סיכום שיעור 3 25.12.22 (מתן הלל)

המשך פונקציות השורה בעמודת הSELECT

CASE - משמש אותנו כ 'אם', זה עושה את הפונקציה של IF, אם משהו עונה על תנאי מסויים אז תעשה ככה וככה, עם CASE מגיעים עוד שני פקודות שמגדירות אותו, WHEN (מתי ש) - ואז נותנים תנאי מסויים כמו למשל שערך בטווח מסויים, לאחר מכן THEN - אם הערך עונה על תנאי מה אנו רוצים שיתרחש. ELSE מגדיר את הערכים שאינם עונים על התנאי שלנו.

התבנית של בפונקציה:

Case…… When….. Then….. END.

**פונקיצה המאפשרת יצירת עמודה זמנית בתצוגה כלומר התנאי לא ישנה את המידע בDB אלא רק לתצוגה. כלומר נקבל חלקי מידע בהתאם לתנאי שלנו אך אנינו מאבדים את המידע.

Case - מציין את תחילת הפונקציה

When - מגדיר את התנאים, ניתן להגדיר מספר תנאים, כלומר מספר WHEN בנוסף כל WHEN יכול להכיל בתוכו תתי תנאים בעזרת הAND ו OR.

Then - מה הערך שחוזר עבור כל תנאי(עבור כל WHEN), יכול להיות ערך מסויים ויכול להיות גם עמודה. כלומר מה יקרא אם אכן הערך עונה על התנאי.

Else -לא חייבים להשתמש, מגדיר מה יקרה אם שאר התנאים לא יתרחשו, הוא כמו THEN מוציא לפועל איזה שהוא ערך מסויים שנקבע מראש, והטריגר שלו הוא כאשר הערך לא עונה על אף אחד מהתנאים.

End - מציין את סיום הפונקציה.

** כאשר מבצעים פונקציה מסויימת ואנחנו רוצים לקבל את כל שאר העמודות כלומר *, נבצע, **שם_הטבלה.*** לדוגמה **users.***

דוגמה:

```
SELECT TotalChildren
CASE
WHEN TotalChildern BETWEEN 0 AND 2 THEN ‘FEW’
WHEN TotalChildern BETWEEN 3 AND 4 THEN ‘AVRAGE’
ELSE ‘THE MORE THE MERRIER’
END AS SIZE
FROM users
```

פלט:

הבא לי מידע מטבלת USERS והצג לי לפי התנאי הבא: אם המספר ילדים הוא בין 0 ל 2 תציג אותו כ"מעט", אם מספר הילדים בין 3 ל 4 תרשום "ממוצע", עבור כל ערך אחר תרשום "כמה שיותר יותר טוב", ותכנה את העמודה הזו "גודל".

דוגמה:

```
SELECT user.* ,
CASE
WHEN userid > 100 THEN 'A'
WHEN userid BETWEEN 75 AND 100 THEN 'B'
WHEN userid BETWEEN 50 AND 74 THEN 'C'
WHEN userid BETWEEN 25 AND 49 THEN 'D'
WHEN userid BETWEEN 0 AND 24 THEN 'E'
END AS group
FROM users
```

פלט:

הבא מטבלת USRES את כל הרשומות והחל עליהם את התנאי הבא: אם הUSERID גדול מ 100 תרשמו A, אם בין 100 ל 75 = B, אם בין 50 ל 75 =C… וקרא לעמודה GROUP.

פונקציותת אגרגטיביות

להבדיל מפונקציות שורה שמבצעות את הפעולה על כל שורה, פונקציות אגרגטיביות מבצעות את פעולתן במאונך כלומר על כל הנתונים בעמודה(COULOM) , משמש אותנו על מנת לקבל מידע ייחודי לעמודות ספציפיות.

הדומה בין שני סוגי הפונקציות הוא שהן שניהן תצוגתיות ואינן יוצרות שינויי בDB, וכן הם יוצרות עמודה שונה של פלט הפונקציה.

** פונקציות אלו נכללות גם הן בפוקציות הSELECT.

SUM - סוכמת טת כלל הרשומות של עמודה מסויימת, כלומר מבצעת במאונך פעולת חיבור עבור כל הערכים בעמודה שבחרנו.

לדוגמה: SUM של כל הציונים בעמודת GRADE

SUM = GARDE 1 + GRADE 2+ GRADE 3 +...+ GRADE N

דוגמה:

```
SELECT SUM(price) products AS price_sum
FROM products
```

פלט:

תוצג לנו רשומה אחת שתכלול את הסכום של כל המחירים של כלל המוצרים.

