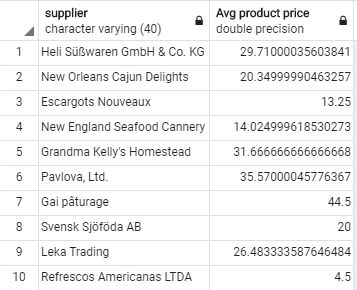
# WINDOW FUNCTIONS

### כמו שלמדנו בשיעור אגרגציות, בעזרת פונקציות טוריות נוכל לקבל אגרגציה של המידע בעמודה ובתוספת פסוקית ה GROUP BY - נוכל לקבל את האגרגציה עבור סדרות של שורות בעלות קשר מסוים (קבוצות) בעמודה.

### בטבלה הבאה אנו יכולים לראות את המחיר הממוצע לכל ספק.

****כפי שניתן לראות, פונקציות טוריות מצמצמות משמעותית את כמות השורות שיחזרו משליפה משום שהן מקבצות את המידע.

מה אם אנחנו רוצים לבצע פונקציות טוריות על המידע בלי באמת לקבץ אותו ולצמצם את כמות השורות המוצגת? למשל, כיצד נוכל לשלוף עבור כל מוצר את מחירו בהשוואה למחיר הממוצע של המוצרים שמגיעים מאותו הספק? כאשר שניהם יוצגו אחד ליד השני?

כפי שראינו, בעזרת פונקציות טוריות אנו יכולים לראות מחיר ממוצע לכל ספק. כדי לראות עבור כל מוצר גם את מחירו וגם את המחיר הממוצע של הספק שלו, אנו יכולים להשתמש בתת שליפה מקושרת:

**עבור כל מוצר הצג את מחירו ואת המחיר הממוצע של מוצרים המגיעים מאותו ספק**

SELECT supp.company\_name AS supplier,

prod.product\_name AS product,

prod.unit\_price AS price,

(

SELECT AVG(avgProd.unit\_price) AS "avg price"

FROM products avgProd

WHERE avgProd.supplier\_id = prod.supplier\_id

)

FROM products prod

INNER JOIN suppliers supp ON

supp.supplier\_id = prod.supplier\_id

ORDER BY supplier;



אבל כבר למדנו שתת שליפה מקושרת היא דרך שננסה להימנע ממנה ככל האפשר בגלל חוסר היעילות שלה. האם יש דרך נוספת לבצע את השליפה הזאת בצורה יותר פשוטה, קצרה ויעילה? ומה אם נרצה בנוסף גם לדעת את ההפרש בין מחירי המוצרים מהגדול לקטן? זה כבר בכלל יהיה ארוך ומסובך.

בדיוק בשביל פעולות כאלו יש לנו את פונקציות החלון – WINDOW FUNCTIONS.

## WINDOW FUNCTIONS

### מהן פונקציות חלון?

בדומה לפונקציה טורית, פונקציית חלון עובדת על סדרת שורות ומפעילה **אגרגציה** עליהן, אך לא תצמצם אותן לכדי שורה אחת.

**הגדרה**: פונקציות חלון הן פונקציות אשר מתבצעות על סדרת שורות הקשורות אחת לשנייה בדרך מסוימת (למשל, תחת אותה העמודה או ערך משותף ביניהן: קטגוריה, שנה, חברה וכו'...) ומחזירות ערך עבור כל שורה בסדרה, בלי לקבץ את סדרת השורות.

### חלון

**הגדרה**: חלון מתאר את סדרת השורות (הקבוצה) עליהן פונקציית החלון עובדת. הפונקציה תחזיר ערכים המתבססים אך ורק על הערכים בחלון.

**אז איך משתמשים בפונקציות חלון?**

ניקח את הדוגמה ממקודם, כדי להבין את מבנה השליפה יותר טוב. אנחנו רוצים לראות עבור על ספק, כל מוצר ואת מחירו, לצד המחיר הממוצע של כל המוצרים המגיעים מאותו ספק.

קודם כל נפשט את השליפה, ונראה רק את הספק, המוצר ואת מחיר המוצר.

