**חפיפת SQL**

# מה זה SQL

**SQL** (**Structured Query Language**) היא שפת מחשב שנועדה לשליפות מידע מבסיס נתונים ולעיבודו. באמצעות השפה ניתן לייצר בסיסי נתונים ולכתוב שאילתות לשליפת מידע מהם.

**שאילתה (Query)** – בקשה לשליפת נתונים מבסיס נתונים. במילים אחרות, מדובר באוסף של פקודות שהפלט שלהן יכול להיות טבלה (כמו לדוגמא – כל העובדים של חברה שהצטרפו אליה בין השנים 2010 ו-2012) או נתון בודד (כמו לדוגמא – השכר ממוצע של העובדים במחלקה מסוימת).

SQLנכתבה עבור מסדי נתונים יחסיים.

# מה זה מסד נתונים יחסי?

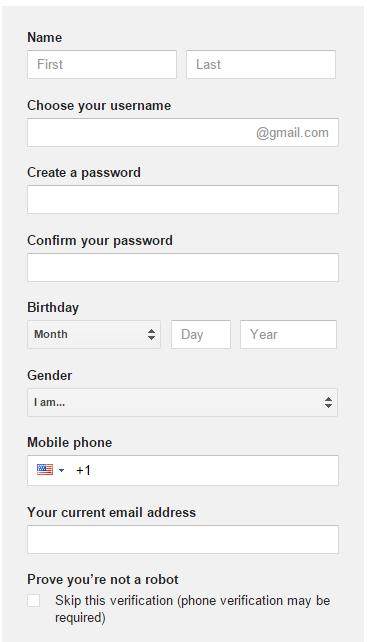
**מסד נתונים יחסי** (**Relational DataBase** או **RDB**) הוא מסד נתונים הבנוי מטבלאות, כך שבין הטבלאות שלו יש יחסים.

כל טבלה מורכבת **משדות** (**field**), כמו לדוגמא מספר תעודת זהות או שם פרטי. לדוגמא:

|  |  |  |
| --- | --- | --- |
| **מספר ID** | **שם פרטי** | **שם משפחה** |
| 123456789 | ישראל | ישראלי |
| 987654321 | יעקב | יעקובי |
| 112233445 | יוסי | יוסף |

בדוגמא לעיל ניתן לראות שיש לנו 3 עמודות. עמודה היא אוסף של ערכים. לדוגמא, השדה שם פרטי מכיל את הערכים "ישראל", "יעקב" ו"יוסי". שדה הוא ערך מסוים בטבלה, כמו "יוסי" או "123456789".

קצת מבלבל, אבל תחשבו על שדה כמו על שדה בטופס הרשמה לאתר:

בטופס יש לנו את השדות כמו שם פרטי, שם משפחה, סיסמא וכו'. בטבלה שתשמור את כל המידע הזה תהיה לנו **עמודה** לכל אחד **מהשדות** שתכיל את כל המידע שרלוונטי לשדה הזה.

כל **שורה** בטבלה יכולה להיקרא גם **רשומה**.

## מפתחות

לכל טבלה יש שדה אחד לפחות שמוגדר כ**מפתח ראשי** (**Primery key**). מפתח ראשי הוא שדה שמגדיר באופן חד חד ערכי את ערכי הרשומה. לדוגמא, השדה של מספר ת.ז הוא המפתח הראשי של הטבלה ממקודם **כי לכל בן אדם יש מספר ת.ז ייחודי לו. שדה זה לא יכול להכיל ערך ריק (NULL).**

**מפתח זר** (**foreign key**) הוא שדה אחד או יותר שמוגדר כמפתח ראשי בטבלה אחרת. לדוגמא, אם הייתה לנו עוד טבלה של הזמנות:

|  |  |  |
| --- | --- | --- |
| **ID הזמנה** | **מספר הזמנה** | **ID מזמין** |
| 1 | 12345 | 123456789 |
| 2 | 54321 | 987654321 |
| 3 | 11223 | 112233445 |

בטבלה זה ה-**ID של המזמין מוגדר כמפתח זר היא הוא מפתח ראשי בטבלה של האנשים שלנו**.

