב-Stream.

דוגמה לשימוש ב-parallelStream	Java Stream

```
Random r = new Random();
List<Long> numberList = new LinkedList<>();
for (int i = 0; i < 1000; i++) {
    numberList.add(r.nextLong());
}
long start = System.nanoTime();
long numberOfPrimes =
    numberList.parallelStream().filter(a -> naivePrimeTest(a)).count();
long end = System.nanoTime();
System.out.println("numberOfPrimes: " + numberOfPrimes + ". time: " +
((end-start)*1E-9) + " seconds");
```

מציאת מספרים ראשוניים בשיטה הרבה יותר מהירה	Java Stream
`long numberOfPrimes = numberList.stream().filter(a -> BigInteger.valueOf(a).isProbablePrime(100)).count();`	

שימוש ב-Reduce

פונקציית reduce לוקחת רצף של אלמנטים ומאחדת אותם לתוצאה לפי הפעולה שנגדיר.

פעולות מוכרות: sum, max, count.

דוגמה לשימוש ב-reduce	Java Stream
`Stream.of(1, 45, 6, 23, 8).reduce(Integer::sum);`	

ניתן לכתוב גם פונקציות reduce משלנו במבנה מהצורה הבאה:

`reduce([identity], accumulator, [combiner])`

- **Accumulator** : פונקציית למבדה עם 2 ערכי קלט: (התוצאה עד עכשיו, האלמנט הנוכחי).
  הפונקציה מחזירה את התוצאה החלקית אחרי הוספת האלמנט הנוכחי.
- **Combiner** : פונקציית למבדה עם 2 ערכי קלט: (2 תוצאות חלקיות).
  הפונקציה מחזירה את התוצאה החלקית של שניהם יחד.
- **Identity** :אלמנט התחלתי שלא משנה את התוצאה (לדוגמא 0 במקרה של סכום, 1 במקרה של הכפלה).

הערה - אם לא משתמשים ב-combiner אז ה-accumulator יחליף את התפקיד שלו.

למשל, הפונקציה count נראית מהצורה הבאה:	Java Stream
`...reduce(0, (x,t)-> x+1, (x,y) -> x+y)`	

פונקציית reduce המקבלת רצף של מספרים ומחזירה את המכפלה של כל המספרים ההופכיים להם.	Java Stream
`Stream.of(1.0,2.0, 2.0, 3.0).reduce(1.0, (x,y) -> (x*(1/y)));`	

פלט: 0.0833333

- הערה: parallelStream לא תומך במערכים, כלומר לא יהיה אפשר לעשות combiner ב-reduce על מערכים.
  הפתרון הוא להמיר את המערך לרשימה ואז יהיה אפשר לעשות parallelStream.
- לדוגמה:

`arr.stream().boxed().collect(Collectors.toList()).parallelStream().reduce(....`

שימוש ב-IntStream

רצף של אלמנטים מסוג Int על זרם, נרצה להשתמש בהם על range מסוים של סדרת מספרים כלשהי.

- הפונקציה **range** - מקבלת ערך התחלתי וערך סוף ומחזירה IntStream ברצף מהערך התחלה (כולל) ל- לערך סוף (**לא** כולל) בצעד מצטבר של 1.
- הפונקציה **rangeClosed** - מקבלת ערך התחלתי וערך סוף ומחזירה IntStream ברצף מהערך התחלה (כולל) ל- לערך סוף (**כולל**) בצעד מצטבר של 1.

Java Stream	כיתבו קטע קוד ב-java streams המחזיר במשתנה answer את הערך של n!

```
int n = 5; // 120
int answer = IntStream.rangeClosed(1,n).reduce(1,(x,y)->x*y);
System.out.println(answer);
```

Java Stream	סכום המספרים המתחלקים ב-3 או ב-7 (ללא שארית) בין המספרים 1 ל- 10,000

```
IntStream().rangeClosed(1,10000)
           .filter(x -> x % 3 == 0 || x % 7 == 0)
           .reduce(0, Integer::sum);
```

שאלת מבחן

Write a Java streams reduce function that given a stream of non-zero numbers will return the following result: $\prod_{i=0}^{N} \frac{1}{x_i}$

(in words: you need to multiply the inverse of all numbers in the stream. E.g. {1, 2, 2, 3} -> $\frac{1}{12}$)

You may only use a reduce function, not any other functions (i.e. no map, filter, etc.)

Java Stream	כתבו רדיוס שמקבלת מספרים ומחזירה את המכפלה של כל המספרים ההופכיים להם

```
num = list.parallelStream().reduce(1, (x,y) -> (1/x*y) , (p,q) -> (p*q));
System.out.println(num)
```

Spark

ה-Spark מאפשר ביצוע של חישובים תוך שמירה על רמה גבוהה של מקביליות וניצול מירבי של משאבי המחשב.
הביצועים של Spark באים לידי ביטוי בעיקר במחשבים מרובי ליבות.
חישובים אופייניים שנבצע באמצעות Spark הם חישובים המורכבים מהרבה צעדים בלתי תלויים ביניהם שירוצו במקביל ולבסוף נאחד את התוצאות.
את הדוגמאות לנושא זה נציג בשפת **Python**.

RDD

ה-RDD הוא אוסף נתונים גמיש ומבוזר.

- אוסף יחידות או פריטים מקיימים ביניהם חוסר תלות ולכן ניתנים לחישוב מקבילי.
  לדוגמא: שורות שונות בקובץ, אוסף של נתוני סטודנטים וכו'..
- אפשר ליצור RDD מקובץ או מרשימה קיימת באמצעות שימוש בפונקציות המיועדות לכך ב Spark .
  ניתן לבצע שרשרת של טרנספורמציות כאשר כל אחת מהן מניבה RDD חדש.
  ובסוף נבצע איסוף לתוך מבנה נתונים פשוט כמו רשימה על-ידי collect.

Spark	ספירת מופעים של מילים בקובץ txt

```
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))
```

פלט:

```
the word: "retrograde" appears 1 time(s)
the word: "grounds" appears 4 time(s)
the word: "VOUS" appears 2 time(s)
the word: ""Flatterers" appears 1 time(s)
the word: "injustice;" appears 1 time(s)
the word: "reciprocity," appears 1 time(s)
```

השימוש ב-flatMap ממפה כל ערך ברשימה ואז מחזיר רשימה של RDD.

Spark | **מיון לפי מפתח בעזרת sortByKey**

```
word_counts = text_file.flatMap(lambda line: line.split(" "))
                       .map(lambda word: (word, 1))
                       .reduceByKey(lambda a,b: a+b)
                       .sortByKey()
                       .collect()

for word,count in word_counts[0:15]:
                     print("the word: \"%s\" appears %d time(s)" %(word,count))
```

פלט:

```
the word: "" appears 2032 time(s)
the word: ""=Man" appears 1 time(s)
the word: ""A" appears 2 time(s)
the word: ""AWAY" appears 1 time(s)
the word: ""Ah," appears 1 time(s)
the word: ""All" appears 1 time(s)
the word: ""And" appears 2 time(s)
the word: ""Another" appears 1 time(s)
```

Spark | **מיון לפי ערך**

```
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)
                        .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))
```

הסבר: כדי למיין לפי ערך, אנחנו נעשה swap בין ה-key לבין ה-value, ונמיין כמו שעשינו בדוגמה הקודמת למפתחות, ואחר כך נעשה שוב swap כדי להחזיר למצב הראשון שהיה.
ה-False ב-sortByKey זה כדי לציין כי המיון מתבצע בסדר **יורד**.

פלט:

```
the word: "the" appears 5839 time(s)
the word: "of" appears 4560 time(s)
the word: "and" appears 3562 time(s)
the word: "to" appears 2716 time(s)
the word: "" appears 2032 time(s)
```

Bi-Grams

בעיבוד שפה טבעית, אנו לרוב מתעניינים בהופעה של זוגות מילים (דו-גרם).
לדוגמא, למשפט הבא: "I did it you did it you did it"
אנו מקבלים את ספירת הבי-גרם הבאה:

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

הצמדי-מילים שלנו יהיה מהצמד של כל מילה והמילה שאחריה. **זה שימושי כשנרצה לנתח משפטים.**
כמובן שאפשר גם להתייחס ל-Tri-Grams ובמקרה הכללי גם עבור N-Grams.

zip היא פונקציה מובנית בפייתון, המקבלת שתי רשימות (או יותר) ומחזירה רשימה של tuples משתי הרשימות.

דוגמה ל-zip

Spark

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

דוגמה לספירת ה-BiGrams

Spark

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

❷>>>pairs = text_file.flatMap(BiGram)
❸>>>count = pairs.map(lambda x: (x, 1)).reduceByKey(lambda a, b: a + b)

>>>print(count.collect()[0:10])
❹>>>print(count.map(lambda xy: (xy[1],xy[0])).sortByKey(False)
             .map(lambda xy: (xy[1],xy[0]))
             .collect()[0:10])
```

❶ כל שורה line שנקבל לפונקציה BiGram, תהפוך לרשימה של מילים, ויוצר רשימה של טאפלים של המילים.
לדוגמה - עבור השורה "hello how are you" הפונקציה split תחזיר לנו את הרשימה: [hello, how, are, you]
והפונקציה תחזיר בעזרת zip את:
zip[[hello, how, are, you], [how, are, you]] = [(hello,how), (how,are), (are,you)]
❷ בתוך pairs נכנס לנו כל הזוגות מכל השורות (זה לפי flatMap).
❸ מוסיף לכל זוג את הספרה "1", כלומר לכל הצמדים יהיה (1, (hello,how)) ועכשיו, המפתח יהיה ה-tuple והערך יהיה פשוט הספרה 1, ומכאן לפי reduceByKey הוא יחפש את כל ה-tuple מאותו הצורה ובמידה ויש הוא יחבר אותם (כלומר סופר את המופעים של אותו צמד-מילים).
❹ דוגמה לפלט מסודר:

```
[(('of', 'the'), 910), (('in', 'the'), 498), (('to', 'the'), 327), (('it', 'is'),
240), (('to', 'be'), 186), (('of', 'a'), 171), (('and', 'the'), 157), (('for',
'the'), 149), (('that', 'the'), 138), (('is', 'the'), 131)]
```

Spark	ממשו את הפונקציה Grams_N המקבלת רשימה a ומספר n

```
def func (a,n):
    return zip(*[a[i:] for i in range(n)])
print(list(func([1,2,3,4,5,6,7,8,9], 4)))
```

פלט:

```
[(1,2,3,4), (2,3,4,5), (3,4,5,6), (4,5,6,7), (5,6,7,8), (6,7,8,9)]
```

שימוש ב-Spark באמצעות פייתון

- In order to import Spark into Python we need first to define the environment variable SPARK_HOME:
  `-export SPARK_HOME=.`
- Then we need to install findspark
  `-pip install findspark`
- Then, in the script we write:
  `-import findspark`
- findspark.init()
  `-from pyspark import SparkContext`
  `-sc = SparkContext("local[*]", "Simple App")`

Spark	דוגמה לחישוב Pi (התיאוריה עצמה לא קשור לקורס)