SELECT supp.company\_name AS supplier,

prod.product\_name AS product,

prod.unit\_price AS price

FROM products prod

INNER JOIN suppliers supp ON

supp.supplier\_id = prod.supplier\_id

ORDER BY supplier;

תצא לנו הטבלה הבאה:



נדגים עליה את כל החלקים במבנה השליפה.

**מבנה של שליפה עם פונקציית חלון**

SELECT col1,

col2,

window\_function(col) OVER (

0ITION BY partition\_expression [optional]

ORDER BY sort\_expression ASC\DESC [optional]

)

FROM table1

WHERE additional\_condition [optional]

**OVER** – פסוקית ה- OVER מגדירה את חלון הפונקציה, בתוך הסוגריים נוכל להגדיר בין אם החלון הוא כלל המידע בעמודה (קבוצה אחת) או מתחלק (מספר קבוצות) ובאיזה סדר הערכים ימוינו בחלון.

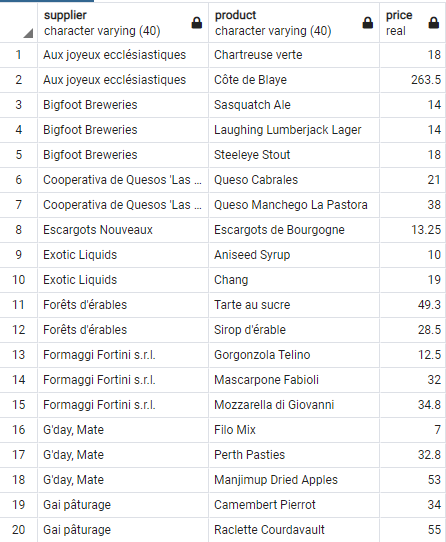
**PARTITION BY** – פסוקית ה- PARTITION BY מאפשרת לחלק את המידע לחלונות (קבוצות) על פי ביטוי/עמודה, כל קבוצה כזו תהווה חלון שעליו תתבצע הפונקציה, פסוקית זו היא אופציונלית, במידה ולא נשתמש בה החלון עליו תתבצע הפונקציה יהיה כל השורות.

כמו שאנחנו רואים, PARTITION BY מחלק לנו את הטבלה לקבוצות, במקרה שלנו על פי הספק.



**ORDER BY** – פסוקית ה- ORDER BY תמיין את הערכים שבחלון לפי הביטוי בפסוקית, כמו פסוקית ה -ORDER BY שאנחנו כבר מכירים.

ORDER BY סידר לנו את הערכים בכל קבוצה, במקרה שלנו על פי מחיר בסדר עולה. כמו שאפשר לראות, הסדר הוא בכל קבוצה, ולא על כל הטבלה.



}

}

}

}

}

AVG

AVG

AVG

AVG

AVG

**WINDOW FUNCTION -** פונקציית החלון שבה נשתמש, בין אם זה פונקציות טוריות שאתם מכירים (SUM, AVG, COUNT וכו') או אחת מן הפונקציות שנכיר בהמשך, שייחודיות לפונקציות החלון.

במקרה שלנו, נרצה לעשות ממוצע לכל אחד מהספקים שלנו.

לאחר שעשינו את כל השלבים האלה, תצא לנו טבלת התוצאה הבאה, שבה העמודה האחרונה מציגה בעבור כל חלון שהוגדר (מוצרי כל ספק) את ממוצע המחירים.



אז בואו ננסה לפתור את השאלה שוב, אך הפעם בעזרת הפונקציות חלון:

**עבור כל מוצר הצג את מחירו ואת המחיר הממוצע של מוצרים המגיעים מאותו ספק**

SELECT supp.company\_name AS supplier,

prod.product\_name AS product,

prod.unit\_price AS price,

AVG(unit\_price) OVER (

PARTITION BY supp.company\_name

) AS "avg price"

FROM suppliers supp

INNER JOIN products prod ON

prod.supplier\_id = supp.supplier\_id;

החישוב של פונקצית החלון מתבצע כחלק מסדר ההרצה הבא:

FROM > JOIN > WHERE > GROUP BY > HAVING > **WINDOW** > LIMIT > SELECT> ORDER BY

לכן חשוב להדגיש כי החישוב קורה על טבלת התוצאה הסופית ומתווסף אליה – הוא אינו משנה את צורת טבלת התוצאה ומספר השורות בה. הוא מתבצע לאחר קיבוץ השורות ועליכם להתחשב בכך בפסוקית ה- GROUP BY.