**ניתן לייצר טבלה עם יותר ממפתח אחד.**

לדוגמא, בטבלה של חיילים אנחנו יכולים להגדיר שגם הת.ז. וגם המספר אישי יהיו מפתח. גם במקרה זה, הסט של המפתחות מגדיר באופן חד חד ערכי את הרשומות בטבלה. אבל, במקרה זה שניהם לא נחוצים כדי להגדיר באופן חד חד ערכי את הרשומה. לכן, נקרא לסט של המפתחות האלה super key.

בואו ניקח עוד דוגמא של הזמנות:

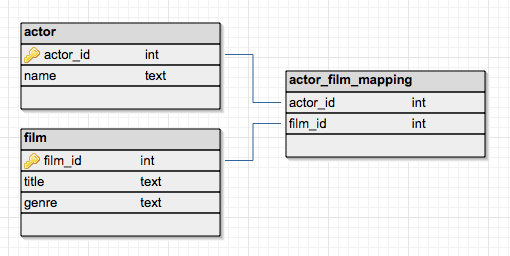
|  |  |  |  |
| --- | --- | --- | --- |
| מספר הזמנה | ID מוצר | שם מוצר | כמות |
| B005 | JAP102459 | Mouse | 5 |
| B005 | DKT321573 | USB | 10 |
| B005 | OMG446789 | LCD Monitor | 20 |
| B004 | DKT321573 | USB | 15 |
| B002 | OMG446789 | Laser Printer | 3 |

**אף אחד מהשדות הנ"ל לא יכול להיות מפתח בפני עצמו. זאת מפני שלכל הזמנה יכולים להיות כמה מוצרים ולכן מספר הזמנה מופיע ביותר משורה אחת. במקרה זה, שילוב של מספר הזמנה וID- מוצר מספקים לנו קשר חד חד ערכי.**

## יחסים בין טבלאות

במסד נתונים סטנדרטי יש בד"כ יותר מטבלה אחת. את הקשרים האלה מייצרים בעזרת המפתחות. כאשר מגדירים קשר בין שתי טבלאות, אחת מהטבלאות מוגדרת כטבלה הראשית, והשנייה כטבלה המשנית.

סוגי קשרים:

* **קשר של יחיד ליחיד** - קשר בו לכל רשומה בטבלה הראשית יש רשומה אחת ויחידה בטבלה המשנית. **לדוגמא**, אם נתונות לנו שתי טבלאות: **טבלה של חיילים** (שמכילה ID ופרטים על הבן אדם) **וטבלה ששומרת מספר חוגר עבור כל אדם** (כלומר מכילה שדה של ID של בן אדם ומספר ביטוח לאומי), ברגע שנקשר את שתי הטבלאות על פי השדה ID הקשר יהיה יחיד ליחיד, **מפני שלכל בן אדם יש מספר חוגר אחד ויחיד**.
* **קשר של יחיד לרבים** - קשר בו לכל רשומה בטבלה הראשית יכולות להיות מספר רשומות בטבלה המשנית. לדוגמא, **לכל אדם יש מספר חיילים אבל כל חייל משויך למפקד ישיר אחד בלבד.**
* **קשר של רבים לרבים**- קשר בו לכל רשומה באחת מהטבלאות המקושרות יכולים להיות מספר רב של רשומות בטבלה השנייה. (יחס זה מיושם באמצעות טבלת ביניים, שמחברת בין שתי הטבלאות המתייחסות). לדוגמא, כל שחקן יכול להופיע ביותר מסרט אחד וכל סרט יכול להכיל יותר משחקן אחד. במקרה זה נייצר עוד טבלה שיהיו בה שתי שדות: ID של השחקן ו-ID של הסרט. **לטבלה זאת קוראים Associative entity.**

**כעת, בטבלה החדשה שיצרנו הרשומות ייצגו קשר חד חד ערכי – כל שחקן יכול להופיע בסרט פעם אחת בלבד וסרט יוכל להכיל יותר משחקן אחד.**

# היסטוריה של SQL

בלה בלה לא מעניין אף אחד.