```
import random
samples = 10**8
def inside(p):
         x, y = random.random(), random.random()
         return x*x + y*y < 1

hits = sc.parallelize(range(0, samples)).filter(inside).count()

print("Pi is approx. %f" % (4.0 * hits / samples))
```

נרצה לחשב את פאי בעזרת שיטה שנקראת "מונטה-קרלו" (לא קשור לחומר).

1. על לוח עץ נצייר ריבוע שאורך צלעו שתי יחידות. נצייר מעגל חסום בריבוע זה (זהו מעגל שרדיוסו שווה ליחידה אחת)
2. נתחיל להטיל חצים אל הריבוע (לא נכוון את החץ למרכז הריבוע, אלא אל הריבוע כולו, באופן אקראי). פונקציית ההסתברות במקרה זה היא אחידה, כלומר יש הסתברות שווה לפגיעה בכל נקודה על פני הריבוע.
3. כל חץ שפגע נבדק האם הוא בתוך העיגול או מחוץ לעיגול (אך עדיין בתוך הריבוע). במקרה זה, זהו החישוב הדטרמיניסטי שמבוצע.
4. נבדוק כמה חיצים היו בתוך העיגול מתוך כמות החצים הכוללת.

Data Frames

עד עכשיו ראינו שימוש ב-RDD, אבל יש ב-Spark עוד שיטה לעבוד איתה שנקראת Data Frames שזה בעצם אפשרות להתייחס ל-Spark "כאילו" הוא DB רלציוני (כלומר טבלאי).

ה-Data Frame נותן אפשרות לעבוד על משהו שיותר מסודר בצורת טבלה עם תכונות המתארות גוף נתונים מסוים.

אפשר ליצור Data Frames על-ידי:

1. מסד נתונים רלציוני
2. עם RDD
3. קבצי XML או JSON

אנחנו ממש יכולים להשתמש בקוד של SQL בשביל לחפש נתונים מסוימים בתוך ה-Data Frame .

דוגמה לשימוש ב-Spark עם שיטת ה-Data Frame — **Spark**

```
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)]

❶ student_rdd = sc.parallelize(student_list)
❷ students_rows = student_rdd.map(lambda x: Row(id=int(x[0]),age=int(x[3]),
                                 firstName=x[1], lastName=x[2]))
❸ df_students = sqlContext.createDataFrame(students_rows)

df_students.show()
```

❶ נהפוך את הרשימה של הסטודנטים לרשימת RDD (כלומר רשימת tuple עם המידע הנתון).

השימוש ב-parallelize היא פונקציה ב-PySpark שנועדה ליצור RDD מרשימה כלשהי ולהחזיר את ה-RDD.

❷ נרצה להפוך את ה-RDD ל- Data Frame כיוון שהוא מתנהג כמו טבלה, אז צריך לעשות איזשהי לוגיקה לנתונים כמו עמודות או שורות ולכן נמפה כל גוף נתון לצורה הדומה לשורה בטבלה.

❸ ברגע שיש לנו שורות שהם לא רק RDD אלא יש להם מיפוי של נתונים, אז יהיה אפשר להשתמש ב- Data Frame וברגע שנעשה show נקבל בצורה מסודרת וטבלאית את כל מה שהיה לנו ברשימה:

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

Spark | **אפשרות נוספת ומהירה יותר בעזרת JSON**

```
students_json =
'[{"id":"111","firstName":"Chaya","lastName":"Glass","age":23},{"id":"222","firstNa
me":"Tal","lastName":"Negev","age":28},{"id":"333","firstName":"Gadi","lastName":"G
olan","age":24},{"id":"444","firstName":"Moti","lastName":"Cohen","age":23}]'

df = sqlContext.read.json(sc.parallelize([students_json]))
df.show()
```

Spark | **אפשרות עם JSON באופן ישיר יותר**

```
df = sqlContext.read.json("myDir\students.json", multiLine=True)
df.show()
```

השימוש ב-**multiLine** מאפשר קריאת קובץ JSON הבנוי בצורת רב שורות.
כלומר, ישנם קבצי JSON שבוניים בצורה של שורה אחת ארוכה, לכן במקרה זה נרצה שהוא יקרא את הקובץ בצורה של מספר שורות.

הפונקציה withColumn מאפשרת לנו להוסיף נתונים.
למשל יש לנו ביד data frame, הפונקציה תאפשר לנו **להוסיף עוד עמודות** מבלי לפגוע בנתונים המקוריים.

Spark | **דוגמה לשימוש ב-withColumn**

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

כלומר בדוגמה הזאת, נוספה לנו עמודה שתשמור לנו את הסה"כ של ה-age עם ה-id (בלי היגיון כלשהו).
פלט:

```
+---+---------+---+--------+-----------+
|age|firstName| id|lastName|age_plus_id|
+---+---------+---+--------+-----------+
| 21|   Chaya|111|   Glass|        132|
| 28|   Tal|222|   Negev|        250|
| 24|   Gadi|333|   Golan|        357|
| 23|   Moti|444|   Cohen|        467|
+---+---------+---+--------+-----------+
```

שימוש בפונקציה הזאת חייבת להיות מצורה של 2 פרמטרים כאשר הראשונה זה השם של העמודה (string) והשני חייב להיות מסוג של column כלשהו ולכן אם נרצה לעשות חישובים של עמודות אחרות זה יקרה מראש כיוון שהעמודות כבר נתונות לנו.

אבל אם נרצה להביא מידע משלנו, אנחנו צריכים להמיר את המידע הזה לסוג של column.
לדוגמה:

Spark | **דוגמה לשימוש ב-withColumn**

```
from pyspark.sql import functions
from pyspark.sql.functions import lit  #lit for literal

df_students2 = df_students.withColumn('young', functions.when(df_students.age < 25,
True).otherwise(False)).withColumn('max_grade', lit(100))

df_students2.show()
```

בדוגמה תהיה לנו עמודה חדשה בשם young כאשר הערך בתוכה יהיה או True או False .
הוא בודק את התנאי עבור כל רשומה שיש לו ב-data frame ומציב ערך בוליאני לפי דרישות התנאי.
בנוסף עמודה max_grade מציבה את הערך 100 לכל סטודנט בכל רשומה והשימוש בפונקציה lit בעצם נועד להציב נתון מספרי מסוג column.
פלט:

```
+---+---------+---+--------+-----+---------+
|age|firstName| id|lastName|young|max_grade|
+---+---------+---+--------+-----+---------+
| 21|     Chaya|111|   Glass| true|      100|
| 28|      Tal|222|   Negev|false|      100|
| 24|     Gadi|333|   Golan| true|      100|
| 23|     Moti|444|   Cohen| true|      100|
+---+---------+---+--------+-----+---------+
```

SQL Query in Spark

אנחנו יכולים לקחת את ה-Data Frame ולהפוך אותו ל-"כאילו" טבלת SQL כך שניתן לו פקודות SQL והוא פשוט יבצע אותם.

Spark | **לדוגמה**

```
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()
```

```
+---------+--------+---+
|firstName|lastName| id|
+---------+--------+---+
|     Gadi|   Golan|333|
|     Moti|   Cohen|444|
+---------+--------+---+
```

תרגיל

נתון אובייקט שמחזיק את כל ההזמנות ונרצה לסנן ולמצוא את הסכום הכולל ששולם עבור כל אחד מהלקוחות שלנו שעומדים בסטטוס A.

Spark	פתרון ב-Spark עם RDD באמצעות map-reduce

```
df = sqlContext.read.json("MyDir\orders.json", multiLine=True)
rdd = df.rdd
calc_amount = rdd.filter(lambda a: a.status == 'A') \
                 .map(lambda row: (row.cust_id, row.amount)) \
                 .reduceByKey(lambda a,b: a+b) \
                 .collect()

for cust_id, sum_amount in calc_amount:
    print('%s : %s' %(cust_id, sum_amount))
```

נמשוך את הנתון בעזרת JSON, נהפוך לצורת RDD ואז נבצע את הסינון של הסטטוס.
ה-map בעצם אספנו מכל רשומה את מספר לקוח ואת הסכום וצמד ואז נסכום את כל ה-amount של אותו מספר לקוח.

Spark	פתרון ב-Spark עם Data Frame בשאילתות SQL

```
df = sqlContext.read.json("myDir\orders.json", multiLine=True)
df.registerTempTable("orders_table")

amount = sqlContext.sql("SELECT sum(amount), cust_id FROM orders_table WHERE status
== 'A' group by cust_id")

amount.show()
```

בשיטה שמוכרת לנו כבר, מבחינת זמן ריצה השיטה הזאת מהירה יותר מהקודמת.

Spark	פתרון ב-Spark עם Data Frame בפקודה ישירה

```
from pyspark.sql import functions

df.filter(df.status == 'A').groupBy(df.cust_id).agg(functions
        .sum(functions.col("amount"))).show()
```

סיכום שימוש בשיטות עבור Spark

Data Frames	RDD
"**מה**" לעשות. לא נשתמש בלמבדות אלא פשוט נשתמש בפונקציות מוכנות מראש.	"**איך**" לעשות. כלומר **באיזה צורה** להשתמש בנתונים (filter, map…).
נתונים המאורגנים במבנים מסודרים יותר.	נתונים חסרי מבנה. כלומר זה כמו סיפור, אין מידע מאורגן.
יותר אופטימלי וביצועים יעילים.	פחות אופטימלי ויעיל

ההבדל בין flatMap לבין map

Map Transformation

```
val mapDF=df.map(fun=> {
    fun.getString(0).split(" ")
})
mapDF.show()
```

output

```
+---------------------------------+
|value                            |
+---------------------------------+
|[Project, Gutenberg’s]           |
|[Alice’s, Adventures, in,
  Wonderland]                     |
|[Project, Gutenberg’s]           |
|[Adventures, in, Wonderland]     |
|[Project, Gutenberg’s]           |
+---------------------------------+
```

flatMap Transformation

```
val flatMapDF=df.flatMap(fun=>
  {
     fun.getString(0).split(" ")
  })
flatMapDF.show()
```

output

