# PostgreSQL Enterprise Bacula Plugin Quick Guide - Bacula Latin America & BrazilSUBQUERY

### מי הם העובדים שמרווחים מתחת ל25,000 וכמה הם מרווחים?

SELECT first\_name,

last\_name,

salary

FROM employees

WHERE salary > 25000;

שליפה פשוטה ומוכרת...

בואו נעשה משהו מאתגר יותר!

הסר את המלבן הלבן בשביל לחשוף את התשובה

### *מה אם נשנה את השאלה למשהו יותר דינמי?*

### שאלה: מי הם העובדים שמרוויחים פחות מהמשכורת הממוצעת בחברה וכמה הם מרוויחים?

SELECT first\_name,

last\_name,

salary

FROM employees

WHERE salary < AVG(salary);

האם תעלה שגיאה? אם כן, איזו שגיאה נפגוש?

ERROR: aggregate functions are not allowed in WHERE

LINE 3: WHERE salary < AVG(salary);

^

פונקציות אגרגציה רצות ומחושבות לאחר סינון הטבלה בWHERE ולכן לא נוכל להשתמש בהן כתנאי בWHERE...

אז איך בכל זאת נוכל לבצע שליפה כזאת?

הסר את המלבן הלבן בשביל לחשוף את התשובה

### תת שליפה – SUBQUERY

SELECT first\_name,

last\_name,

salary

FROM employees

WHERE salary < (

SELECT AVG(salary)

FROM employees

);

נצטרך לבצע שליפה בתוך שליפה על מנת להשיג את הרשומות הנדרשות, זהו **Subquery**.

**הגדרה:**

* Subquery היא שליפה, שיכולה להיכתב מתוך פקודת DML אחרת SELECT, INSERT UPDATE, DELETE) ) הרעיון הוא לאפשר לקונן את הפקודה (למשל: SELECT) ע"י נתונים שנשלפים בשליפה אחרת, בעזרת תת השליפה.

תת השליפה מאפשרת לנו לכתוב שאילתות **יותר דינמיות ומונעות מידע**.

אנו כמובן נתמקד בפקודות .**SELECT** נחזור לשליפה שבדוגמה:

**הדבר הראשון שמתבצע הוא תת השליפה** – כלומר, תישלף המשכורת הממוצעת מהטבלת העובדים.

SELECT AVG(salary)

FROM employees;

קיבלנו את המשכורת הממוצעת בחברה - 22944.444444444444

לאחר מכן מתבצעת השליפה החיצונית כאילו היא הייתה שליפה רגילה. המשכורת הממוצעת שנשלפת מתת השליפה היא 22944.444444444444₪.

לאחר ביצוע תת השליפה, תיראה השליפה כך:

SELECT first\_name,

last\_name,

salary

FROM employees

WHERE salary < 22944.444444444444;

וזאת הופכת לשליפה פשוטה כמו בדוגמה הראשונה.

תת השליפה נותנת לנו לשלב את שני השלבים לשאילתה אחת שחוסכת מאתנו למצוא את הממוצע ולאחר מכן להכניס אותו כנתון hard-coded לשאילתה.

למעשה יצרנו שאילתה **דינמית** שתחזיר ממוצע חדש בכל שינוי במידע ותחזיר תוצאות בהתאם.

## מה יכול להופיע בתוך תת שליפה?

תת שליפה יכולה להכיל כל מה ששליפה רגילה יכולה להכיל: WHERE, GROUP BY, HAVING, JOIN וכד'.

Subquery יכול להכיל Subquery נוסף בתוכו (אין הגבלה למספר ה-Subquery-ים שאפשר לקנן זה בתוך זה).

## תת שליפות בפסוקיות שונות

כמו שליפה רגילה, תת שליפה יכולה להחזיר ערך בודד או מספר שורות, לכן, ניתן לבצע תת שליפה בכל אחת מן הפסוקיות בפקודת SELECT (SELECT, FROM, WHERE, JOIN, HAVING) תלוי איך מבצעים את תת השליפה.

**SELECT**

כאשר נשתמש בתת שליפה בתוך פסוקית הSELECT - נתייחס אליה כאל "ביטוי טורי" לכל דבר, כמו פונקציה טורית, חישוב מתמטי, טקסט, או עמודה, תת השליפה תחזיר ערך יחיד אשר יישלף פעם אחת ויישאר **קבוע** עבור כל רשומה שנשלפת.