# פלטפורמות

ישנם מספר פלטפורמות שמאפשרות שימוש ב-SQL, כמו Microsoft SQL Server , Oracleו-HANA. לכל אחת מהן יש Syntax מעט שונה עבור כל מיני דברים. בכל מקרה, במדריך זה נציג את ה-syntax הבסיסי שיהיה נכון עבור רוב הפלטפורמות.

# פקודות SQL

ישנן מספר סוגי פקודות של SQL: DDL ,DML ו-DRL.

DDL (Data Definition Language) – פקודות שמשמשות לטיפול במבנה הנתונים, כמו יצירת טבלאות, מחיקת טבלאות וכו'.

DML (Data Manipulation Language) – פקודות שמשמשות לטיפול בנתונים עצמם כמו הוספה/עדכון/מחיקה של שורות וכו'.

DRL (Data Retrieval Language) – פקודות שמשמשות לאחזור ושליפת נתונים.

# DRL

נתחיל בללמוד על הפקודות שמשמשות ליצירת שאילתות.

# פקודות ואופרטורים בסיסיים (SELECT, WHERE, OR, IN, AND, NOT)

להלן דוגמא לשליפה בסיסית ב-SQL:

SELECT \* FROM Customers;

**הפקודה SELECT היא פקודה ששולפת נתונים מטבלה מסויימת. היא עושה זאת מתוך הטבלה שכתובה לאחר ה-FROM. שתי המילים הנ"ל הן מילים שמורות, כלומר לא ניתן להשתמש בהן בתור שמות לשדות או טבלאות. את כל המילים השמורות נכתוב עם אותיות גדולות. לא מדובר במשהו הכרחי - הקוד עדיין יעבוד אם לא – אבל מדובר במשהו שיהפוך את הקוד לקריא הרבה יותר.**

**בהגה המקצועית, נכתוב ונתייחס ל-SELECT כאל "פסוקית". בעתיד נלמד על עוד פסוקיות.**

במקרה שלנו, ניתן לראות שרשום \* לאחר ה-SELECT, מה שיגרום לשאילתה להחזיר לנו את כל השדות שקיימות בטבלה. במידה ונרצה רק את השם הפרטי והמשפחה של הלקוחות שלנו, נרשום:

SELECT first\_name, last\_name FROM Customers;

במקום \* נרשום את השדות שאותן נרצה שלוף. ניתן גם לראות **ששם של שדה לא יכול להכיל רווח או כל מיני תווים מיוחדים כמו $ או % ולכן נהוג להפריד מילים עם \_ או עם אותיות גדולות – לדוגמא firstName.**

אוקיי. סבבה. נחמד. שלפנו מהטבלה Customers את כל השמות של כל הלקוחות שלנו. אבל במקרה זה, אנחנו נקבל את **כל הרשומות שיש לנו בטבלה**. אחרי הכל, המטרה של השליפות הן לשפוף רק את המידע שאנחנו רוצים ולא את כולו.

בשביל זה המציאו את ה-**WHERE**. ה-WHERE היא פסוקית שמאפשרת לנו לשלוף את כל הרשומות שבהן מתקיים **תנאי מסויים או מספר תנאים מסויימים**.

לדוגמא:

SELECT first\_name, last\_name FROM Customers

WHERE first\_name = 'Yossi';

השליפה הזאת תיתן לנו את שמות כל הלקוחות ששמם הפרטי הוא יוסי, כלומר את הרשומות בהן התנאי שם פרטי = יוסי הוא מחזיר אמת (כלומר נכון).

אם נרצה את כל הלקוחות שקוראים להם יוסי, יאיר, יוחנן וירין נרשום

WHERE first\_name = 'Yossi' OR first\_name = 'Yair' OR first\_name = 'Yohanan' OR first\_name = 'Yarin';

כדי לעשות זאת נעשה פה שימוש באופרטור **OR** שיחזיר לנו את הרשומות אם אחד מהתנאים הוא נכון.