נוכל גם להשתמש בתנאי הWHERE בשילוב אם הפונקציות הנ"ל כל מנת ליצור סינון מסויים למידע אותו נרצה לקבל, לדגומה את סכום המחירים רק ביפן, או סכום מחירת הכרטיסים בסינימה סיטי. **במקרה זה הסכום (וגם כל שאר הפונקיצות האגרגטיביות) יבוצע רק על הרשומות שחוזרות מהתנאי שנתנו.

דוגמה:

```
SELECT SUM(price) AS price_sum
FROM products
WHERE customer_country = 'Japan'
```

פלט:

נקבל את הסכום של המחירים של כל המוצרים שנמכרים ביפן.

-AVG מחשב ממוצע על עמודה מוסיימת.

AVG = (GRADE1+GRADE2+GRADE3+...+GRADE N) / N

דוגמה:

```
SELECT AVG(price) AS price_avg
FROM products
WHERE productname NOT LIKE '%GO%'
```

פלט:

הצג לי את ממוצע המחירים של כל המוצרים שאין להם את האותיות "GO" בשם המוצר.

MAX/MIN- מפונקציה המחזירה את הערך המקסימלי או המינימלי של העמודה, ניתן גם להשתמש בעמודות מסוג תאריך.(Max(field) / Min(field

-MAX הערך הגבוהה ביותר בעמודה, או התאריך המוקדם ביותר כלומר הכי הרבה שנים.

-MIN הערך הנמוך ביותר או התאריך הכי מאוחר.

**ברגע שעושים הדרגציה נקבל רק את השורה הבודדת של התוצאה.

דוגמה

```
SELECT  MAX(price)  AS max_price ,
        MIN(price)  AS min_price
FROM products
```

פלט:

הצג לי מטבלת המוצרים את המחיר הגבוהה ביותר והמחיר הנמוך ביותר.

דוגמה

```
SELECT
 MAX(rental_date )  AS max_rental_date ,
MIN( rental_date ) AS min_rental_date
FROM products
```

פלט:

בפונקצית הMAX אנחנו מבקשים את התאריך הכי ישן של ההשכרה, וMIN את האחרון שהשכיר.

** גם כאן ניתן להתשמש בWHERE על מנת לספצף את התוצאות שאנו רוצים לדוגמה לתת לנו את המחיר הכי גבוה של המשחק מקטגוריית משחקי קופסא, או האדם המבוגר ביותר בארץ יפן.

דוגמה:

```
SELECT
MAX(price) AS max_price
MIN(price) AS min_price
FROM product
WHERE is_return = 1
```

פלט:

מה המוצר הזול ביותר והיקר ביותר שקיבל את הערך 1 בעמודת ה is_return.

COUNT- לספור כמה שורות , לדוגמה כמה מכרנו מתחילת החודש, כמה אנשים קנו את הספר וכו... COUNT סופר את כל השורות, אם יש לנו ערכי NULL כלומר אין ערך, אז הCOUNT ידלג עליו ויהיה פער הבין הכמות הכללית של השורות למספר השורות שCOUNT ספר, הפער יהיה בגלל הNULL.

** ניתן להגדיר לפונקציה לספור כמות רשומות כללית על ידי כתיבת *.

ניתן להגידר לפונקציה לספור כמות רשומות לעמודה ספציפית על ידי כתיבת שם העמודה.

**לשים לב שאנחנו מכנים את העמודה עם AS .

דוגמה:

```
SELECT COUNT(*)
FROM products
```

פלט:

ספור עבורי כמה רשומות יש לי בכלל הטבלה.

דוגמה:

```
SELECT COUNT(*)
FROM products
WHERE produtname LIKE '%GO%'
```

פלט:

ספור לי את הכל המוצרים שיש להם "GO" בשם , מטבלת פרודקט.

**קאגל אתר עם הרבה דאטה סטים להורדה. Kaggle.com

GROUP BY- אם אנחנו רוצים לעשות קבוצות מסויימות לפי דברים מסויימים, לדוגמה לקבץ לפי מדינות , לפני תאריכים מסוימיים, אזורים, וכך ניתן להשיג מידע על קבוצות.

זוהי פקודה שתגרום לקיבוץ התוצאות של הפונקציה האגרגטיבית עבור קבוצות מוגדרות, למשל סכימת מחירי המוצרים עבור כל מדינה.

קיבוץ של דאטה, אם אנחנו רוצים שהוא יסכם לנו ברמת קבוצות עלינו להוסיף את הGROUP BY ובסוגריים את שם העמודה, שאנחנו רוצים שיתקבצו הסווגים שלה.