**\*ניתן להשתמש גם בתת שליפה מקושרת אשר תשתנה בהתאם לרשומה שנשלפת.**

פעולה זו יכולה להיות שימושית במקרים בהם נרצה לשלוף ערך מטבלה אחרת (או אגרגציה) ולהשוות לערך מהטבלה בשליפה החיצונית.

**שאלה: מה המשקל של כל הזמנה לעומת המשקל הממוצע של הזמנה?**

SELECT order\_id,

freight,

(

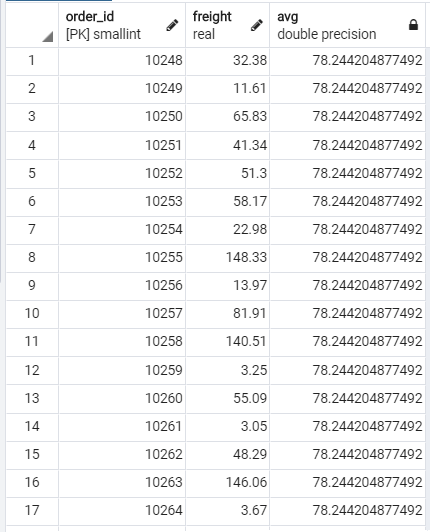
SELECT AVG(freight)

FROM orders

) AS "avg"

FROM orders

בתת השליפה נשלוף את המשקל הממוצע של הזמנה, ערך זה יישאר קבוע עבור כל רשומה ותת השליפה תתבצע פעם אחת בלבד, לאחר מכן תתבצע השליפה החיצונית.



**FROM/JOIN**

כאשר מבצעים תת שליפה ב- FROM או JOIN ניתן להתייחס אליה כמו טבלה רגילה ממנה נשלוף או נצרף מידע. פעולה זו יכולה להיות שימושית במיוחד עם **אגרגציות** (לדוגמה, אגרגציה על טבלה שכבר סוכמה בדרך כלשהי), השוואה בין משתנים שונים (על ידי JOIN או שליפה מטבלאות מקוסטמות לצורך הסקת מסקנות מעמיקה יותר.

**שאלה :כמה הזמנות יש בממוצע למדינה?**

SELECT ROUND(AVG(countries.Order\_Count), 2) AS "avg orders for country"

FROM (

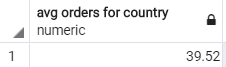
SELECT ship\_country,

COUNT(order\_id) AS Order\_Count

FROM orders

GROUP BY ship\_country

) countries;

****

**שאלה נוספת:**

**עבור כל מדינה שלוף את מספר ההזמנות שמשקלן גדול מהמשקל הממוצע שנשלח אל המדינה, עבור כל מדינה הצג גם את המשקל המקסימלי ומינימלי בטווח המשקלים שנשלף.**

SELECT ord.ship\_country,

MIN(ord.freight),

MAX(ord.freight),

COUNT(ord.order\_id)

FROM orders ord

INNER JOIN (

SELECT ship\_country,

AVG(freight) AS avg\_freight

FROM orders

GROUP BY ship\_country

) cnt\_avg ON

ord.ship\_country = cnt\_avg.ship\_country

WHERE ord.freight > cnt\_avg.avg\_freight

GROUP BY ord.ship\_country;

****

**שאלה:**

**הצג את המחיר הנוכחי של כל מוצר ואת המחיר הממוצע שבו הוא נמכר (אחרי הנחה)**

SELECT prod.product\_name,

prod.unit\_price,

avg\_sales.avg\_sale\_price

FROM products prod

LEFT JOIN (SELECT product\_id,

ROUND((AVG(unit\_price - unit\_price discount)::numeric), 2) AS "avg\_sale\_price"