כדי לחסוך בכתיבה, המציאו את האופרטור **IN** שמאפשר לעשות את אותו דבר בדיוק רק בלי לכתוב כל פעם OR או first\_name:

WHERE first\_name IN ('Yossi', 'Yair', 'Yohanan' 'Yarin');

ואם נרצה את כל הרשומות שבהם השם הוא "יוסי כהן"? במקרה זה נצטרך לכתוב תנאי גם עבור שם המשפחה.

WHERE first\_name = 'Yossi' AND last\_name = 'Cohen';

ניתן לראות שעשינו שימוש באופרטור **AND** שיחזיר לנו את הרשומות רק עם השם הפרטי הוא יוסי וגם שם המשפחה הוא כהן. אם היינו משתמשים ב-OR, היינו מקבלים את כל הרשומות בהן השם הפרטי הוא יוסי או שם המשפחה הוא כהן, ואז היינו מקבלים אנשים כמו מרדכי כהן או יוסי שמחיוב, מה שלא התכוונו אליו.

באותה צורה, ניתן לשלוף את כל האנשים ששמם הפרטי הוא יוסי או יאיר ושם המשפחה שלהם הוא כהן:

WHERE first\_name IN('Yossi','yair') AND last\_name = 'Cohen';

ומה אם אנחנו רוצים לשלוף את כל אלה ששמם הוא **לא יוסי.** פשוט. נשתמש באופרטור NOT:

WHERE NOT first\_name='Yossi';

ניתן לראות שהאופרטור ממוקם לפני התנאי שלנו.

# כתיבת קוד קריא

אולי בשלב הזה לא נתקלו בקוד מאוד מסובך ומסורבל, אבל תאמינו לי – זה יגיע. וכשזה יגיע, צריכים להיות מוכנים.

איך מתכוננים?

יש כל מיני קונבנציות לכתיבת קוד SQL קריא. לכל מקום עבודה יש את הקונבנציות שלו ולא מדובר במשהו מאוד אוניברסלי.

אבל בכל זאת, כמה טיפים:

* אם יש לכם שורה שיש בה הרבה תנאים והיא הופכת לממש ארוכה – תפצלו אותה לשתי שורות. זה מותר. לאחר אופרטור AND או OR תלחצו על אנטר ותפלצו את השורה לשתיים.
* שמות של פקודות ואופרטורים – באותיות גדולות, כמו בדוגמאות שלנו.
* לרוב משתמשים בטאבים כדי לסדר את הקוד. לדוגמא:

SELECT

T1.col1

FROM

table1 T1

* בסופו של דבר תחשבו על הקורא – הקוד צריך להיות קריא וברור. תשימו את עצמכם בנעל של מישהו חיצוני שזה עתה רואה את הקוד שלכם ותחשבו אם הוא היה מסתדר אתו. אם לא, אז יש מקום לשיפור.

# SELECT DISTINCT

תחשבו על מקרה בו שיש לכם טבלה של מכירות ואתם רוצים לקבל רשימה של כל הלקוחות שקנו. בטבלה יש לכם שדות כמו ID של קנייה, ID קונה, ID של חנות וכו'.

אם היינו עושים משהו כמו:

SELECT customerID FROM Orders;

היינו מקבלים משהו כזה:

123456789

123456789

987654321

123412345

987654321

אפשר לראות שיש לנו כפילות של ID-ים. למה זה? כי לקוח אחד יכול לבצע יותר מהזמנה אחת, ולכן הוא מופיע ביותר מרשומה אחת. לכן, כדי לקבל את כל הלקוחות ברשימה בה כל אחד מהם מופיע פעם אחד בלבד נשתמש באופרטור DISTINCT:

SELECT DISTINCT customerID FROM Orders;

במקרה זה SQL יצמצם לנו רשומות זהות ולא יציג לנו כל לקוח כמה פעמים שהוא מופיע כתלות במספר ההזמנות שלו.