**שימו לב!**   
חלק מן הפונקציות אינן מקבלות פרמטרים כלל (PARTITION BY ו- ORDER BY) ולכן ניתן להגדיר חלון בעזרת הסוגריים ולהשאיר אותו ריק.

פונקציות חלון תמיד יגיעו בפסוקית ה-SELECT.

**WINDOW**

פסוקית WINDOW היא פסוקית המאפשרת לנו להגדיר חלון ולתת לו שם ובכך לחסוך חזרה על קוד במידה ואנו משתמשים באותו החלון מספר פעמים.

פסוקית WINDOW תמיד תגיע לאחר פסוקית ה- FROM.

SELECT window\_function1() OVER win,

window\_function2() OVER win,

FROM table\_name

WINDOW win AS (PARTITION BY col1 ORDER BY col2)

**Frame**

FRAME מאפשר לנו להגדיר תת סדרה בתוך החלון שעליה הפונקציה תתבצע, כלומר נוכל להגדיר לפונקציה על כמה שורות לפני ואחרי השורה הנוכחית בחלון היא תתבצע.

זאת אומרת, שמעבר לכך שפונקציית חלון מאפשרת לכל שורה לקבל ערך בהתאם לקבוצה שאליה היא משתייכת מבלי להיות מקובצת, השורה יכולה לקבל ערכים שתלויים בשורות שקדמו לה או נמצאות אחריה.

SELECT col1,

col2,

window\_function(col) OVER (

PARTITION BY partition\_expression [optional]

ORDER BY sort\_expression ASC\DESC [optional]

ROWS\RANGE BETWEEN start‑point AND end‑point

)

FROM table1

**ROWS ו- RANGE**

פסוקיות אלו מאפשרות לנו להגדיר את כמות השורות ב- FRAME בעת ביצוע פונקציית חלון בעזרת BETWEEN.  
ל- BETWEEN ניתן 2 פרמטרים, נקודת התחלה וסוף, זאת אומרת כמה שורות לפני השורה הנוכחית וכמה אחריה.

**נקודת התחלה** **–** תוגדר בעזרת מספר/ביטוי ולאחר מכן המילה השמורה PRECEDING.  
**נקודת הסוף –** תוגדר בעזרת מספר/ביטוי לאחר מכן המילה השמורה FOLLOWING.

PARTITION BY partition\_expression [optional]

ORDER BY sort\_expression ASC\DESC [optional]

ROWS\RANGE BETWEEN startpoint PRECEDING AND endpoint FOLLOWING

בנקודות התחלה והסוף ניתן להציב גם את הביטויים הבאים (במקום מספר):  
**UNBOUNDED** – הביטוי UNBOUNDED מאפשר לנו בצירוף אחת מן המילים השומרות FOLLOWING או PRECEDING להגדיר ל- FRAME להתייחס לכל השורות בחלון לפני/אחרי השורה הנוכחית בעת ביצוע הפונקציה.  
**CURRENT ROW** - הביטוי CURRENT ROW מאפשר לנו להגדיר ל- FRAME להתייחס לכל השורות בחלון מהשורה הנוכחית או עד השורה הנוכחית (תלוי בנקודה בו נכתב).

גם ROWS וגם RANGE מאפשר לנו להגדיר טווח שורות בתוך החלון לביצוע הפונקציה או בקצרה FRAME. **אז למה צריך את שניהם?**

**הבדל בין ROWS ל- RANGE**בפשטות, ההבדל בין ROWS ל- RANGE הוא שבעזרת ROWS נגדיר "פיזית" כמה שורות לפני ואחרי השורה הנוכחית אנחנו רוצים ועם RANGE נגדיר "לוגית" כמה שורות היינו רוצים.