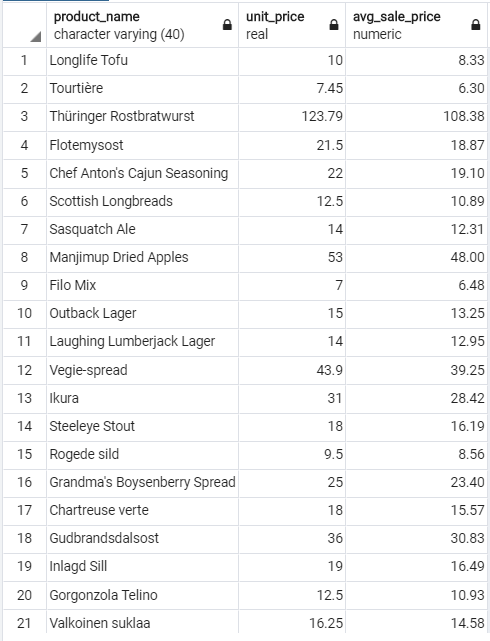
FROM order\_details

GROUP BY product\_id) avg\_sales ON

avg\_sales.product\_id = prod.product\_id

כמו שניתן לראות קודם כל יצרנו טבלה אשר מסכמת את המחיר הממוצע שבו נמכר מוצר לאחר הנחה מתוך טבלת פרטי ההזמנות, לאחר מכן צירפנו את הטבלה הזאת אל טבלת המוצרים שקיימת בDB דרך המפתח product\_id

הסר את המלבן הלבן בשביל לחשוף את התשובה

****

**WHERE/HAVING**

תת שליפה בפסוקיות אלו יכולה להחזיר ערך בודד או מספר שורות, האופרטורים בהם נשתמש בתנאי יהיו בהתאם לתוצאה המצופה מתת השליפה.

השימוש בתת שליפה פסוקיות אלו הוא לסינון הרשומות הנשלפות בהתאם לערכים אשר נשלוף בתת השליפה כפי שניתן לראות בדוגמאות הראשונות.

מוזמנים לצפות בסרטון: <https://www.youtube.com/watch?v=GpC0XyiJPEo>

## טיפול בשורות המוחזרות מתת שליפה

אם אנחנו משתמשים באופרטורים הלוגים הפשוטים כדי להשוות את הערך שמוחזר ע"י תת השליפה, אין אפשרות להשתמש בתת שליפה שמחזירה יותר משורה אחת. אם קורה מצב כזה, נקבל שגיאת זמן ריצה. למשל:

SELECT customer\_id,

company\_name,

country

FROM customers

WHERE customer\_id = (

SELECT DISTINCT customer\_id

FROM orders

WHERE (EXTRACT(YEAR FROM order\_date) = 1996) AND

(EXTRACT(MONTH FROM order\_date) = 7)

);

נקבל את השגיאה

ERROR: more than one row returned by a subquery used as an expression

הפקודה לא תקינה, כי תת השליפה מחזירה יותר מערך אחד (סביר להניח שיש יותר מאדם אחד שהזמין מוצרים בחודש הזה בשנה הזאת).

כדי לטפל בבעיה זו קיימים מספר פרדיקטים (ביטוי TRUE/FALSE אשר עוזרים לנו לטפל בתנאים שמעריכים מספר שורות, בחלקם נתקלנו בעבר.

### פרדיקטים לטיפול בתוצאות תת שליפה שמחזירה יותר משורה אחת:

**1. IN**

 WHERE שם טור IN (תת שליפה)

התנאי יחזיר TRUE אם הערך שבטור הוא **אחד** הערכים שמחזירה תת השליפה. לדוגמה:

WHERE customer\_id IN (

SELECT DISTINCT customer\_id

FROM orders

WHERE (EXTRACT(YEAR FROM order\_date) = 1996) AND

(EXTRACT(MONTH FROM order\_date) = 7)

)

##### **2.ANY**

 WHERE שם טור (<, =, >) ANY (תת שליפה)

התנאי יחזיר TRUE אם הערך שבטור מקיים את האופרטור הלוגי **לפחות עבור אחד** מהערכים שמחזירה תת השליפה. למשל התנאי:

  WHERE salary < ANY (

SELECT   AVG(salary)

                FROM employees

                GROUP BY country

);

יחזיר TRUE אם המשכורת הנבדקת קטנה לפחות מאחת המשכורות הממוצעות בכל מדינה

אם תת השליפה לא מחזירה ערכים התנאי יחזיר FALSE.

##### **3. ALL**

WHERE שם טור (<, =, >) ALL (תת שליפה)

התנאי יחזיר TRUE אם הערך שבטור מקיים את האופרטור הלוגי **עבור כל אחד** מהערכים שמחזירה תת השליפה. למשל התנאי:

WHERE customer\_id <> ALL (

SELECT customer\_id

FROM customers

WHERE contact\_title ~\* '.\*manager'

)

יחזיר TRUE אם הID הנבדק שונה מכל אחת מID של המנהלים.