ניתן לבחור ערכים distinct-ים עבור יותר משדה אחד:

SELECT

    DISTINCT product\_id,

    quantity

FROM

    ORDER\_ITEMS

ORDER BY

    product\_id;

שימו לב: הפעולה תסנן לנו את כל השורות הזהות לא רק עבור product\_id אלא גם עבור quantity (כל השדות). לא ניתן לבצע distinct עבור שדה אחד ולהציג ב-select עוד שדות.

# אופרטור BETWEEN

אופרטור זה מאפשר לנו לסנן שדה עבור טווח ערכים. לדוגמא:

SELECT

    product\_name,

    standard\_cost

FROM

    products

WHERE

    standard\_cost BETWEEN 500 AND 600

ORDER BY

    standard\_cost;

בדוגמא זו, סיננו את המוצרים שמחירם בין 500 ל-600 **(כולל 500 וכולל 600)**.

ניתן לעשות זאת גם עבור תאריכים:

order\_date BETWEEN DATE '2016-12-01' AND DATE '2016-12-31'

**\*\*הפורמט של התאריך שלמעלה ('YYYY-MM-DD') נכון ל-DB של אורקל ויכול להיות שונה עבור DB שונים. לכל DB יש את האפשרות לשנות את הפורמט הזה.**

# CASE

CASE הוא אופרטור שיופיע בפסוקית SELECT שמאפשר לנו לייצר תנאים (אם ככה, אז ככה ועו').

בואו נראה דוגמא:

SELECT OrderID, Quantity,  
CASE  
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
    WHEN Quantity = 30 THEN 'The quantity is 30'  
    ELSE 'The quantity is under 30'  
END AS QuantityText  
FROM OrderDetails;

ניתן לראות שייצרנו עמודה חדשה שנקראת QuantityText שהתוכן שלה תלוי בכמות של ההזמנה. כאשר (WHEN) הכמות היא מעל 30 אז (THEN) ערך העמודה יהיה 'The quantity is greater than 30', כאשר (WHEN) הכמות היא שווה ל-30 אז (THEN) ערך העמודה יהיה 'The quantity is 30', ואם אף אחד מהתנאים שהוצגו למעלה (ELSE) לא מתקיים, תציג 'The quantity is under 30' (זאת אומרת שאם הכמות היא לא מעל 30 ולא שווה ל-30, היא חייבת להיות מתחת לשלושים).

הפקודה נסגרת באמצעות כתיבת END, כלומר כל מה שרשום לנו בין ה-CASE ל-END שייך לתנאי שלנו.

# אופרטור LIKE

LIKE הוא אופרטור ממש מגניב. הוא מאפשר לנו לחפש בפסוקית ה-WHERE ערכים שמתאימים **לתבנית מסוימת**.

במקום לחפור לכם בואו נראה דוגמאות (הרבה מאוד דוגמאות):

WHERE CustomerFirstName LIKE 'A%'

בדוגמא זאת אנחנו מחפשים לקוחות ששמם הפרטי מתחיל ב-A.

מה זה ה-%?

ה-% אומר ש"כל מחרוזת של תווים שתבוא אחר כך מתקבלת על הדעת". כלומר, במקרה שלנו, רצינו רק את אלה ששמם הפרטי מתחיל ב-A. אבל, לא הגדרנו תנאי אחר חוץ מזה. יכול להיות ששמו של הבן אדם הוא Arial ויכול להיות ששמה היא Ana. **ה-% אומר שיכול לבוא אחרי ה-A איזה מחרוזת שאנחנו רוצים באיזה גודל שאנחנו, גם מחרוזת בגודל אפס (כלומר אין תווים אחרי ה-A).**

\*אם היינו רוצים לסנן רק את השמות שמתחילים ב-A וששמם הוא באורך של 5 תווים, נכתוב:

WHERE CustomerFirstName LIKE 'A\_\_\_\_'

הסימן \_ מתייחס לתו בודד. כלומר ברגע שרשמנו 4 קוום תחתונים אחרי ה-A נקבל מחרוזות ששמם הפרטי מתחיל ב-A ויש אחרי ה-A עוד **ארבעה תווים בדיוק.**