נסתכל על הדוגמה הבאה:

SELECT cat.category\_name,

prod.product\_name,

prod.unit\_price,

SUM(prod.unit\_price) OVER (

PARTITION BY cat.category\_id

ORDER BY prod.unit\_price

ROWS BETWEEN UNBOUNDED PRECEDING AND 1 FOLLOWING

) AS "ROWS",

SUM(prod.unit\_price) OVER (

PARTITION BY cat.category\_id

ORDER BY prod.unit\_price

RANGE BETWEEN UNBOUNDED PRECEDING AND 1 FOLLOWING

) AS "RANGE"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id

במקרה הזה הגדרנו FRAME מ- UNBOUNDED PRECEDING ל- 1 FOLLOWING וביצענו פונקציית SUM על מחיר המוצר.

****כמו שניתן לראות בטבלה, אנחנו מקבלים תוצאות שונות עבור ROWS ו- RANGE, הסיבה לכך היא ש- ROWS ו- RANGE מפרשים את הביטוי בצורה שונה.

ניתן לראות דוגמה לכך בטבלה:

נסתכל על השורה השלישית. הערך ב- unit\_price הוא 14 והערך ב- RANGE הוא 55.25.

ערך זהה נמצא גם בשורה הרביעית, למה?



בואו נסתכל על הטווח שהגדרנו.

ניקח את השורה השלישית ונסתכל על הביטוי RANGE BETWEEN UNBOUNDED PRECEDING AND 1 FOLLOWING.

ניתן לראות שבשורה השלישית, הטווח למעשה הוא 4.5 עד 15 מה שמסביר למה unit price של שורה 4 נכנס גם ל- FRAME ואפילו הערך בשורה 5, בניגוד ל- ROWS.

ומה עם RANGE בשורה החמישית? הערך ב- RANGE שווה גם שם 55.25, שוב ניקח את השורה ונסתכל על הביטוי, הטווח המוגדר הוא (16 – 4.5) כך שלמעשה אותן שורות מ - FRAME של שורה 3 ושורה 4 נמצאות ב- FRAME של שורה 5 ולכן הפונקציה מחזירה עבורה את אותו הערך.

ה- DEFAULT FRAME: שימו לב שכל עוד השתמשתם בסעיף ה- ORDER BY, נוצר לכם FRAME דיפולטי שמוגדר לכם מתחילת המידע ועד לשורה הנוכחית, כולל (PRECEDING UNBOUNDED), וכך גם אם לא הגדרתם באמצעות ROWS או RANGE את ה- FRAME, הוא קיים.

## פונקציות

אפשר לחלק את פונקציות החלון ל3 סוגים:

* Aggregate
* Ranking
* Value

**AGGREGATE**

אלו הפונקציות שאנו כבר מכירים, SUM, AVG, COUNT, MIN, MAX.

**עבור כל קטגוריה הצג סכמה הדרגתית של מחירי המוצרים מהקטן לגדול**

SELECT cat.category\_name AS category,

prod.product\_name AS product,

prod.unit\_price AS price,

SUM(unit\_price) OVER (

PARTITION BY cat.category\_name

ORDER BY unit\_price

) AS "summed"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id



**RANKING**

פונקציות אלו משמשות לדירוג השורות לפי סדר המיון של ערכן, כל שורה תקבל ערך אשר מהווה את "דרגתה", השימוש הנפוץ בהן הוא שליפת שורות על בסיס מדרג כלשהו (לדוגמה עשרת ההזמנות הכי כבדות).

הפונקציות הן: **RANK, DENSE\_RANK, ROW\_NUMBER, NTILE, CUME\_DIST.**

1. **RANK()** – מדרגת כל שורה בתוך החלון שלה (1,2,3....), במידה וישנם 2 ערכים זהים הם יקבלו דירוג זהה והפונקציה תדלג על הדרגה הבאה (1,2,2,4)
2. **DENSE\_RANK()** – מדרגת כל שורה בתוך החלון שלה (1,2,3....), במידה וישנם 2 ערכים זהים הם יקבלו דירוג זהה, הפונקציה תמשיך אל הדרגה הבאה (1,2,2,3)
3. **ROW\_NUMBER()** – מדרגת כל שורה בחלון, הפונקציה מתחילה ב1, ערכים זהים ידורגו שונה.