**אם תת השליפה לא מחזירה ערכים התנאי יחזיר TRUE.**

### הפרדיקט ALL לעומת MAX

**שאלה: מה הם המוצרים הכי יקרים בכל קטגוריה? הצג את מספר הקטגוריה, שם המוצר, ומחיר מסודר לפי קטגוריות**

ניתן לענות על שאלה זו בשתי דרכים.

הראשונה – באמצעות שימוש בפרדיקט ALL.

SELECT prod.category\_id,

prod.product\_id,

prod.product\_name,

prod.unit\_price

FROM products prod

WHERE unit\_price >= ALL (

SELECT prodcat.unit\_price

FROM products prodcat

WHERE prod.category\_id = prodcat.category\_id

)

ORDER BY prod.category\_id;

הסר את המלבן הלבן בשביל לחשוף את התשובה

השנייה – באמצעות שימוש בMax - .

SELECT prod.category\_id,

prod.product\_id,

prod.product\_name,

prod.unit\_price

FROM products prod

WHERE unit\_price = (

SELECT MAX(prodcat.unit\_price)

FROM products prodcat

WHERE prod.category\_id = prodcat.category\_id

)

ORDER BY prod.category\_id;

הסר את המלבן הלבן בשביל לחשוף את התשובה

**מה עדיף?**

מבחינת יעילות: הפקודה MAX יותר יעילה מ ALL.

האם זה תמיד יעבוד? מה יקרה כשבתת שליפה לא ישלפו שורות?

אם השתמשנו ב ALL – כל השורות ישלפו.

אם השתמשנו ב MAX – אף שורה לא תישלף.

**מסקנה:** זה תלוי בשליפה.

בשליפה שבדוגמה שלפנו מטבלת המוצרים כך שאם יש מוצר חייבת להיות לו קטגוריה ולכן תמיד יהיה לפחות מוצר אחד בתת השליפה, משמע תמיד יהיו בה תוצאות.

לכן MAX יותר עדיף במקרה הזה.

**4.EXIST**

הפרדיקט EXISTS מאפשר לבדוק האם ישנן שורות כלשהן בטבלת התוצאה של תת השליפה.

WHERE   EXISTS  (תת שליפה)

התנאי יחזיר TRUE אם תת השליפה תחזיר לפחות שורה אחת.

למשל:

SELECT  first\_name, salary

FROM    employees

WHERE   EXISTS (

SELECT \*

  FROM employees

    WHERE salary > 10000

)

בדוגמא הזו שולפים את שמות כל העובדים בארגון, רק אם יש עובדים שמשכורתם גדולה מ- 10000.

## JOIN לעומת SUBQUERY

**מה הן הקטגוריות שיש תחתן יותר מ10 מוצרים?**

נוכל לפתור את השליפה בשתי דרכים:

דרך ראשונה: באמצעות תת שליפות ופרדיקטים.

SELECT cat.category\_name

FROM categories cat

WHERE cat.category\_id IN

(

SELECT prod.category\_id

FROM products prod

GROUP BY prod.category\_id

HAVING COUNT(\*) > 10

)

דרך שנייה: באמצעות JOIN.

SELECT cat.category\_name

FROM categories cat

INNER JOIN products prod ON

cat.category\_id = prod.category\_id

GROUP BY cat.category\_name

HAVING COUNT(\*) > 10

**מה ההבדל?**

JOIN ו Subquery שניהם משמשים להשגת מידע נוסף, מעבר למידע שיש לנו בטבלה כלשהי. לכן את חלק מהשאילתות תוכלו לפתור באמצעות שניהם(ההבדל בניהם הוא בעיקר לוגי).

מתי נשתמש ב Join?

כשנרצה לאחד מידע בין טבלאות. נרצה להתאים לכל רשומה בטבלה 1 את הרשומה המתאימה לה בטבלה 2 כדי להשיג עוד מידע.

**לדוגמה:** שלוף עבור כל עובד את שמו ואת שם המחלקה שבה הוא נמצא.

זהו מקרה קלאסי שבו נעשה איחוד בין המידע בטבלת t\_empl והמידע בטבלת t\_depart.

מתי נשתמש ב Subquery?

כשנרצה לשלוף ערך יחיד או כמה ערכים ורשומות לצורך סינון או לצורך הצגת מידע נוסף.