אם נרצה את אלה ששמם הפרטי מתחיל ב-"Ar", נכתוב:

WHERE CustomerFirstName LIKE 'Ar%'

אם נרצה את אלה ששמם הפרטי מתחיל ב-A ונגמר ב-l, נכתוב:

WHERE CustomerFirstName LIKE 'A%l'

השם יכול להיות בגודל של 2 אותיות, 3 אותיות ואפילו 50. כל עוד השם מתחיל ב-A ונגמר ב-l אנחנו נקבל את השם הזה בשליפה. בין ה-A ל-l יכולות להיות כל מחרוזת בכל גודל אפשרי.

אם נרצה את אלה שבשמם הפרטי יש את האות a, נכתוב:

WHERE CustomerFirstName LIKE '%a%'

**כלומר כל עוד יש a בשם, נקבל אותו בשליפה.**

אם נרצה את אלה ששמם הפרטי מתחיל ב- A**ושהאות השלישית בשם שלהם היא i**, נכתוב:

WHERE CustomerFirstName LIKE 'A\_l%'

השם יכול להיות בגודל של 2 אותיות, 3 אותיות ואפילו 50. כל עוד התו הראשון הוא A, השני הוא תו כלשהו, השלישי הוא l ומה שבה אחר כך מתקבל בברכה, אנחנו נקבל את השם הזה בשליפה.

עכשיו, אם נרצה את השמות במתחילים ב-A או ב-B או ב-C? איך נעשה את זה?

באמצעות סוגריים מרובעים []:

WHERE CustomerFirstName LIKE '[ABC]%'

נכניס את התווים שאנחנו רוצים שאחד מהם יופיע לתוך הסוגריים המרובעים. הסוגריים מייצגים תו אחד בודד שהערך שלו חייב להיות או A או B או C.

ואם אנחנו רוצים את כל השמות שמתחילים באותיות A-M? נשתמש במקף (-) כדי להגדיר טווח של תווים:

WHERE CustomerFirstName LIKE '[A-M]%'

# שרשור (אופרטור ||)

אם לדוגמא יש לנו בטבלה את השם של הבן אדם מופרד לשם פרטי ושם משפחה, אפשר להשתמש באופרטור || כדי לשרשר את השם עמודה אחת:

SELECT first\_name || ' ' || last\_name AS contact\_name

FROM contacts;

תראו שלא רק אפשר לשרשר עמודות אלא גם לשרשר מחרוזת באמצע כמו הרווח שהוספנו לשם המלא.

# Aliases

לפעמים השמות של השדות בטבלאות שלנו הם לא משהו שכיף להציג בתוצאה של שאילתה. לרוב אנחנו לא נרצה להציג עמודה ששמה customer\_id כי זה שם טכני, מכוער, מגעיל, מבייש... הבנתם את הרעיון.

בשביל זה המציאו את ה-.Alias הוא מאפשר לנו לשנות את שם התצוגה של עמודה בשאילתה. לדוגמא:

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

באמצעות הפקודה AS נתנו שם חדש לשדה. במידה ונרצה שם שמכיל רווח, נכניס את כולו למרכאות (").

אז, במקום משהו מגעיל כמו

|  |  |
| --- | --- |
| Customer\_id | CustomerName |
| 1 | Alfreds Futterkiste |
| 2 | Ana Trujillo Emparedados y helados |
| 3 | Antonio Moreno Taquería |
| 4 | Around the Horn |
| 5 | Berglunds snabbköp |
| 6 | Blauer See Delikatessen |

נקבל תוצאה

|  |  |
| --- | --- |
| ID | Customer |
| 1 | Alfreds Futterkiste |
| 2 | Ana Trujillo Emparedados y helados |
| 3 | Antonio Moreno Taquería |
| 4 | Around the Horn |
| 5 | Berglunds snabbköp |
| 6 | Blauer See Delikatessen |

מה שמיד הופך את הטבלה להרבה יותר מגניבה.