```
+-----------+
|      value|
+-----------+
|    Project|
|Gutenberg’s|
|    Alice’s|
| Adventures|
|         in|
| Wonderland|
|    Project|
|Gutenberg’s|
| Adventures|
|         in|
| Wonderland|
|    Project|
|Gutenberg’s|
+-----------+
```

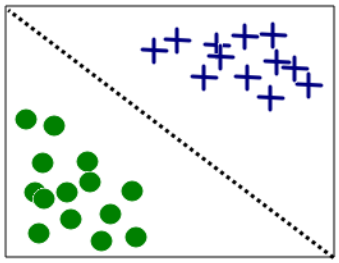
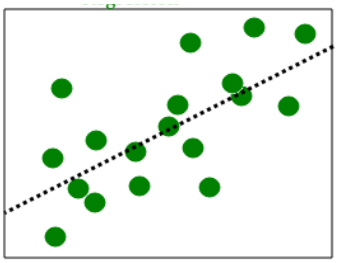
מבוא ללמידת מכונה

הקשר בין מסדי נתונים ללמידת מכונה הוא כאשר חסר לנו מידע מסוים.
כאשר אין לנו מידע מסוים ודווקא אותו נרצה לשלוף.
לדוגמא, נניח שחלק מהמשתמשים שלנו מתויגים כמועסקים או מובטלים, והמערכת שלנו צריכה להחליט אם להציג מודעה שרלוונטית רק למשתמשים מובטלים.

- **מודל** - ייצוג ספציפי שנלמד מנתונים על ידי יישום אלגוריתם למידת מכונה כלשהו. מודל נקרא גם השערה.
- **לייבל**- לייבל הוא הדבר שאנחנו חוזים - המשתנה y. הלייבל יכול להיות המחיר העתידי של חיטה, סוג החיה המוצגת בתמונה, משמעות קליפ שמע או כמעט כל דבר אחר.
- **פיצ'ר/תכונה** - פיצ'ר הוא משתנה הקלט - המשתנה x. למידת מכונה פשוטה עשויה להשתמש בפיצ'ר אחד, ואילו למידת מכונה מתוחכמת יותר יכולה להשתמש במיליוני פיצ'רים, המוגדרים כך: $x_1, x_2, ..., x_N$.
  לדוגמה - מודל שמזהה הודעות ספאם, הפיצ'רים (תכונות) יכולים להיות הדברים הבאים:
  - מילים בטקסט הדוא"ל.
  - כתובת השולח.
  - השעה שהדוא"ל נשלח.
  - דוא"ל מכיל את המשפט "שימי תבורי".
- **הכשרה / Training** - פירושה יצירת או למידת המודל. כלומר, אתה מציג לייבלים לדוגמה של המודל ומאפשר למודל ללמוד בהדרגה את הקשרים בין הפיצ'רים ללייבלים.
- **ניבוי / Prediction** - לאחר שהמודל שלנו מוכן, ניתן להזין אותו באוסף של קלטים (פיצ'רים) אליהם הוא יספק פלט (לייבל) צפוי.

המחשב מקבל קלטים לדוגמה על-ידי ה-"מורה" ומחזיר פלטים רצויים, והמטרה היא ללמוד את כללי המיפוי של הקלטים והפלטים. תהליך האימון נמשך עד שהמודל משיג את רמת הדיוק הרצויה בנתוני האימון.
אנחנו נתעסק קטגוריות הבאות של למידה מכונה:

- **סיווג (Classification)**: הקלטים מחולקים לשני קבוצות או יותר, ועל הלומד לייצר מודל המקצה קלטים בלתי נראים לאחת או יותר מהקבוצות האלה. בדרך כלל מתמודדים עם זה בצורה מפוקחת.
  למשל סינון דואר זבל הוא דוגמה לסיווג, כאשר הקלטים הם הודעות דואר אלקטרוני (או אחרות) והמחלקות הן "דואר ספאם" ו"לא ספאם ". (אנחנו נלמד **Naïve Bayes** ו-**Logistic Regression**).
- **רגרסיה (Regression)**: זו גם בעיית למידה מפוקחת, אך התפוקות רציפות ולא בדידות.
  למשל חיזוי שוק - אתה מאמן את המחשב עם נתוני שוק היסטוריים ומבקש מהמחשב לחזות את המחיר החדש בעתיד. (אנחנו נלמד **Linear Regression**).

הבדלים בין קלסיפיקציה לבין רגרסייה		
	קלסיפיקציה	**רגרסייה**
כרוך בחיזוי של	ערכים בדידים	ערכים רציפים
אופי הנתונים החזויים	לא מסודר	מסודר
שיטת חישוב	על ידי מדידת דיוק	על ידי מדידת טעות ממוצע בשורש ריבועי
אלגוריתמים לדוגמא	Logistic Regression, Naïve Bayes	Linear Regression
שאלות לדוגמה	● האם המייל הזה ספאם או לא? ● האם זה תמונה של כלב או חתול?	● מה הערך של בית בתל אביב? ● מה ההסתברות שהמשתמש ילחץ על המודעה הזאת?
השוואה גרפית		

סיווג בייסיאני נאיבי - Naïve Bayes

מסווגי Naive Bayes הם אוסף של אלגוריתמי סיווג המבוססים על משפט Bayes.
המודל חוזה בעיקר תוצאות טובות על קלסיפיקציה של ספאם בדואר אלקטרוני.
תהליך השימוש במסווג בייסיאני מתחלק לשני שלבים:

1. המסווג מקבל training set - אוסף של דוגמאות/לייבלים והסיווג (classification) שלהן.
כל דוגמא/לייבל מיוצגת על ידי וקטור של ערכים, כאשר כל ערך מציין את הערך שמקבלת הדוגמא/הלייבל בפיצ'ר מסוים.
2. בהינתן דוגמא/לייבל חדש שהסיווג שלו אינו ידוע, המסווג צריך לחזות את הסיווג שלו.

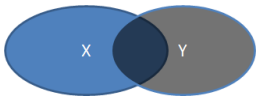
המסווג נקרא "נאיבי" מכיוון שהוא מניח שכל פיצ'ר הוא בלתי תלוי בפיצ'רים אחרים לדוגמה:

- אין תלות בין קומת הבית למחיר הבית.
- אין תלות בין גודל החלון לבין גודל הדלת.

התלות היחידה שהמודל מניח שיש זה בין הנתונים שלנו (הדגימות) לבין לסיווג הסופי, לדוגמה:

- יש תלות בין גודל הבית למחיר הבית.
- יש תלות בין קומת הבית למחיר הבית.

נוסחת בייס:



$$p(Y|X) = \frac{p(Y)p(X|Y)}{p(X)}$$

תרגיל לדוגמה

נרצה לסווג בין הודעות אימייל ספאם לבין הודעות אמיתיות.
ובהינתן שורת טקסט test נרצה לסווג אותו לקבוצה המתאימה.

Real (Non-Spam):
-Will See you later.
-Will you want to meet later?
-I'm waiting for you.

Spam:
-Buy it, pay later! Click me!
-You Won 10000 Dollars! Click here!

Test:
- Are you paying too much? Click now!

הנוסחה:

$$y^* = \underset{k \in \{1,..,K\}}{\operatorname{argmax}} \; p(y = k) \prod_{i=1}^{n} p(x_i \mid y = k)$$

סימונים:

הקבוצות (במקרה שלנו ספאם או אמיתי) מסומנות כך: $y_1, y_2, ..., y_m$. למשל $(y = 0) = spam$.

נסמן את המשפטים שלנו ב- $x_1, x_2, ..., x_m$ כאשר m זה מספר המשפטים/רשומות שלנו באותה קבוצת סיווג y.

למשל $x_1 = Buy\ it,\ pay\ later!\ Click\ me!$ בדוגמה מלמעלה.

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

נספור מופע של כל מילה ב-2 הקבוצות ונציב כל אחד במקומו בטבלה.
לכל מילה נבדוק את ההסתברות שהיא שייכת לקבוצה לאחת הקבוצות.

נוסחת Laplace להסתברות - אם נכפיל את כל ההסתברויות של כל מילים באותה קבוצה נקבל הסתברות 0 כי ככל הנראה נפגוש מילים שלא קיימות באותה הקבוצה. לכן, Laplace מניח שקיים משפט שבו קיימים כל המילים בעולם, כלומר אנחנו מגדירים את כל המילים ב-1 מראש ואז לא נגיע למצב בו נכנס לסיטואציה שנכפיל ב-0 ומכאן:

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

חישוב ה-Laplace:

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

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

נשים לב כי אנחנו מחלקים (המכנה) במספר המשפטים של אותו הקבוצה + 1.
גילינו שעבור הקבוצה של ה-Spam קיבלנו הסתברות גדולה יותר ולכן נסווג את המשפט שלנו (Test) לקבוצת הספאם.

סיווג מרובה

איך נכין את הנתונים שלנו מראש לקראת בקשות מרובות של סיווג?

- סך כל המשפטים שיש במילון שלנו (בכל הקבוצות יחדיו) ונסמן אותו ב- p_{tot}.
- נספור תחילה כמה משפטים יש בכל קבוצה ונסמן p_k.
- נבדוק את כל המילים שמופיעות בכל המשפטים וניצור מהם מילון.
- לכל מילה i במילון ולכל קבוצה k, נגדיר את ההסתברות של כל מילה להופיע בתוך קבוצה ונסמן p_{ki}.

לכן לכל שאילתה חדשה פשוט נחשב:

$$y^* = \max_{k\in\{1,\dots,K\}} \frac{p_k}{p_{tot}} \cdot \prod_{i=1}^{n} \frac{p_{ki}}{p_k}$$

דוגמה

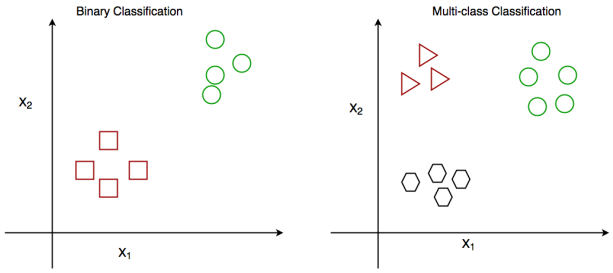
נתבונן ב-2 הגרפים הבאים:

עד עכשיו עבדנו עם 2 מחלקות סיווג כמו הגרף השמאלי (כמו למשל ספאם או לא ספאם).

בסיווג מרובה נשתמש ביותר מ-2 מחלקות סיווג (כמו הגרף הימני). נתבונן בגרף הימני:

ה-p_{tot} הוא מספר כל הצורות בסה"כ בגרף.

ה-p_k הוא מספר הצורות בכל קבוצה, למשל בקבוצה הירוקה $p_{green} = 5$.



Naïve Bayes | **דוגמה בפייתון עם Spark**

```
import numpy as np

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

pk = input_data.map(lambda (message, cls): (cls, 1)).reduceByKey(lambda a,b:
a+b).collectAsMap()

ptot = sum(pk.values())

pki = input_data.flatMap(lambda (message, cls): list(set([(cls,w) for w in
message.split()]))).map(lambda (cls, word): ((cls,word), 1)).reduceByKey(lambda
a,b: a+b).collectAsMap()

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

print(class_probs)