שימוש נפוץ של subquery הוא להחזרת תוצאות חישוב (סכום, ממוצע, מינימום, מקסימום).

**לדוגמה:** הצג את השמם ושכרם של כל העובדים שמרוויחים מעל השכר הממוצע השכר של בארגון. בנוסף הצג את ממוצע השכר.

SELECT first\_name,

(

SELECT AVG(salary)

                  FROM employees

)

FROM employees

WHERE salary > (

SELECT AVG(salary)

                  FROM employees

);

במקרה הזה השתמשנו בממוצע גם לסינון וגם להצגה.

**מה עדיף?**

**יעילות:** הפקודה Join יותר יעילה מ Subquery - באופן כללי תת שליפה (תת שליפה מקושרת עוד יותר) היא פעולה לא יעילה משום שהיא יכולה להיות מורכבת ודורשת סריקה של הטבלה בתת השליפה.

במקרה ונעבוד עם טבלאות גדולות הדבר יכול לגרור לזמני שליפה מאוד ארוכים.

לכן, נשתדל להשתמש בתת שליפה רק במקרים אשר באמת מצריכים תת שליפה. בעבודה עם טבלאות גדולות במקרים שבהם נזדקק לתת שליפה מומלץ להגביל את השורות הנשלפות בתת השליפה בעזרת LIMIT לצורכי בדיקת השליפה (שימו לב להוריד את פקודת הLIMIT - כשהשליפה מוכנה).

עובדה מעניינת: בחלק מהמקרים PostgreSQL תמיר את תת השליפה ל – Join.

לכן נעדיף להשתמש ב Join כשנוכל.

**לא בכוח:** למרות שהפקודה Join נחשבת יותר יעילה אין צורך לנסות להמיר כל תת שליפה ל Join. חשבו על ההבדל הלוגי(זה לא תמיד נכון להשתמש ב Join מבחינה לוגית). בנוסף, לפעמים זה יכול לסבך את השליפה ולהפוך אותה לפחות קריאה.

**הוראות:** אם כתוב בהוראות התרגול/ התר"צ להשתמש רק ב Subquery או רק ב Join נכתוב את השליפות לפי ההוראות!

שימו ♥ במקרה של השליפה לדוגמה שתי התשובות טובות ושתיהן יתקבלו.

זה תחום אפור הפעילו שיקול דעת!

## CTE

CTE (Common Table Expression), מאפשר ליצור טבלת תוצאה זמנית שזמינה רק בתוך מרחב הביצוע וזמן הריצה של פקודת DML (SELECT/INSERT/UPDATE…), נשמע מוכר?

SUBQUERY ו - CTE מאוד דומים בשימושם, למעשה CTE הוא תוספת מאוחרת יותר לשפת ה - SQL.

### מבנה של פקודת SELECT בשימוש עם CTE

WITH

cte\_name(col1, col2, ...)

AS

(

SELECT col1,

col2,

...

FROM …..

)

נפרק את ה - Syntax

* **שלב ראשון** נכתוב את פסוקית WITH אשר מכריזה על ה - CTE ולאחר מכן נגדיר שם ל - CTE אליו נוכל להתייחס בשליפה המרכזית כמו אל טבלה רגילה.
* **שלב** **שני** נוסיף לאחר שם ה CTE - בסוגריים רשימת עמודות (אלו יהיו שמות העמודות של הטבלה הזמנית), מספר העמודות חייב להיות זהה למספר העמודות הנשלפות ב cte definition - אם לא נציין עמודות, עמודות ה CTE יהיו כשמות העמודות הנשלפות בcte definition - .
* **שלב שלישי** נכתוב את פקודת AS כדי להגדיר את התוכן CTEה - (cte\_definition).
* **שלב רביעי** תחת פקודת ה - AS נכתוב שליפה רגילה בעזרת פקודת SELECT אשר תגדיר את תוכן הטבלה הזמנית של ה - CTE (חלק זה הוא ה - cte definition).
* **שלב אחרון** שימוש בטבלה שנוצרת כתוצאה מCTE - בשאילתת SQL (נשתמש בטבלה על ידי השם שהגדרנו כמו בטבלה רגילה).