****חשוב לשים לב-** כאשר נשתמש בפקודה זו, השדות היחידים שיכולים להופיע בSELECT מלבד הפונקציות האגרגטיביות הם **השדות שלפיהם קיבצנו**. לא ניתן לשים * או קטגוריה שלא נכללת בפונקציה או בקיבוץ.

**ניתן לבצע שני רמות ויותר של GROUP BY, מוסיפים עוד עמודה, ואז זה יתן סכום או פעולה כלשהי אחרת של כל אחת מהקבוצות שנשים, החיתוך בין כל העמודות. בצורה זאת ניתן להשוות ברמות גבוהות יותר וברזולוציות משתנות של הדאטה. לדוגמה לחלק את המידע לסוגי מדינות וסוגי

ספורט, ואז נקבל את הנתונים על ישראל כדורסל, ישראל כדורגל, ישראל טניס, גרמניה כדורסל, גרמניה כדורגל, גרמניה טנים, שוויץ כדורסל וכ... על כל המדינות ועל כל סוגי המשחקים.

דוגמה:

```
SELECT customer_country,
        SUM(price) AS sum_price
FROM products
GROUP BY customer_contry
```

פלט:

נקבל את הסכומים של סכומי המוצרים עבור כל אחת ממדינות הלקוחות.

```
SELECT category_name ,
            Customer_country
            AVG(price) AS avg_price,
            MAX(rental_date) AS last_rental_date
FROM products
GROUP BY customer_country,
            category_name
```

פלט:

תקבץ לי לפי שני משתנים אחד לפי ארץ מוצא ולפי שם הקטגוריה, על כל אחד המשילוביים הייחודיים תן לי את ממוצע מחירי המוצרים וגם את התאריך הישם ביותר בו השכירו.

**יש לשים לב שמיקומו של הGROUP הוא מתחת לFROM אך מתרחש לפני הSELECT.

-Having מאפר לנו לבצע התניה על התוצאות של הפונקציה הדרגטיבית, היא דומה לWHERE בכך שיוצרת התניה שמסנן את המידע , אך השונה זה שהיא מתרחשת לאחר ה GROUP BY והיא לא משנה את הנתונים עוד לפני הוקבצו יחד.למשל אני רוצה לקבל בממוצע את כל מי שקיבל מעל ציון מסיים.

```
SELECT
Customer_contry, Catgory_name,
AVG(PRICE)
FROM products
GROUP BY customer_contry
        ,category name
HAVING avg(price) > 8.99
```

פלט

נקבץ תחילה לפי מדינת הלקוח והשם של הקטגוריה, ואז בכל אחד מאלה נציג רק את הנתונים שהמחיר שלהם גבוהה מ 8.99.

**משתמשם בHAVING במקום WHERE כיוון שהHAIVNG קורה לאחר ההדרגציה ולכן הפעולה נעישת במלואה ולאחר מכן יש חיפוש של המידע המבוקש, לעומת זאת הWHERE קורה לפני הGROUP BY ולכן ישפיע עליו וישנה את התוצאות שנקבל.

דוגמה

```
SELECT category_name ,
        AVG (price)  AS avg_price
FROM products
WHERE customer_country = 'Greece'
GROUP BY category_name
HAVING AVG(price) > 4
```

פלט

הצג לי את כל קטגוריות המוצרים ביוון שהמחיר שלהם הוא גבוהה מ 4.

** בשביל לנטרל שורות נעשה - - לפני השורה

** כאשר אנחנו עושים WHERE נשים 1=1 כעל מנת שיהיה נוח יותר קרואה ואז שמים AND.

OREDER BY - סידור , מהנמוך לגבוהה או מהגבוהה לנמוך, רושמים ORDER BY לאחר מכן העמודה או הפונקיצה ואז **DESC** אם אנחנו רוצים בסדר יורד (שהגבוהה ביותר בערך בו בחרנו יהיה עליון), אם אנחנו רוצים את הסדר ההפוך נרשום בסוף **ASC**, הראשי תיבות DESENDING ו - ASENDING.

קוד הORDER BY מתרחש מאחורי הקלעים , הוא מתבצע אחרון גם לאחר הSELECT, זה כאילו מתרחש הכל ואז מגיעים לשורת הORDER BY, עקב כך לא יהיה לנו בעיה להשתמש בשם חדש שהגדרנו בעמודת הSELECT.

אפשר למיין לפי A-Z א-ת לפי גודל מספרים, תאריכים כדי להשיג כרונוגלוגית. ניתן למיין לפי יותר משדה אחד, ואז כל עמודה תתמיין לנו בצורה לא תלויה לפי הפקודה שנתנו לה.