y_star = np.argmax(np.array(class_probs))
print(y_star)
```

מדדים לקלסיפיקציה טובה

Confusion Matrix	Classified as Positive	Classified as Negative
Really Positive	True Positive	False Negative
Really Negative	False Positive	True Negative

Accuracy - נכונות, כמות ההצלחות ביחס לכל החיזויים (כלומר, האם הצלחנו לסווג נכון)

$$\frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Negative + False\ Positive} = \frac{True\ Positive + True\ Negative}{All}$$

Recall - כמה צדקנו בתוך הקטגוריה.

$$\frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Really\ Positive}$$

Precision - דיוק, כמה אנחנו מדוייקים בקטגוריה.

$$\frac{True\ Positive}{True\ Positive + False\ Positive}$$

שאלת מבחן

	Classified as Primary	Classified as Social	Classified as Promotions
Really Primary	10	4	10
Really Social	5	12	1
Really Promotions	10	0	8

א. חשבו את ה-Accuracy

$$Accuracy = \frac{10+12+8}{10+4+10+5+12+1+10+0+8} = \frac{30}{60} = \frac{1}{2}$$

ב. חשבו את ה-Recall של הודעות מסוג Social

$$Recall_{Social} = \frac{12}{5+12+1} = \frac{12}{18} = \frac{2}{3}$$

ג. חשבו את ה-Precision של הודעות מסוג Primary

$$Precision_{Primary} = \frac{10}{10+5+10} = \frac{10}{25} = \frac{2}{5}$$

F-Measure

בעצם סוג של ממוצע. זה המדד לדיוק המבחן, הוא מאחד את העיקרון של Recall ושל Precision.

הנוסחה:

$$F = \frac{2\cdot(Precision \cdot Recall)}{Precision + Recall}$$

תרגיל

נתון מדגם של 100 אנשים, כאשר 60 בריאים ו-40 חולים, עבור 2 קטגוריות נרצה לדעת למי הניקוד הגבוה ביותר?

על פי C_2:

- מהבריאים הוא קבע ש: 10 חולים, 50 בריאים.
- מהחולים הוא קבע ש: 25 חולים, 15 בריאים.

על פי C_1:

- מהבריאים קבע ש: 30 חולים, 30 בריאים.
- מהחולים קבע ש: 35 חולים, 5 בריאים.

נחשב את $F(C_1)$:

$$Precision_1 = \frac{30}{35},\ Recall_1 = \frac{30}{60}$$

$$F_1 = \frac{2\cdot Precision \cdot Recall}{Precision + Recall} = \frac{2\cdot\frac{30}{35}\cdot\frac{30}{60}}{\frac{30}{35}+\frac{30}{60}} = \frac{12}{19}$$

נחשב את $F(C_2)$:

$$Precision_2 = \frac{50}{65},\ Recall_2 = \frac{50}{60}$$

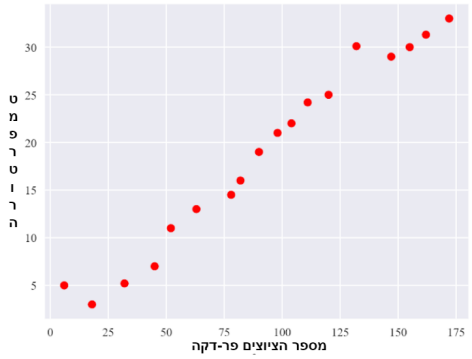
$$F_2 = \frac{2\cdot Precision \cdot Recall}{Precision + Recall} = \frac{2\cdot\frac{50}{65}\cdot\frac{50}{60}}{\frac{50}{65}+\frac{50}{60}} = \frac{4}{5}$$

ולכן הממוצע של קטגוריה C_2 גבוה יותר.

רגרסיה לינארית

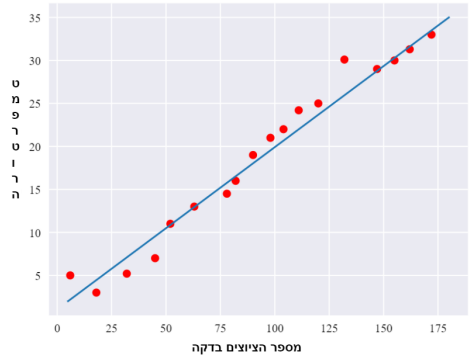
נתחיל מדוגמה (מומלץ לקרוא אותה כדי להבין את ההגדרות הפורמליות לאחר מכן):
זה זמן רב ידוע שצרצרים (זן חרקים) מצייצים בתדירות גבוהה יותר בימים חמים יותר מאשר בימים קרירים יותר.
במשך עשרות שנים מקטלגים מדענים מקצועיים וחובבים נתונים על ציוצים לדקה וטמפרטורה.
כמתנת יום הולדתך, דודתך מעניקה לך את מאגר הצרצרים שלה ומבקשת ממך ללמוד מודל לחיזוי הקשר הזה.
באמצעות נתונים אלה, אתה רוצה לחקור את הקשר הזה.

נתבונן בגרף הבא:



כצפוי, העלילה מראה את הטמפרטורה עולה עם מספר הציוצים

האם הקשר הזה בין ציוצים לטמפרטורה הוא **ליניארי**?
כן, אתה יכול לשרטט קו ישר אחד כמו הבא כדי לערוך יחס זה:



נכון, הקו לא עובר בכל נקודה, אך הקו מראה בבירור את הקשר בין ציוצים לטמפרטורה.
באמצעות המשוואה לקו ישר, תוכל לרשום את הקשר הזה באופן הבא:

$$y = mx + b$$

כאשר:

- y הוא הטמפרטורה - הערך שאנחנו מנסים לנבא.
- mהוא שיפוע הקו.
- xהוא מספר הציוצים בכל דקה - הערך של תכונת (פיצ'ר) הקלט שלנו.
- bהוא יירוט ה-y.

פונקציית הניבוי

לפי מוסכמות בלימוד מכונה, נכתוב את המשוואה למודל באופן קצת שונה, הפונקציה נקראת פונקציית הניבוי:

$$y' = h(x) = w \cdot x + b$$

כאשר:

- $h(x)$- הוא הלייבל החזוי (פלט רצוי).
- b - היא ההטיה "bias" (יירוט ה- y), המכונה לפעמים w_0.
- w- הוא המשקל של הפיצ'ר x. משקל הוא אותו מושג כמו "השיפוע" m במשוואה המסורתית של קו.
- x- הוא הפיצ'ר (קלט ידוע).

כדי לחזות את הטמפרטורה $h(x)$ לערך חדש של ציוצים לדקה x , אז פשוט נמיר את הערך x במודל זה.

למרות שמודל זה משתמש בפיצ'ר אחת בלבד, דגם מתוחכם יותר עשוי להסתמך על מספר פיצ'רים שלכל אחת מהן משקל נפרד. לדוגמה, מודל הנשען על שלוש פיצ'רים עשוי להיראות כך:

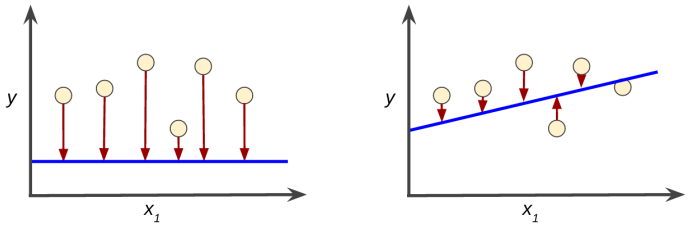
$$y' = w_1x_1 + w_2x_2 + w_3x_3 + b$$

גודל ומזעור הטעות

הכשרת מודל פירושה פשוט ללמוד (לקבוע) ערכים טובים לכל המשקולות וההטיה מדוגמאות הלייבלים.
אלגוריתם למידת מכונה בונה מודל על ידי בחינת דוגמאות רבות וניסיון למצוא מודל שממזער אובדן/טעות;
תהליך זה נקרא **מזעור טעות**.

הטעות הוא העונש לחיזוי רע. כלומר, טעות הוא **מספר** המציין עד כמה חיזוי המודל היה בדוגמה אחת.
אם חיזוי המודל מושלם, ההפסד הוא אפס; אחרת, ההפסד גדול יותר.
מטרת הכשרת המודל היא למצוא סט של משקולות והטיות בעלות אובדן נמוך, בממוצע, בכל הדוגמאות.
לדוגמא, האיור הבא מציג מודל בטעות גבוה משמאל ומודל בטעות נמוכה מימין:

- החץ האדום מייצג את **גודל הטעות**.
- הקו הכחול מייצג את הניבוי שלנו.



שימו לב שהחצים בעלילה השמאלית ארוכים בהרבה מעמיתיהם בעלילה הימנית.
ברור שהקו בעלילה הימנית הוא מודל ניבוי טוב בהרבה מהקו בעלילה השמאלית.

פונקציית הטעות/ההפסד

נשאל האם נוכל ליצור פונקציה מתמטית - **פונקציית הטעות** - שתצבור את הטעויות האישיות בצורה משמעותית.
מודלי הרגרסיה הליניארית שנבחן כאן משתמשים בפונקציית הטעות הנקראת **פונקציית הטעות בריבוע** המחשבת את הטעות הממוצעת בריבוע לכל דוגמא לאורך כל מערך הנתונים.
כדי לחשב אותה נסכום את כל הטעויות בריבוע עבור דוגמאות בודדות ואז נחלק במספר הדוגמאות הכולל שלנו:

$$\frac{1}{N} \cdot \Sigma_{(x,y)\in D} \left(h(x) \ - \ y \right)^2$$

כאשר:

- (x, y)היא דוגמה כך ש:
  - x הוא אוסף של פיצ'רים (למשל ציוצים לדקה, מין , גיל) שהמודל משתמש כדי ליצור ניבויים.
  - yהוא הדוגמה של הלייבל (למשל, הטמפרטורה).
- $h(x)$היא פונקציית הניבוי של המשקולות וההטיות בשילוב עם מכלול הפיצ'רים.
- Nהוא מספר הדוגמאות בקבוצה D.
- Dהוא קבוצת נתונים המכילה דוגמאות לייבל רבות, שהן זוגות (x, y).
  כלומר עבור הנתונים שלנו: $X = \{x_1, x_2,...., x_N\}, \ Y = \{y_1, \ y_2,...., y_N\}$
  הקבוצה Dהיא בעצם: $D = \{(x_1, y_1),..., (x_N, \ y_N)\}$

אנחנו נתייחס מעכשיו לפונקציה שהצגנו בצורה אחרת באופן הבא:

$$\frac{1}{2N} \cdot \Sigma_{i=1}^{N} (h(x_i) \ - \ y_i)^2$$

הסיבה שאנו מחלקים ב-2N היא כדי שלא נחשוב שהטעות גדלה ככל שיש לנו יותר נתונים כי אחרת אם רק נסכום את המרחקים בריבוע אז ככל שיהיה לנו יותר נתונים ככה יהיו יותר טעויות.

עדכוני פרמטר להקטנת הטעות

נחזור לפונקציית הניבוי עבור פיצ'ר אחד: $y' \ = \ h(x) = w \cdot x \ + \ b$.
המערכת בוחנת את הערך של פונקציית הטעות ומייצרת ערכים חדשים עבור w-וb:

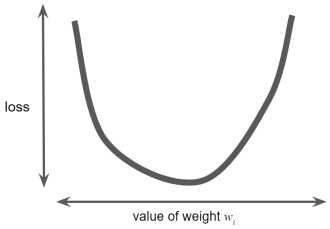
$$J(w, b) \ = \frac{1}{2N} \cdot \Sigma_{i=1}^{N} (h(x_i) \ - \ y_i)^2$$

כלומר נרצה למצוא w, b **שימזערו** לנו את הטעות $J(w, b)$.
לבנתיים, רק נניח שהתיבה המסתורית הזו מציגה ערכים חדשים ואז המערכת מעריכה מחדש את כל הפיצ'רים הללו מול כל אותם לייבלים, ומניבה ערך חדש לפונקציית הטעות , שמניבה ערכי פרמטר חדשים.
הלמידה ממשיכה לחזור עד שהאלגוריתם יגלה את פרמטרי המודל עם האובדן הנמוך ביותר האפשרי.
בדרך כלל אתה חוזר עד שההפסד הכללי מפסיק להשתנות או לפחות משתנה לאט במיוחד. כשזה קורה, אנו אומרים שהמודל **התכנס**.

אלגוריתם Gradient Descent

נרצה למצוא w,b שימזערו לנו את פונקצית הטעות $J(w, b)$.
נעשה זאת באמצעות גרדיאנט - כלומר נגזרת תלת-מימדית.

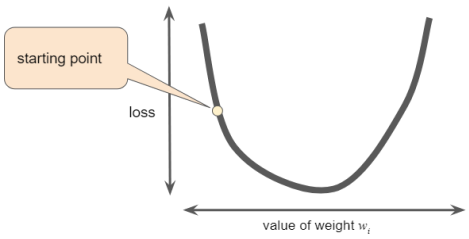
נניח שהיה לנו זמן לחשב את הטעות לכל הערכים האפשריים של w_1. עבור סוג בעיות הרגרסיות שבדקנו, העלילה המתקבלת של הטעות לעומת w_1 תמיד תהיה קמורה. במילים אחרות, העלילה תמיד תהיה בצורת קערה, כזו:



לבעיות קעורות יש מינימום אחד בלבד; כלומר רק מקום אחד בו השיפוע הוא בדיוק 0.
המינימום הוא המקום שבו פונקציית הטעות מתכנסת.

חישוב פונקציית הטעות עבור כל ערך אפשרי של w_1 על פני מערך הנתונים כולו תהיה דרך לא יעילה למצוא את נקודת ההתכנסות. בואו נבחן מנגנון טוב יותר - פופולרי מאוד בלימוד מכונה - הנקרא ירידת שיפוע או Gradient Descent.

השלב הראשון באלגוריתם הוא לבחור ערך התחלתי (נקודת התחלה) עבור w_1. נקודת המוצא לא משנה הרבה; האיור הבא מראה כי בחרנו נקודת התחלה גדולה מ -0:



אלגוריתם ירידת השיפוע מחשב את שיפוע עקומת הטעות בנקודת ההתחלה. כאן באיור מלמעלה, שיפוע הטעות שווה לנגזרת (שיפוע) של העקומה, ואומר לך איזו דרך היא "חמה יותר" או "קרה יותר".

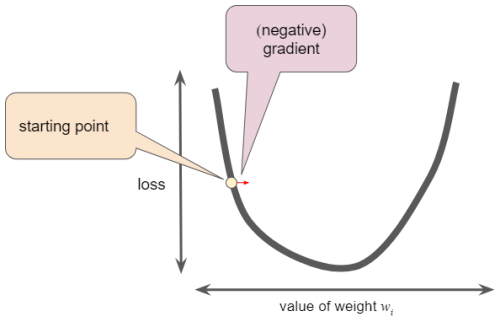
- כשיש משקלים מרובים, השיפוע הוא וקטור של נגזרות חלקיות ביחס למשקולות.

נשים לב ששיפוע (גרדיאנט) הוא **וקטור**, ולכן יש לו שני המאפיינים הבאים:

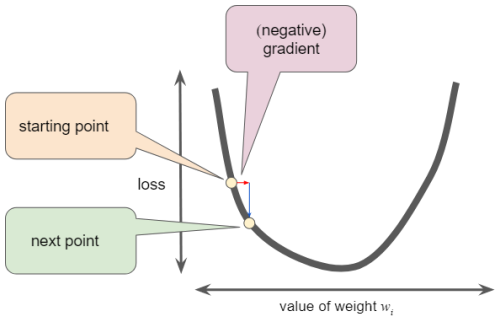
1. כיוון
2. גודל

השיפוע תמיד מצביע לכיוון העלייה התלולה ביותר בקשר לטעות.

אלגוריתם ירידת השיפוע לוקח צעד בכיוון השיפוע השלילי במטרה להפחית את האובדן במהירות האפשרית.

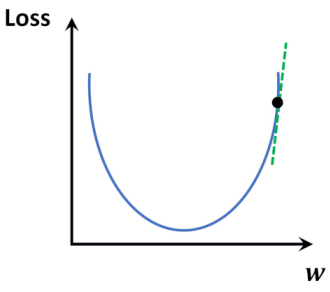


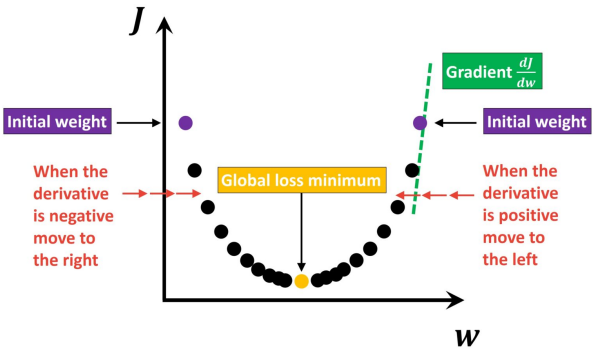
באיור למעלה אפשר לראות כי ירידת שיפוע מסתמכת על שיפועים שליליים.
כדי לקבוע את הנקודה הבאה לאורך עקומת פונקציית הטעות, האלגוריתם מוסיף חלק קטן מגודל השיפוע לנקודת ההתחלה, כפי שמוצג באיור הבא:



צעד הגרדיאנט מעביר אותנו לנקודה הבאה בעקומת הטעות.
ירידת השיפוע חוזרת על התהליך הזה, ומתקרבת למינימום.

<u>המחשות נוספות</u>





לסיכום

נרצה למזער את הפונקציה $J(w,b)$ שהיא פונקציית הטעות, מותר לנו "לשחק" עם המשתנים w,b ולמצוא כאלה שימזערו לנו את הטעות. אם הנגזרת מצביעה על השיפוע, אז אם נעשה צעד בכיוון הפוך לנגזרת אנחנו כנראה נרד קצת בערך (שזה מה שאנחנו רוצים).

עבור פונקציית הטעות שלנו:

$$J(w,b) = \frac{1}{2N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i)^2$$

הנגזרת לפי wתהיה:

$$\frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i) x_i$$

הנגזרת לפי bתהיה:

$$\frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i)$$

ומכאן פונקציית הגרדיאנט יהיה:

$$\nabla(J(w,b)) = \left(\frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i) x_i \ , \ \frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i) \right)$$

האלגוריתם

1. נבחר w,b רנדומלי
2. נעשה בכל פעם צעד α בכיוון הנגדי (זה קצב ההתכנסות).
3. נחזור על זה בצורה הבאה עד להתכנסות:
   a. נעדכן את w להיות:

$$w = w - \alpha \cdot \frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i) x_i$$

   b. נעדכן את b להיות:

$$b = b - \alpha \cdot \frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i)$$

מימוש האלגוריתם ב-Python

Python