פונקציות אלו מבצעות פעולה מאוד דומה, השימוש בכל אחת הוא בהתאם לצורך בשליפה.

**עבור כל קטגוריה דרג את המוצרים מהיקר לזול**

SELECT cat.category\_name AS category,

prod.product\_name AS product,

RANK()\DENSE\_RANK()\ROW\_NUMBER() OVER (

PARTITION BY cat.category\_name

ORDER BY unit\_price DESC

) AS "rank"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id

**ROW\_NUMBER**



**DENSE\_RANK**



**RANK**

1. **NTILE()** – מאפשרת לחלק את השורות בחלון בצורה שווה (ככל האפשר) למספר המדרגות שצוין (נקרא גם "באקטים"), במידה ומספר ה"באקטים" גדול ממספר השורות, מספר ה"באקטים" יהיה מספר השורות.

**עבור כל קטגוריה הצג כל מוצר ובאיזה מדרג מחירו ביחס לשאר המוצרים ( 1-4)**

SELECT cat.category\_name AS category,

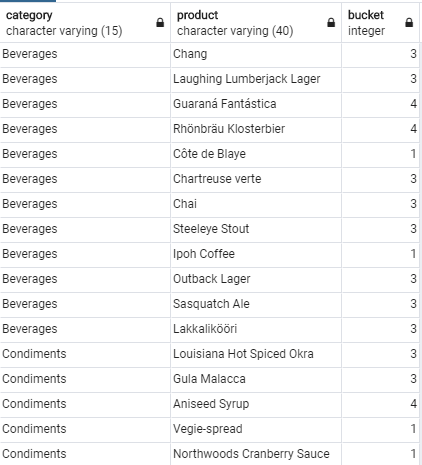
prod.product\_name AS product,

NTILE(4) OVER (ORDER BY prod.unit\_price DESC) AS "bucket"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id

****ORDER BY cat.category\_name

1. **CUME\_DIST () -** תיתן לנו את התפלגות המצטברת של הערכים בחלון, כל ערך יקבל מספר המייצג את אחוז הערכים בחלון שקטנים או שווים לערך שנבדק, למעשה הפונקציה תיתן לנו מיקום יחסי של הערך בחלון ביחס  לערכים השונים.

פונקציה זו מאפשרת לנו לחשב את המיקום היחסי של ערכים בחלון מבחינת פרמטר שאנחנו נקבע.

עבור ערך יחיד היא תחשב את המיקום היחסי שלו בקבוצת ערכים, זאת אומרת, מכמה ערכים הוא גדול/קטן (תלוי בorder by) או שווה מתוך כמות הערכים בקבוצה.

**דוגמה**

**מה היא ההתפלגות המצטברת של מחירי המוצרים בכל קטגוריה?**

SELECT cat.category\_name AS category,

prod.product\_name AS product,

CUME\_DIST() OVER (

PARTITON BY cat.category\_name

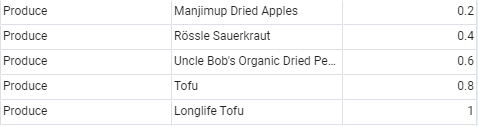
ORDER BY prod.unit\_price DESC

) AS "relative position "

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id



כאן, העמודה relative position מחשבת לנו, עבור כל מוצר, מה אחוז המוצרים מהקטגוריה שלהם שמחירם גדול או שווה למחיר שלהם (כולל עצמם).

אז נגיד עבור המוצר Tofu, 80% מהמוצרים בקטגוריה שלו (קטגורית produce) יקרים ממנו או שווים לו במחירם.

**VALUE**

פונקציות אלו מאפשרות לנו להציג לצד הערך הנוכחי ערכים אחרים מהקבוצה.