**פונקציה זו נכתבת אחרונה, לאחר הHAVING

** כאשר אנחנו עושים ASC ב A-Z הראשון יהיה A.

** הסדר הנכון לעשות כאשר מתעסקים עם DATA מאד גדול הוא להתחיל עם WHERE כדי שהנתונים יהיו מצומצמים מראש. כיוון שזה יכול לתקוע את הDB, ולתקוע את האופרציה.

```
SELCET category_name ,
MAX(price) AS max_price ,
COUNT(*) AS rentals_amount
FROM products
WHERE customer_country= 'Greece'
GROUP BYcategory_name
HAVING MAX(price) > 6
ORDER BY max_price DESC
```

פלט:

הצג לי בכל קטגוריות המוצרים ביוון את המחיר הגבוה ביותר פר מוצר, וספור לי כמה השכירו כל אחדם מהם, הצג לי רק נתונים שהמחיר המקסימום הוא מעל 6, **וסדר לי** את הנתונים בצורה שהמקסימום הגבוהה ביותר יהיה ברשומה הראשונה.

דוגמה:

```
SELECT
studentname ,
lastname
FROM students
ORDER BY studentname ASC  ,
                lastname DESC
```

פלט:

התוצאות יסתדרו קודם כל לפי השם הפרטי , ולאחר מכן רשומות בעלות שם פרטי זהה יסודרו כעל פי שם משפחה בסדר יותר.

**בדוגמה ביצענו סידור כפול , מה שרושמים ראשון יהיה הסידור הראשי והסידורים שאחריו יהיו בתוכו, וכך בצורה הרכרכית ניתן יותר משני סידורים בו זמנית.

LIMIT- מגביל את כמות השורות שאחנחנו רוצים לקבל, יקרה לאחר הORDER BY . תומך ברוב התוכנות SQL, מחליף את פקודת הTOP.

תשובות לשיעורי בית

```
--Q1
SELECT
AVG(`Grade`) AS Grade_Average
FROM `Grades`

--Q2
SELECT
MAX(`Grade`) AS TOP_ONE
FROM `Grades`

--Q3
SELECT
COUNT(`Grade`) AS Number_of_grades
FROM `Grades`

--Q4
SELECT
COUNT(*) AS Number_of_Students
FROM `Students`
```

```
--Q5
SELECT COUNT(*)
FROM `Courses`

--Q6
SELECT
MAX(ID) AS Max_ID,
MIN(ID) AS Min_ID
FROM `teachers`

--Q7
SELECT
MIN(`Date_of_birth`) AS Youngest,
MAX(`Date_of_birth`) AS Oldest
FROM `teachers`
WHERE `house_id` = 1

--Q8
SELECT
SUM(`Grade`)
FROM `Grades`
WHERE `Grade_Type_ID` = 3

--Q9
SELECT
```

```
AVG(`Grade`) AS AVG_Grades_By_ID
FROM `Grades`
GROUP BY `Grade_Type_ID`

--Q10
SELECT
COUNT(`Grade_type_name`) AS NUM_Of_Tests
FROM `Grade_Types`
GROUP BY `CourseID`

--Q11
SELECT COUNT(`Course_ID`) AS Number_Of_Courses_Per
FROM `Student_Enrollment`
GROUP BY `Student_ID`

--Q12
SELECT
COUNT(`Course_ID`) AS Number_Of_Courses_Per
FROM `Student_Enrollment`
GROUP BY `Student_ID`
HAVING COUNT(`Course_ID`) >= 3
```

```
--Q13
SELECT
`house_id` ,
COUNT(id) AS Num_of_teacher,
MAX(`Date_of_birth`) AS MAX_Bithday
FROM `teachers`
GROUP BY `house_id`
HAVING COUNT(`Date_of_birth`) < 1931-01-01

--Q14
SELECT
`Grade_Type_ID`,
COUNT(`Grade`)
FROM `Grades`
WHERE `Grade_Type_ID` <> 2
GROUP BY `Grade_Type_ID`
HAVING COUNT(`Grade`) >= 2

--Q15
SELECT *
FROM `Books`
ORDER BY `pubdate` DESC
```

```
--Q16
SELECT
*
FROM `Books`
ORDER BY `price` DESC,
                `ytd_sales` DESC

--Q17
SELECT
`pubdate` ,
COUNT(`title`) AS Books_Per_Date
FROM `Books`
GROUP BY `pubdate`
ORDER BY `pubdate` ASC

--Q18
SELECT
`type` ,
SUM(`ytd_sales`) AS Yearly_Earning
FROM `Books`
GROUP BY `type`
HAVING SUM(`ytd_sales`) > 10000
```