```python
import numpy as np
galaxy_data = np.array([[2,70],[3,110],[4,165],[6,390],[7,550]])
w = 0
b = 0
alpha = 0.01

for iteration in range(10000):
    deriv_b = np.mean(1*((w*galaxy_data[:,0]+b)-galaxy_data[:,1]))
    deriv_w = np.mean(galaxy_data[:,0]*((w*galaxy_data[:,0]+b)-galaxy_data[:,1]))
    b -= alpha*deriv_b
    w -= alpha*deriv_w
    if iteration % 200 == 0 :
        print("it:%d,  grad_w:%.3f, grad_b:%.3f, w:%.3f, b:%.3f" %(iteration,
              deriv_w, deriv_b, w, b))

print("Estimated price for Galaxy S5: ", w*5 + b)
```

החישובים מתבססים בדיוק על הנגזרות שהגדרנו:

$$\nabla(J(w,b)) = \left(\frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i) x_i \ , \ \frac{1}{N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i) \right)$$

בפלט יהיה אפשר לראות את ההתכנסות בין כל 200 איטרציות.

<u>מה קורה אם נרצה להוסיף עוד פיצ'רים?</u>

במקרה כזה נרצה להשתמש בפרבולה ולא קו ליניארי.

אומנם אנחנו מתעסקים ברגרסיה לינארית אך אם נרצה לדייק אכן נרצה להתעסק עם פרבולות.

הרעיון: $x_i = (x_{i1}, x_{i2}, \ldots x_{is})$ לכן:

משוואת הניבוי שלנו תיהיה מהצורה הבאה (תלוי במספר ההוספות שלנו):

$$h(x) = (w_1 \cdot x + w_2 \cdot x^2 + b)$$

בפועל, קיבלנו כאן משוואת פרבולה (משוואה ריבועית).

מימוש עבור 2 פיצ'רים | **Spark**

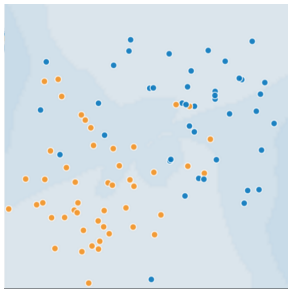
```
import numpy as np
data_x = np.array([[2,4],[3,9],[4,16],[6,36],[7,49]])
data_y = np.array([70,110,165,390,550])
w1 = 0
w2 = 0
b = 0
alpha = 0.001
for iteration in range(1000000):
	deriv_b = np.mean(1*((w1*data_x[:,0]+w2*data_x[:,1]+b)-data_y))
	deriv_w1 = np.dot(((w1*data_x[:,0]+w2*data_x[:,1]+b)-data_y), data_x[:,0]) *
1.0/len(data_y)
	deriv_w2 = np.dot(((w1*data_x[:,0]+w2*data_x[:,1]+b)-data_y), data_x[:,1]) *
1.0/len(data_y)
	b -= alpha * deriv_b
	w1 -= alpha * deriv_w1
	w2 -= alpha * deriv_w2