**שאלה: כמה הזמנות יש בממוצע למדינה?**

SELECT ROUND(AVG(countries.Order\_Count), 2) AS "avg orders for country"

FROM (

SELECT ship\_country,

COUNT(order\_id) AS Order\_Count

FROM orders

GROUP BY ship\_country

) AS countries

נהפוך את תת השליפה ל - CTE

WITH

cte\_countries\_orders(country, order\_count)

AS

(

SELECT ship\_country,

COUNT(order\_id)

FROM orders

GROUP BY ship\_country

)

----- Main Query -----

SELECT ROUND(AVG(order\_count), 2) AS "avg orders for country"

FROM cte\_countries\_orders

תחילה הכרזנו על CTE בשם cte\_countries\_orders עם עמודות countries ו order\_count - לאחר מכן יצרנו שליפה המחזירה את המידע על כמות הזמנות פר מדינה אל ה CTE - לבסוף יצרנו לשליפה אשר מתייחסת אל ה – CTE (שולפת ממנו) ולמעשה שלפנו את הממוצע של עמודת order\_count מה .CTE -

**שימו לב! ב-CTE לא נשתמש ב-ORDER BY, מכיוון שלא רואים את טבלת התוצאה של הCTE, אלא אם יש צורך או ביקוש בשאלה.**

אם זה כמו SUBQUERY אז למה בכלל להשתמש בCTE? למה שניהם קיימים?

מוזמנים לצפות בסרטון: <https://www.youtube.com/watch?v=_SanZ41uTlw>

**CTE VS SUBQUERY**

כפי שנאמר - CTE וSUBQUERY - דומים כאשרCTE הינו תוספת מאוחרת יותר לשפה, אך בכל זאת, ישנם הבדלים וסיבות להשתמש בכל אחד.

למה להשתמש בCTE - ?

* + **קריאות**: כמו שניתן לראות בדוגמה למעלה ה - SYNTAX של CTE מאפשר לנו כתיבה יותר מסודרת ויותר קריאה של שליפות בכך שאנחנו מכריזים על "תתי שליפות" מראש, בצורה זו יהיה לנו יותר קל לדבג את הקוד, אומנם בשליפה הנ"ל זה נראה זניח אך בשליפות יותר מורכבות עם תתי שליפות יותר מורכבת ההבדל בא יותר לביטוי.

**לדוגמה:**

* **כתיבה יעילה:** השליפה הנכתבת ב - CTE ניתנת לשימוש חוזר על ידי ציון שמה של הטבלה ובכך חוסכת לנו את הצורך לכתוב מחדש תת שליפה אשר חוזרת מספר פעמים בשליפה החיצונית.

ניתן להשתמש בCTE בתוך CTE אחר בתנאי שהוגדר לפניו. בעזרת CTE ניתן ליצור שליפות רקורסיביות (מועילות בשליפת מידע היררכי), אך לא נעמיק בנושא זה.

**שאלה:**

**הצג עבור כל עובד סיכום הזמנה שביצע הכולל שם עובד, מספר הזמנה, שם חברת הלקוח, מספר מוצרים ומחיר כולל, יש להציג רק הזמנות שסכומן מעל 1000**

WITH

order\_summery(order\_id, products, total\_price)

AS

(

SELECT order\_id,

COUNT(product\_id),

SUM(unit\_price \* (1 - discount) \* quantity)

FROM order\_details

GROUP BY order\_id

),

employee\_orders(employee, order\_id, customer\_company, products, total\_price)

AS

(

SELECT emp.first\_name || ' ' || emp.last\_name,

order\_summery.order\_id,

cust.company\_name,

order\_summery.products,

order\_summery.total\_price

FROM orders ord

INNER JOIN customers cust ON

cust.customer\_id = ord.customer\_id

INNER JOIN order\_summery ON

order\_summery.order\_id = ord.order\_id

RIGHT JOIN employees emp ON

emp.employee\_id = ord.employee\_id)

----- Main Query -----

SELECT employee,

order\_id,

customer\_company,

products,

total\_price

FROM employee\_orders

WHERE total\_price > 1000;

למה להשתמש בSUBQUERY - ?

ניתן להשתמש ב - SUBQUERY כעמודה (בתנאי שהוא מחזיר ערך אחד), עם CTE הדבר אינו אפשרי.

ניתן להשתמש בSUBQUERY כתנאי לפילטור שורות (כשהוא מחזיר ערך יחיד או בעזרת פרידקיטים), עם CTE הדבר אינו אפשרי (אלא אם עושים SUBQUERY על הCTE).