פונקציות אלו יכולות להיות שימושיות כשנרצה להעריך את השורה הנוכחית ביחס לשורות אחרות (לדוגמה להציג את המחיר של כל מוצר ולידו את מחיר המוצר היקר ביותר עבור כל ספק).

הפונקציות הן: **LAG, LEAD, FIRST\_VALUE, LAST\_VALUE**

1. **LAG ()** – מחזירה את הערך בסוגריים מהשורה שנמצאת לפני השורה הנוכחית בN שורות (נעביר כפרמטר) בחלון עבור כל שורה בחלון.
2. **LEAD()** – מחזירה את הערך בסוגריים מהשורה שנמצאת אחרי השורה הנוכחית בN שורות (נעביר כפרמטר) בחלון עבור כל שורה בחלון.

**בכל קטגוריה הצג עבור כל מוצר את מחירו ואת מחיר המוצר היקר/זול ממנו (מהיקר לזול)**

SELECT cat.category\_name AS category,

prod.product\_name AS product,

prod.unit\_price AS price,

LAG\LEAD(prod.unit\_price, 1) OVER (

PARTITION BY cat.category\_name

ORDER BY unit\_price DESC

) AS "higher price" ] \ "lower price"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id



**LAG**



**LEAD**

1. **FIRST\_VALUE()** – מחזירה את הערך בסוגריים מהשורה הראשונה בחלון עבור כל שורה בחלון.
2. **LAST\_VALUE()** – מחזירה את הערך בסוגריים מהשורה האחרונה בחלון עבור כל שורה בחלון.

**בכל קטגוריה הצג עבור כל מוצר את הפרש מחירו ממחיר המוצר היקר ביותר**

SELECT cat.category\_name AS category,

prod.product\_name AS product,

prod.unit\_price AS price,

FIRST\_VALUE(unit\_price) OVER (

PARTITION BY cat.category\_name

ORDER BY unit\_price DESC

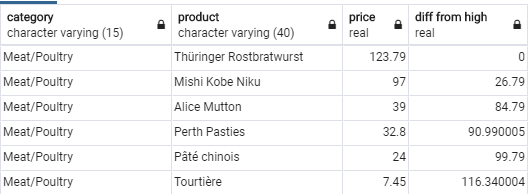
) – unit\_price

AS "highest price"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id



**בכל קטגוריה הצג עבור כל מוצר את הפרש מחירו ממחיר המוצר הזול ביותר**

SELECT cat.category\_name AS category,

prod.product\_name AS product,

prod.unit\_price AS price,

LAST\_VALUE(unit\_price) OVER (

PARTITION BY cat.category\_name

ORDER BY unit\_price DESC

) – unit\_price

AS "highest price"

FROM categories cat

INNER JOIN products prod ON

prod.category\_id = cat.category\_id

האם זה יעבוד?

התשובה היא **לא**.



אז איך נוכל לבצע את השליפה?

הסיבה שהשליפה למעלה לא עובדת היא בגלל שפונקציית LAST\_VALUE אשר שולפת את הערך מהשורה האחרונה, שולפת לנו בעצם את הערך מהשורה הנוכחית שהיא השורה האחרונה באותו הרגע (למעשה אנחנו החסרנו את המחיר מעצמו ובגלל זה קיבלנו 0), זה קורה בגלל ה- FRAME.

הגדרנו מקודם מהו FRAME - תת סדרה בתוך החלון שעליה הפונקציה תתבצע.

ה- FRAME הדיפולטי הוא מהשורה הראשונה עד לשורה הנוכחית (UNBOUNDED עד CURRENT ROW), ומשום שלא הגדרנו FRAME בשליפה, השורה האחרונה ב- FRAME הייתה גם השורה הנוכחית בעת ביצוע הפונקציה ולכן התוצאה הייתה 0 עבור כל המוצרים.

## סיכום

פונקציות חלון (או פונקציות אנליטיות) פותחות לנו עולם אפשריות, בעזרתן נוכל ליצור ולחקור קבוצות במידע ובשילובן יחד עם תתי שליפות, JOIN ופונקציות נוספות אותן למדנו נוכל ללמוד את ה- DB לעומק ולהפיק דוחות משמעותיים.

Look through the window