print("Estimated price for Galaxy S5: ", np.dot(np.array([5,25]),np.array([w1,
w2])) + b)
print("Estimated price for Galaxy S1: ", np.dot(np.array([1,1]),np.array([w1, w2]))
+ b)
```

סכנת התאמת יתר (Overfitting)

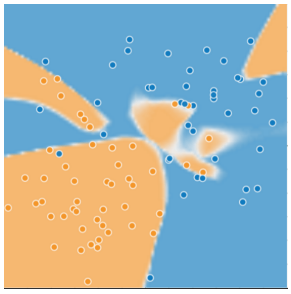
על מנת לפתח אינטואיציה לגבי המושג הזה, נתבונן ב-3 האיורים הבאים.
ננית שכל נקודה באיורים אלה מייצגת את מיקום העץ ביער. לשני הצבעים המשמעויות הבאות:

- הנקודות הכחולות מייצגות עצים חולים.
- הנקודות הכתומות מייצגות עצים בריאים.

נתבונן **באיור 1**:

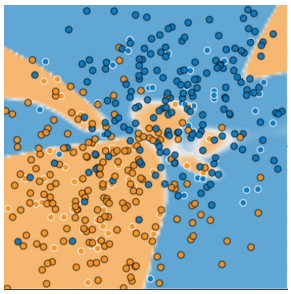


האם אתם יכולים לדמיין מודל טוב לחיזוי עצים חולים או בריאים הבאים?
קחו רגע וציירו בראש קשת המפרידה בין הכחולים והכתומים או מעגלים שמייצגים את הריכוז של 2 הסוגים.
לאחר מכן, הסתכלו **באיור 2**, המראה כיצד מודל למידת מכונה מסוים הפריד בין העצים החולים לבין העצים הבריאים.
שימו לב כי דגם זה הניב הפסד נמוך מאוד (כלומר הוא דיי מדוייק):



במבט ראשון, נראה שהמודל שמוצג באיור 2 עושה עבודה מצוינת בהפרדת העצים הבריאים לבין החולים. או שמא?
הפסד נמוך, אך עדיין מודל גרוע?

איור 3 מראה מה קרה כשהוספנו נתונים חדשים למודל. התברר שהמודל הסתגל בצורה גרועה מאוד לנתונים החדשים.
נשים לב שהמודל טעה בסיווג שגוי של הרבה מהנתונים החדשים.



המודל המוצג באיור 2 ו- 3 מתאים יותר מדי לנתונים שעליו התאמן באופן ייחודי.
מודל שהוא יתר על המידה מקבל הפסד נמוך במהלך האימון, אך עושה עבודה גרועה בחיזוי נתונים חדשים.

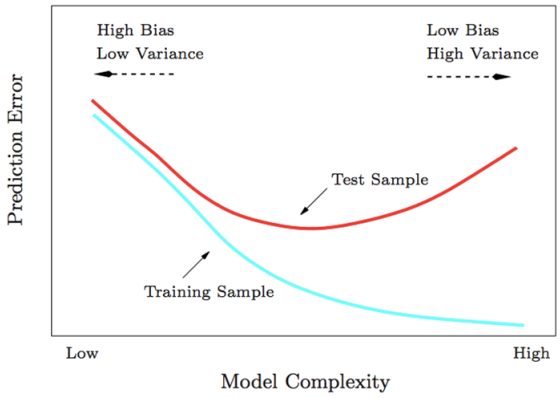
אם מודל מתאים היטב למדגם הנוכחי, כיצד נוכל לסמוך שהוא יחזה טוב על נתונים חדשים?
למרבה הצער, המודל לא יכול לראות את כל האמת; המודל יכול לדגום רק מתוך מערך נתוני אימון.
אם מודל מתאים היטב לדוגמאות הנוכחיות, איך אתה יכול לסמוך על כך שהמודל יחזה גם טוב על דוגמאות שטרם נראו?

אם אתה בונה מודל ממערכת הנתונים שלך, איך היית מקבל את הנתונים שטרם נראו?
ובכן, דרך אחת היא לחלק את מערך הנתונים לשתי קבוצות משנה:

- **Training set** - קבוצת משנה להכשרת מודל.
- **Test set** - קבוצת משנה לבדיקת המודל.

ביצועים טובים במערכת הבדיקות הם אינדיקטור שימושי לביצועים טובים בנתונים החדשים באופן כללי, בהנחה ש:

- ה-Test set גדול מספיק.
- אתה לא מרמה באמצעות אותה Test set שוב ושוב.



<u>נקודות חשובות</u>

- ככל שמוסיפים יותר פיצ'רים צפוי שה **train error** יקטן. הכוונה שאם כל פעם נאמן את הנקודות הוא יתקבע אליהם כי הוא למד את כל הנתונים שלהם וכבר לא יהיה שינוי זה נקרא התאמת יתר או באנגלית over-fitting כי הוא יהיה מקובע רק למה שלימדו אותו, ככל שניתן יותר נתונים על מה שלימדו אותו כבר אז הוא לא יוכל לחוות דברים חדשים הוא יהיה רק מקובע על מה שלמד, אז כשנכניס לו את מה שלימדו אותו, ה-train error יהיה קטן כי הוא יודע בדיוק מה לעשות. לכן ככל שנוסיף יותר פיצ'רים אז ה-train error יקטן כי זה רק מחזק את הדיוק של הניבוי ומקטין את הניבוי.
- קבוצת ה-test היא קבוצה חדשה שהמחשב עוד לא מכיר שגם להם יש error אבל המחשב עוד לא אימן אותם, כלומר גם לקבוצה הזאת יש loss. קבוצת ה-test תיהיה עם נתונים שהם יחסית קרובים וזהים לנתוני הלמידה.
- אנחנו לא רוצים train גבוהה. המודל הוא נכון ככל שה-error נמוך ואם הוא גבוהה אז לקבוצת הtrain הזאת אין משמעות.
- ראינו כי ב-overfitting ה-train error יצנח ל-0 אבל ה-test error נקבל טעות ענקית, ה-test error דווקא יגדל יותר וזה לא הצפי שלנו עבור ה-test.

רגרסיה לוגיסטית

רגרסיה לוגיסטית הוא מודל נוסף לקלסיפיקציה.
במקום לחזות סיווג של בדיוק 0 או 1, רגרסיה לוגיסטית מייצרת הסתברות - ערך בין 0 ל -1.

<u>דוגמה</u> -
זיהוי דואר זבל. אם המודל מסיק להודעת דוא"ל מסוימת ערך של 0.932, זה מרמז על הסתברות של 93.2% שהודעת הדוא"ל היא ספאם. ליתר דיוק, פירוש הדבר שמכלול הדוגמאות שלגביהן המודל מנבא 0.932 יהיה למעשה דואר זבל 93.2% מהזמן ושאר 6.8% לא.

רגרסיה לוגיסטית היא מנגנון יעיל במיוחד לחישוב הסתברויות.
אנחנו יכול להשתמש בהסתברות שהוחזרה כהמרה לקטגוריה בינארית.
מטרת הסיווג בינארי היא לחזות נכון אחת משתי תוויות אפשריות (למשל, "דואר זבל" או "לא דואר זבל").

כיצד מודל רגרסיה לוגיסטית יכול להבטיח תפוקה שתמיד נופלת בין 0 ל -1.
הפונקציה הבאה מייצרת תפוקה עם אותם מאפיינים:

הפונקציה הלוגיסטית

- האוספים:

$$z = w_i x_i + b$$

- פונקציית הניבוי:

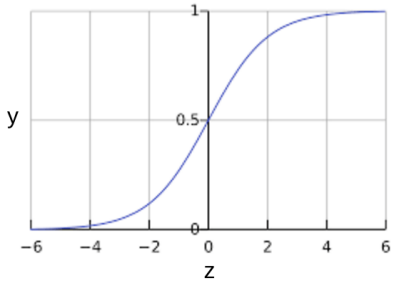
$$h(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(wx+b)}}$$

- פונקציית הטעות:

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

- הגרדיאנט של פונקצית הטעות:

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



האלגוריתם בשימוש Gradient Descent

- נבחר w,b רנדומליים
- נבחר את קצב ההתכנסות α.
- נחזור על הפעולות הבאות עד להתכנסות:
  - נעדכן את w להיות:

$$w = w - \alpha \cdot \frac{1}{m} \cdot \Sigma_{i=0}^{n} x_i(h(x_i) - y_i)$$

  - נעדכן את b להיות:

$$b = b - \alpha \cdot \frac{1}{m} \cdot \Sigma_{i=0}^{n} 1 \cdot (h(x_i) - y_i)$$

דוגמה

נניח שהיה לנו מודל רגרסיה לוגיסטית עם שלושה פיצ'רים שלמדו את ההטיה והמשקולות הבאים:

$$b = 1$$
$$w_1 = 2$$
$$w_2 = -1$$
$$w_3 = 5$$

נניח עוד כי ערכי הפיצ'רים הבאים לדוגמא נתונה הם:

$$x_1 = 0$$
$$x_2 = 10$$
$$x_3 = 2$$

לכן עבור:

$$z = w_1x_1 + w_2x_2 + w_3x_3 + b$$

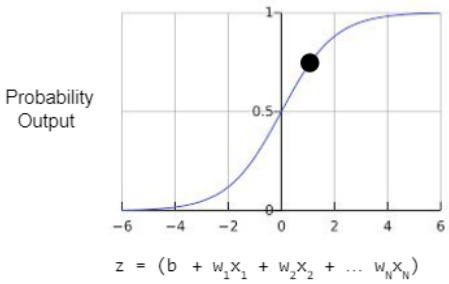
הסיכוי יהיה:

$$z = (2)(0) + (-1)(10) + (2)(5) + 1 = 1$$

כתוצאה מכך, חיזוי הרגרסיה הלוגיסטית לדוגמא המסוימת הזו יהיה 0.731:

$$h(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-1}} = 0.731$$

הפלט של ההסתברות:



ואם נסווג באופן בינארי ניתן להגיד כי אם ההסתברות גדולה מ-0.5 אז הוא שייך לקבוצה 1 כלשהי ולא לקבוצה 0.
נראה את רגרסיה לוגיסטית כסוג של אינדיקטור.

דוגמה

נרצה לדעת אם הבן אדם מועסק או מובטל.
יש לנו מסד נתונים, אנחנו רוצים לדעת לאיזה אנשים צריך להציע עבודה. (תלוי אם מועסקים או לא).
נרצה לבנות מסווג (classifier) בעזרת Logistic Regression עם 3 פיצ'רים: (גיל, מגדר ושנות ניסיון).

הנתונים שבידינו

Employed users:
–Female, 28 years old, 4 years of experience
–Female, 60 years old, 34 years of experience
–Female, 25 years old, 3 year of experience
–Male, 54 years old, 20 years of experience
–Male, 24 years old, 2 years of experience
–Male, 39 years old, 12 years of experience
–Male, 30 years old, 4 years of experience

Unemployed users:
–Female, 36 years old 10 years of experience
–Female, 26 years old 1 year of experience
–Male, 44 years old, 9 years of experience

Python — **נממש עם אותם נתונים שהצגנו למעלה**

```python
import numpy as np
data_x = np.array([[1,28,4],[1,60,34],[1,25,3],[0,54,20],[0,24,2],[0,39,12]
                   ,[0,30,4],[1,36,10],[1,26,1],[0,44,9]])

data_y = np.array([1,1,1,1,1,1,1,0,0,0])

def h(x,w,b):
    return 1 / (1+np.exp(-(np.dot(x,w) + b)))

w = np.array([0.,0,0])
b = 0
alpha = 0.001

for iteration in range(100000):
    deriv_b = np.mean(1*( (h(data_x,w,b)- data_y)))
    deriv_w = np.dot((h(data_x,w,b) - data_y), data_x)*1/len(data_y)
    b -= alpha*deriv_b
    w -= alpha*deriv_w

print("User [1, 49, 8] prob of working: ", h(np.array([[1, 49, 8]]),w,b))
print("User [0, 29, 3] prob of working: ", h(np.array([[0, 29, 3]]),w,b))
print("User [1, 29, 3] prob of working: ", h(np.array([[1, 29, 3]]),w,b))
```

פלט:

```
User [1, 49, 8] prob of working:  [ 0.21107079]
User [0, 29, 3] prob of working:  [ 0.66430518]
User [1, 29, 3] prob of working:  [ 0.43735087]
```

כלומר, כל מה שמתחת לחצי שייך לקבוצה 0 כלומר שהם אנשים שלא יקבלו תעסוקה.
כל מה שמעל לחצי שייך לקבוצה 1 כלומר יקבלו תעסוקה.
בעצם הרעיון כאן מתעסק באינדיקציה.

משמעות התוצאות שמקבלים מהפונקציה הלוגיסטית

- התוצאה שנקבל מהפונקציה הלוגיסטית מתייחסת להסתברות המופע השייך למחלקה $\Pr(Y = 1|X)$.
  כלומר, בהינתן Xמה הסיכוי שה-$Y = 1$.
- רגרסיה לוגיסטית הוא מודל מפלה שרק רוצה לראות את ההבדלים בין 2 קבוצות.
  לא מעניין כמה שייכים לקבוצות אלא רק אינדיקציה כלשהי.

שאלת מבחן של ML (מועד ב' 2018)

נתון דאטה בטבלה גדולה המתארת שתי קבוצות סטודנטים, האחת לומדת מדעי המחשב והשנייה לומדת אדריכלות (ראו מטה דוגמא חלקית מהדאטה). לפי הדאטה מחצית מהסטודנטים הלומדים מדעי המחשב יש אופניים, וכן למחצית הסטונדטים ממדעי המחשב יש מכונית, אותו הדבר בסטודנטים הלומדים ארכיטקטורה. אולם, לכל הסטודנטים הלומדים מדעי המחשב אם יש מכונית אין אופניים ולהפך, אם אין מכונית יש אופניים. אולם לכל הסטודנטים הלומדים ארכיטקטורה, אם יש מכונית יש גם אופניים ואם אין מכונית גם אין אופניים. (ניתן להניח שהתנהגות זו אופיינית לסטודנטים וזה לא רק במקרה).

Name	Has Bicycle	Has own car	Department
Tamar	Yes	Yes	architecture
Danny	Yes	No	CS
Nofar	No	Yes	CS
Zofiya	No	No	architecture

אנו רוצים לבנות מודל שבהינתן סטודנט או סטודנטית יחליט האם הם ממחלקת מדעי המחשב או ארכיטקטורה על סמך בעלות על אופניים ומכונית **בלבד**.

א. נניח ובנינו מודל Naïve Bayes לבעיה, האם הוא יעבוד טוב? נמקו היטב.
ב. נניח ובנינו מודל logistic regression לבעיה, האם הוא יעבוד טוב? נמקו היטב.

פתרון

קודם נפשט את הנתונים (הסיווג), הפיצ'רים/תכונות שלנו בעצם יהיו $X = (Has\ Bicycle,\ Has\ Car)$
על סמך הפיצ'רים נשאל האם $Y = CS$ או האם $Y = Architecture$.

א. מודל Naive Bayes לא יתכנס לתשובה הנכונה כיוון שלפי ההנחה של המודל הזה, אין תלות בין הפיצ'רים אבל קיים לנו תלות חזקה כי אם בהינתן שאני ל-CS ואני יודע שיש לי אופניים זה אומר לי שבטוח אין לי רכב.
ואם אני סטודנט לארכיטקטורה ואני יודע שאין לי רכב זה אומר שאין לי גם אופניים.
כלומר $\Pr(has\ Bicycle \mid CS,\ has\ Car) = \Pr(has\ Bicycle \mid CS)$ וזה לא נכון!

ב. בגלל ש-logistic כן לוקח בחשבון את 2 הנתונים ומייצר משקלים שממזערים עבור הנתונים אז ה-logistic regression כן יעבוד טוב מכיוון שהוא כן לוקח את התלות בין 2 הנתונים ומייצר w ו-b שממזערים לנו את ההפסד אז מן הסתם אנחנו נתכנס לתשובה שהיא יותר טובה מ-50%.
למה 50%? כיוון שההסתברות לאופניים היא ½ ולרכב היא ½ וזה נכון עבור 2 מחלקות הסטודנטים.
כלומר אם:

$$(w_1,\ w_2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = (0.5,\ 0.5) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

באופן חמדני בחרנו את המשקלים לפי הנתון שלנו, כלומר תמיד נקבל תוצאה שלא אומרת לנו כלום על היחס בדומה למקרה של Naive Bayes אבל מכיוון שיש לנו אפשרות לבחור שרירותי משקלים $w_1,\ w_2$ אנחנו נבחר דווקא משקלים שכן "יחשפו" לנו את הטעות (כלומר כמה אנחנו רחוקים מהאמת).

<u>סיכום רגרסיות לינארית ולוגיסטית</u>

- **פונקציית הניבוי**
  - לינארית - $h(x) = w \cdot x + b$
  - לוגיסטית - $h(x) = \frac{1}{1+e^{-(wx+b)}}$
- **מציאת גרדיאנט**

  תקף לשניהן - לגזור לפי w ולפי b ולהציב את הערכים הנתונים.
  - נגזרת לפי w היא: $\frac{1}{N} \cdot \Sigma_{i=1}^{N} (\ h(x_i) - y_i\ )\ x_i$
  - נגזרת לפי b היא: $\frac{1}{N} \cdot \Sigma_{i=1}^{N} (\ h(x_i) - y_i\ )$

  ומכאן פונקציית הגרדיאנט יהיה:

$$\nabla(J(w,b)) = \left(\frac{1}{N} \cdot \Sigma_{i=1}^{N} (\ h(x_i) - y_i\ )\ x_i\ ,\ \frac{1}{N} \cdot \Sigma_{i=1}^{N} (\ h(x_i) - y_i\ ) \right)$$

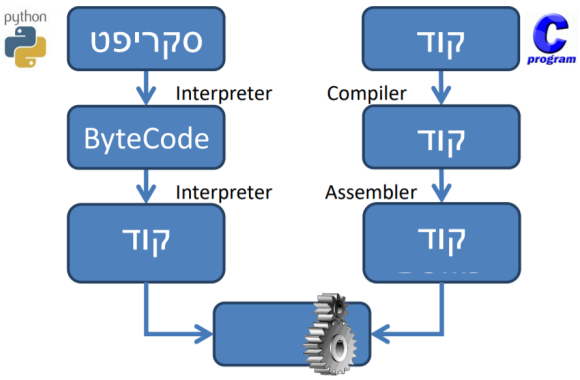
- **מציאת טעות**

  הצבה בפונקציית הטעות עם הערכים הנתונים: $J(w,b) = \frac{1}{2N} \cdot \Sigma_{i=1}^{N} (h(x_i) - y_i)^2$
- **תיקון הטעות**
  - נגזרת לפי w היא: $w = w - \alpha \cdot w$
  - נגזרת לפי b היא: $b = b - \alpha \cdot b$

מבוא ל-Python

פייתון היא שפת תכנות ברמה גבוהה. פייתון היא שפת תכנות מונחית עצמים שיש לה תמיכה עצומה בספריות והופכת מימוש של תוכניות ואלגוריתמים לקלים יותר.

שפת פייתון פותחה **כשפת סקריפטים.** כדי להבין מהי שפת סקריפטים נצטרך להבין קודם כל כיצד עובדת שפה שאינה שפת סקריפטים, לדוגמה שפת C. כל שפת תוכנה צריכה להפוך בדרך כלשהי לשפת מכונה, כדי שהמעבד של המחשב יוכל להריץ אותה. **ההבדל** בין שפת סקריפטים לשפה שאיננה שפת סקריפטים הוא במסלול שעובר הקוד עד שהוא הופך לשפת מכונה. קוד שנכתב בשפת C צריך לעבור שני שלבים לפני שהמעבד של המחשב יכול להריץ אותו: **השלב הראשון** נקרא קומפילציה והוא מבוצע על ידי תוכנה שנקראת קומפיילר. הקומפיילר ממיר את הקוד משפת C לשפת אסמבלי . **השלב השני** מבוצע על ידי תוכנה שנקראת אסמבלר. האסמבלר ממיר את הקוד משפת אסמבלי לשפת מכונה.



שפת סקריפטים (פייתון) לעומת שפת C

על סקריפט פייתון פועלת תוכנה שנקראת **interpreter "פרשן"**.

ה-interpreter עובד בצורה אחרת לגמרי מאשר הקומפיילר והאסמבלר בהם משתמשים כדי לתרגם את שפת C לשפת מכונה. הוא אינו יוצר קובץ אסמבלי וגם אינו יוצר קובץ הרצה. במקום זאת, כל פקודה שכתבנו בשפת פייתון מתורגמת לשפת מכונה רק בזמן הריצה. תוך כדי תהליך הפירוש, נוצר קובץ עם סיומת pyc שמכיל bytecode – הוראות שונות של ה-interpreter – אך כאמור **זה אינו קובץ בשפת מכונה**, כלומר, מעבד לא מסוגל להריץ את הקובץ הזה.

מה אפשר להסיק ממה שלמדנו? קודם כל, ששפת פייתון היא הרבה יותר "סלחנית" לשגיאות מאשר שפות אחרות. לכן, הדעה הרווחת היא שקל יותר ללמוד לכתוב קוד בשפת פייתון. עם זאת , גם לשפת C יש יתרונות על פייתון: **ראשית**, אם נכתוב קוד לא זהיר בשפת פייתון הוא יתרסק תוך כדי ריצה . **אין** מנגנון כמו הקומפיילר של C, שמונע מאיתנו לכתוב קוד שלא עומד בכללי התחביר של השפה ולכן הסיכוי לבעיות בזמן ריצה הוא קטן יותר. התרסקות של קוד תוך כדי ריצה **היא חמורה** לאין שיעור מאשר שגיאת קומפילציה, אותה ניתן לגלות ולדבג לפני ההרצה. **שנית**, העובדה שקוד בשפת C מתורגם לשפת מכונה לא שורה אחר שורה אלא כקובץ אחד, מאפשרת לבצע תהליכי ייעול (אופטימיזציה) של הקוד, כך שהוא עשוי **לרוץ יותר מהר** ולצרוך פחות זיכרון **מאשר** קוד מקביל בשפת פייתון. זה דבר חשוב למי שרוצים להריץ אפליקציות "כבדות", כגון גרפיקה מורכבת או הצפנה.

משתנים וטיפוסים בסיסיים

שמות המשתנים מכילים אותיות מספרים וקו-תחתון.
הם יכולים להתחיל עם תו או עם קו-תחתון אבל לא יכולות להתחיל עם מספר.
למשל אתה יכול לקרוא למשתנה בשם $planet_1$ולא בשם 1_planet.

מחרוזות

בפייתון אפשר להשתמש ב-Strings בצורה הבאה: "hi" או עם 'hi'.
גמישות זו מאפשרת לך להשתמש במרכאות במחרוזות שלך:

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

כאשר נרצה להדפיס מחרוזת נשתמש בפונקציה ***print()*** למשל:

```
name = "planet mars"
print(name)
```

נקבל את הפלט:

```
planet mars
```

הפונקציה ***title()*** תדפיס לנו כותרת רשמית עבור כל מחרוזת:

```
name = "planet mars"
print(name.title())
```

כלומר נקבל את הפלט הבא:

```
Planet Mars
```

במקום $planet\ mars$.

הפונקציות ***upper()*** ו-***lower()*** ידפיסו לנו את המחרוזת באותיות גדולות או קטנות:

```
name = "planet mars"
print(name.upper())
print(name.lower())
```

פלט:

```
PLANET MARS
```

```
planet mars
```

במצבים מסוימים, תרצה להשתמש בערך של משתנה בתוך מחרוזת.
לדוגמה, ייתכן שתרצה שני משתנים מייצגים שם פרטי ושם משפחה בהתאמה, ואז רוצה לשלב ערכים אלה כדי להציג את שמו המלא של מישהו:

```
first_name = "Michael"
last_name = "Jackson"
full_name = f"{first_name} {last_name}"
print(full_name)
```

פלט:

```
Michael Jackson
```

כדי להכניס משתנה לתוך מחרוזת, יש להציב את האות 'f' ממש לפני פתיחת המרכאות.
על כל משנה במחרוזת יש לשים {}.

מספרים

עבור מספרים יש את האריתמטיקה המוכרת: +,-,\,* וכו'...

```
a = 5
b = 5
print(a+b)
print(a-b)
print(a/b)
print(a*b)
```

נקבל את הפלט הבא:

```
10
0
1.0
25
```

בפייתון נכתוב חזקה כך: 2**2 ונקבל את הפלט 4 במידה ונדפיס.

רשימות

בפייתון, הסימון [ ] המציין רשימה ובה אלמנטים המופרדים בפסיק (,).

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(planets)
```

הפלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter']
```

כדי לגשת לתאים במערך נציין כך: $print(planets[0])$ונקבל את הפלט Earth.
כמובן שגם כאן תקף פונקציות המחרוזות כמו למשל $print(planets[0].upper())$נקבל EARTH.
בהתבקש לגשת לערך באינדקס 1-, פייתון תמיד תחזיר את הערך האחרון ברשימה.
כלומר במקרה שלנו עבור: $print(planets[-1])$נקבל את הפלט Jupiter.

הפונקציה ***append()*** תוסיף לנו ערכים חדשים לסוף הרשימה, למשל:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
planets.append('Neptune')
print(planets)
```

נקבל את הפלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter', 'Neptune']
```

אפשר גם להוסיף אלמנט לרשימה בכל מיקום שתרצה בעזרת הפונקציה ***insert()***, למשל:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
planets.insert(0,'Neptune')
print(planets)
planets.insert(3,'Saturn')
print(planets)
```

נקבל את הפלט הבא:

```
['Neptune', 'Earth', 'Mars', 'Venus', 'Jupiter']
['Neptune', 'Earth', 'Mars', 'Saturn', 'Venus', 'Jupiter']
```

אם אתה יודע את המיקום של הפריט שאתה רוצה להסיר ברשימה, אתה יכול להשתמש בהצהרת ***del***.