בהשוואה של CTE אל מול SUBQUERY:

מבחינת יעילות- **אין הבדל חד משמעי בין הביצועים** של שתי הפעולות ולרוב יבוצעו באותה צורה על ידי

ה DB -, לכן, נשתמש בכל אחד מהם על פי שיקולים אחרים:

נראות והבנה – קוד נקי ומסודר, יותר קל להבנה.

כתיבה – כתיבת הקוד ואופן העבודה איתו.

**אז במה נשתמש ומתי?**

****נעבור עכשיו לסיכום כללי של איך נשלב את תתי השליפות בפסוקיות השונות.

* עבור שליפת **ערך יחיד**:

נשתמש בSUBQUERY בפסוקית WHERE או SELECT.

נשתמש בה בפסוקית הSELECT בשביל להציג את הערך- לדוגמה להציג את המשכורת הממוצעת של העובדים.

נשתמש בה בפסוקית הWHERE על מנת לסנן לפי הערך- לדוגמה לייצר תנאי שבו נציג את כל העובדים שהמשכורת שלהם גדולה מהמשכורת הממוצעת.

* עבור שליפת **עמודה**:

נשתמש בSUBQUERY בWHERE

בפסוקית הWHERE נוכל ליצור סינון עפ"י העמודה- לדוגמה נשלוף עמודה של המשכורת הממוצעת של כל אחד מהתפקידים הקיימים בחברה וניצור תנאי שבו הרשומות שישלפו בשליפה הראשית יהיו של עובדים שהמשכורת שלהם גבוהה מלפחות אחד ממשכורות הממוצעות.

* עבור שליפת **טבלה**:

נשתמש **בCTE** או ב**SUBQUERY בFROM/JOIN**

במידה וקיים צורך לשלוף יותר מעמודה אחת בתת השליפה אנו נתייחס לתוצאה של אותה תת שליפה כטבלה ולכן נחבר אותה בשליפה הראשית בFROM או בJOIN.

**חשוב להדגיש:** CTE וSUBQUERY בFROM/JOIN משמשים לאותה מטרה, כן נשאף להשתמש ב – CTE במקרה של שליפה של טבלה, זאת כדי להפוך את הקוד לקריא יותר ולשימוש חוזר באותה טבלה. במקרים בהם השאילתה קצרה ניתן להשתמש ב – Subquery(תוך הקפדה על סטנדרטים ונראות).

**נקודות נוספות**

* לCTE ניתן וחייב להגדיר שם (Subquery יכול לקבל שם רק בשימוש FROM) לכן ניתן להתייחס אליו במקומות שונים.
* ניתן להשתמש בCTE אך ורק בשליפה המרכזית (שאינה CTE) שבאה לאחר סיום פקודת ה – WITH.
* ישנם מקרים בהם ה - DB יצטרך לבצע materialization לCTE - (יצירה ממשית של הטבלה ושמירתה בזיכרון לצורך שימוש מאוחר יותר) דבר אשר יכול לפגוע בביצועים משום שהוא מונע מהDB לתכנן שליפה יעילה על ידי הסתכלות על כלל השליפה (כולל החיצונית), הדבר משתנה מ - DBMS ל DBMS - ולכן כלל האצבע יהיה לבדוק על ה DB עליו נבצע שליפות ולהשוות זמני ריצה (בפוסטגרס לרוב הביצועים יהיו זהים אך זה תלוי בשליפה ואם יש חשד להאטה מומלץ לבדוק את 2 האופציות(.

**סיכום**

תת שליפות יכולות להיות מאוד מעילות בין אם זה בצורת SUBQUERY או CTE והן חלק בלתי נפרד מן השפה.

בעזרת תת שליפות נוכל לייבא מידע נוסף אשר יעזור לנו לכוון את השליפות לפיו וליצור שליפות דינמיות או לאחד טבלאות מקוסטמות אשר נוכל להפיק מהן מסקנות משמעותיות, אך כפי שצוין, תת שליפות יכולות לפגוע בביצועי השאילתות משום שהן דורשות למעשה לבצע עוד שליפה.

במידה ונעבוד עם הרבה מידע ואו עם הרבה תתי שליפות נסתכן בהאטת מהירות השליפה, לכן יש להשתמש בתתי שליפות במקרים אשר הפתרון באמת מצריך תת שליפה, אם ניתן לפתור בדרך אחרת, זאת לרוב תהיה הדרך המומלצת.