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
del planets[3]
```

נקבל את הפלט:

```
['Earth', 'Mars', 'Venus']
```

הפונקציה ***pop()*** תמחק ותחזיר את האיבר במקום האחרון ברשימה.

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
popped = planets.pop()
print(planets)
print(f"{popped} has popped from the list")
```

הפלט:

```
['Earth', 'Mars', 'Venus']
Jupiter has popped from the list
```

בנוסף אפשר להשתמש ב-pop לכל מקום ברשימה, למשל:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
popped = planets.pop(1)
print(planets)
print(f"{popped} has popped from the list")
```

הפלט:

```
['Earth', 'Venus', 'Jupiter']
Mars has popped from the list
```

לפעמים אנחנו לא יודעים את האינדקס של הערך אבל אנחנו יודעים מהו הערך, לכן נשתמש במקרה זה בפונקציה ***remove()***. למשל:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
planets.remove('Venus')
print(planets)
```

הפלט:

```
['Earth', 'Mars', 'Jupiter']
```

הפונקציה ***sort()*** תמיין לי את הרשימה בסדר אלפבתי :

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
planets.sort()
print(planets)
```

הפלט:

```
['Earth', 'Jupiter', 'Mars', 'Venus']
```

באפשרותינו גם למיין את הרשימה בסדר הפוך בצורה ***reverse=True*** כך:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
planets.sort(reverse=True)
print(planets)
```

הפלט:

```
['Venus', 'Mars', 'Jupiter', 'Earth']
```

הפונקציה ***sorted()*** תציג לך את הרשימה ממוינת אבל לא באמת תשנה אותה, למשל:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(sorted(planets))
print(planets)
```

הפלט:

```
['Earth', 'Jupiter', 'Mars', 'Venus']
['Earth', 'Mars', 'Venus', 'Jupiter']
```

הפונקציה ***reverse()*** תהפוך את הסדר של הרשימה (לא מיון הפוך) באופן הבא:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(planets)

planets.reverse()
print(planets)
```

הפלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter']
['Jupiter', 'Venus', 'Mars', 'Earth']
```

הפונקציה ***len()*** תחזיר לנו את אורך הרשימה שלנו, למשל:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(f"the length of this list is: {len(planets)}")
```

הפלט:

```
the length of this list is: 4
```

פעולות עם רשימות

על כל לולאה צריך לציין 'for', שם יחידון ואחריו 'in' המתייחס לרשימה כלשהי,
בנוסף בפייתון בשונה משפות תכנות אחרות, ה-"בלוק" לא מוגדר כך {} אלא לפי tabs, אפשר לראות בדוגמה הבאה שפונקצית ה-print שלנו נמצא בתוך הבלוק של הלולאה:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']

for planet in planets:
   print(planet)
```

פלט:

```
Earth
Mars
Venus
Jupiter
```

הפונקציה ***range()*** תציג לנו סדרת מספרים למשל:

```
for number in range(1,5):
   print(number)
```

פלט:

```
1
2
3
4
```

ניתן גם ליצור סדרה ולשמור אותה בתור משתנה של סדרה בצורה הבאה:

```
numbers = list(range(1,6))
print(numbers)
```

פלט:

```
[1, 2, 3, 4, 5]
```

נראה דוגמא לרשימות ולולאות:

```
squares = []
for value in range(1, 11):
   square = value ** 2
   squares.append(square)
print(squares)
```

נקבל את הפלט:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

סטטיסטיקות פשוטות על רשימות מספרים עם הפונקציות ***sum(), max(), min()*** :

```
digits = [1,2,3,4,5,6,7,8,9]
print(min(digits))
print(max(digits))
print(sum(digits))
```

נקבל את הפלט:

```
1
9
45
```

ניתן גם להכריז על רשימה חדשה ובמקביל כבר לאחסן אותה בנתונים שנרצה באופן הבא:

```
squares = [value ** 2 for value in range(1, 11)]
print(squares)
```

פלט:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

כדי לעבוד רק עם **חלק** מרשימה נעבוד באופן הבא:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(planets)
print(planets[1:3])
```

פלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter']
['Mars', 'Venus']
```

נשים לב שאינדקס 3 לא כלול כאן, כלומר האינדקס האחרון לא שייך לחיתוך.

ניתן גם לכתוב בצורה הזאת כלומר מאינדקס 0 עד 2 (לא כולל):

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(planets)
print(planets[:2])
```

פלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter']
['Earth', 'Mars']
```

ובצורה הבאה נעבוד עם רשימה חלקית כך שאנו משתמשים מהערך האחרון ברשימה ועד לאינדקס שהגדרנו (כולל האינדקס הזה):

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
print(planets)
print(planets[2:])
```

פלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter']
['Venus', 'Jupiter']
```

כדי לבצע **העתקה עמוקה** של רשימות נציין את הסימון [:] עבור רשימה חדשה באופן הבא:

```
planets = ['Earth', 'Mars', 'Venus', 'Jupiter']
otherPlanets = planets[:]
print(planets)
print(otherPlanets)

planets.append('Neptune')
otherPlanets.append('Uranus')
print(planets)
print(otherPlanets)
```

פלט:

```
['Earth', 'Mars', 'Venus', 'Jupiter']
['Earth', 'Mars', 'Venus', 'Jupiter']
['Earth', 'Mars', 'Venus', 'Jupiter', 'Neptune']
['Earth', 'Mars', 'Venus', 'Jupiter', 'Uranus']
```

Tuples/טאפלים

בשונה מרשימה, את הטאפל אנחנו נגדיר עם סוגריים עגולים ().
היכולת לשנות רשימות חשובה במיוחד כאשר אתה עובד עם רשימת משתמשים באתר או רשימת דמויות במשחק. עם זאת, לפעמים תרצה ליצור רשימה של פריטים שלא יכולים להשתנות.
Tuples מאפשרים לך לעשות בדיוק את זה. פייתון מתייחס לערכים שאינם יכולים להשתנות כ-immutable, ורשימה שאינה ניתנת לשינוי נקראת ***Tuple***.

לדוגמא, אם יש לנו מלבן שתמיד צריך להיות בגודל מסוים, אנו יכולים להבטיח שגודלו לא ישתנה על ידי הכנסת המידות לטאפל:

```
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

פלט:

```
200
50
```

בואו נראה מה יקרה אם נשנה ערך בטאפל:

```
dimensions = (200, 50)
dimensions[0] = 250
```

פלט שגיאה:

```
TypeError: 'tuple' object does not support item assignment
```

כמובן שנוכל "לדרוס" טאפל עם טאפל חדש, כלומר:

```
stars = ('Earch','Mars')
print("Original stars")
for star in stars:
   print(star)

stars = ('Venus','Jupiter')
print("New stars")
for star in stars:
   print(star)
```

פלט:

```
Original stars
Earch
Mars
New stars
Venus
Jupiter
```

תנאים ואופרטורים

השוואה	
a==b	a <= b
a != b	a > b
a < b	a >= b

פעולות לוגיות		
$x < 5 \; and \; x < 10$	מחזיר True אם התנאי נכון	and
$x < 5 \; or \; x < 4$	מחזיר True אם אחד התנאים נכון	or
$not(x < 5 \; and \; x < 10)$	מחזיר את שלילת הפסוק	not

פעולת זהות		
$x \; is \; y$	מחזיר True אם המשתנים הם מאותו סוג אובייקט	is
$x \; is \; not \; y$	מחזיר True שני המשתנים לא מאותו סוג אובייקט	is not

פעולת הכלה		
$x \; in \; y$	מחזיר True אם ערך ספציפי נמצא בסדרת אובייקט כלשהו	in
$x \; not \; in \; y$	מחזיר True אם ערך ספציפי לא נמצא בסדרת אובייקט	not in

למשל:

```
companies = ['nasa', 'space x']
for company in companies:
   if company == 'nasa':
       print(company.upper())
   else:
       print(company.title())
```

פלט:

```
NASA
Space X
```

בפייתון התנאי else-if מיוצג בצורה ***elif***, למשל:

```
age = 25
if age < 4 :
   print("what a baby!")
elif age < 18 :
   print("you are still young!")
else:
   print("watch your hair!")
```

פלט:

```
watch your hair!
```

כדי לבדוק אם רשימה ריקה (בדומה לפונקציה המוכרת ***isEmpty*** בשפות אחרות), ניתן פשוט לכתוב את שם משתנה של הרשימה בתנאי ה-if, למשל:

```
companies = []

if companies:
   for company in companies:
       if company == 'nasa':
           print(company.upper())
       else:
           print(company.title())
else:
   print("the list is empty!")
```

פלט:

```
the list is empty!
```

Dictionaries/מילונים

מילון נגדיר עם {} כאשר כל זוג מופרד ב- : (נקודותיים) משמאל זה המפתח ומימין זה הערך.
נעבוד על משחק המציג כוכבי לכת, לכל כוכב צבע ומספר ירחים.
נתבונן במילון הפשוט הבא:

```
earth = {'color': 'blue', 'moons': 1}
print(earth['color'])
print(earth['moons'])
```

פלט:

```
blue
1
```

ניתן גם להוסיף למילון קיים בצורה הבאה:

```
earth = {'color': 'blue', 'moons': 1}
print(earth)
earth['number of people'] = '7 Billion'
earth['number of days'] = 365
print(earth)
```

פלט:

```
{color': 'blue', 'moons': 1'}
{color': 'blue', 'moons': 1, 'number of people': '7 Billion', 'number of days': 365'}
```

ניתן גם לאתחל מילון ריק ואחר כך להוסיף אליו זוגות מתי שנרצה וכמה שנרצה:

```
earth = {}
earth['number of people'] = '7 Billion'
earth['number of days'] = 365
print(earth)
```

פלט:

```
{'number of people': '7 Billion', 'number of days': 365}
```

אם נרצה למחוק זוג מהמילון שלנו נשתמש ***ב-del*** למשל:

```
earth = {'color': 'blue', 'moons': 1}
print(earth)
del earth['moons']
print(earth)
```

פלט:

```
{color': 'blue', 'moons': 1'}
{'color': 'blue'}
```

שימוש ב-keys בסוגריים מרובעים כדי לקבל את הערך שאתה מעוניין במילון עלול לגרום לבעיה אפשרית אחת: אם המפתח שאתה מבקש אינו קיים, תקבל שגיאה.
במילונים, באופן ספציפי, תוכלו להשתמש בשיטת ***get()*** כדי להגדיר ערך ברירת מחדל שיוחזר אם המפתח המבוקש לא קיים.

שיטת ***get()*** דורשת מפתח בארגומנט הראשון. בארגומנט האופציונלי השני, אתה יכול להעביר את הערך שיוחזר אם המפתח לא קיים:

```
earth = {'color': 'blue', 'moons': 1}
points = earth.get('points','No points value assigned.')
print(points)
```

פלט:

```
No points value assigned.
```

מעבר שלם על מילון באמצעות לולאות for :

```
earth = {
   'color': 'blue',
   'moons': 1,
   'population': '7 billion'
}

for key, value in earth.items():
   print(f"\nKey: {key}")
   print(f"Value: {value}")
```

פלט:

```
Key: color
Value: blue

Key: moons
 Value: 1

Key: population
Value: 7 billion
```

הדפסת כל המפתחות במילון:

```
earth = {
   'color': 'blue',
   'moons': 1,
   'population': '7 billion'
}
for key in earth.keys():
   print(f"Key: {key}")
```

פלט:

```
Key: color
Key: moons
Key: population
```

כדי לעבור על כל הערכים במילון נבצע אותו מימוש רק שהפעם נכתוב (earth.values.
כשנרצה לממש מילון בתוך מילון נוסף, נפעל בצורה הבאה:

```
planets = {
   'earth': {'color': 'blue', 'moons': 1, 'population': '7 billion'},
   'mars': {'color': 'red', 'moons': 2, 'population': '0'}
}

for stars, star_info in planets.items():
   print(f"planet: {stars.title()}")
   color = star_info['color']
   moons = star_info['moons']
   population = star_info['population']
   print(f"color: {color}, moons: {moons}, population: {population}\n")
```

פלט:

```
planet: Earth
color: blue, moons: 1, population: 7 billion

planet: Mars
color: red, moons: 2, population: 0
```

חלאס.