ניתן לבצע alias גם עבור טבלאות שלמות – אבל זה משהו שנלמד בפרק של join, שם ממש נזדקק לפיצ'ר הזה.

# עמודות מחושבות

לא תמיד הנתונים ששמורים בטבלאות שלנו נותנים לנו את מה שאנחנו רוצים. לדוגמא, נניח יש לנו הזמנות ובין היתר שדה מחיר ההזמנה. ונניח אנחנו רוצים להציג את המחיר פלוס מע"מ (17%). ניתן לעשות זאת באמצעות פעולה מתמטית פשוטה:

SELECT

    order\_id, price \* 1.17

FROM

    orders

בדוגמא זו כפלנו את המחיר ב-1.17 מה שיוסיף לנו לתוצאה 17%.

ישנן עוד הרבה אופרטורים מתמטים שאפשר להשתמש בהם כמו חיבור (+), חיסור (-), חילוק (/). חוץ מהאופרטורים המוכרים יש עוד הרבה אופרטורים. לדוגמא:

* כדי לחשב ערך מוחלט של שדה ניתן לכתוב ABS(field\_name)
* כדי לחשב ערך של שדה בריבוע ניתן לכתוב POWER(field\_name,2)
* כדי לחשב טנגנס של שדה ניתן לכתוב TAN(field\_name) (המורה למתמטיקה שלי ממש גאה בי עכשיו שנתתי את הדוגמא הזאת – תמיד לרשותך בוריס)

ישנן עוד המון פונקציות שימושיות. מומלץ בחום לחקור באינטרנט.

ואם אתם שואלים את עצמכם *"האם אפשר להשתמש ב-alias בעמודה מחושבת?!!!".* אז קודם כל יפה ששאלתם, והתשובה היא כן!

לדוגמא:

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address  
FROM Customers;

אם לא יצא לכם להכיר, הפונקציהCONCAT שמשרשרת לנו מחרוזות ושדות (פירגנתי לכם בפונקציה על חשבון הבית).

# ORDER BY

לפעמים אנחנו נרצה למיין את הפלט של השאילתא שלנו. נעשה זאת באמצעות הפקודה ORDER BY. היא תופיע בסוף השאילתה, לאחר ה-FROM,WHERE וה-JOIN (שעוד לא למדנו).

לדוגמא, אם יש לנו טבלה עם שמות אנחנו יכולים למיין את השאילתה שלנו לפי השם משפחה

ORDER BY LAST\_NAME ASC

הפרמטר ASC הוא קיצור של והוא ascendingיגרום לשאילתה שלנו להתמיין לפי **סדר עולה** על פי שם המשפחה, מ-A ל-Z. כלומר, רשומות שבהן שם המשפחה הוא Affleck יופיו לפני שורות עם שם משפחה Demon.

אם נרצה למיין בסדר יורד, כלומר מ-Z ל-A, נשתמש בפרמטר DESC, מלשון descending.

אם יש לנו ספר טלפונים ואנחנו רוצים למיין לפי שם משפחה, לאחר מכן לפי שם פרטי ואז לפי מספר טלפון, נכתוב:

ORDER BY LAST\_NAME ASC, FIRST\_NAME ASC, PHONE ASC

# Aggregations ו-GROUP BY

לא תמיד הנתונים ששמורים בטבלאות שלנו נותנים לנו את מה שאנחנו רוצים. לדוגמא, הנתון "ממוצע מחירי המוצרים" לא שמור לנו בטבלה. יש לנו טבלה של מוצרים, שבה בין היתר שמור המחיר. כדי לקבל את ממוצע המחירים צריך לבצע חישוב מסויים. לא מדובר בחישוב כמו שעשינו בפרק "שדות מחושבים", בו יצרנו שדה עם ערך חישובי שמחושב **עבור כל שורה,** אלא בחישוב במתבצע על קבוצה של נתונים.

SELECT

     AVG(standard\_cost)

FROM

    products;

הקוד למעלה יחזיר לנו **שורה אחת** והיא תכיל את ממוצע המחירים של כל המוצרים (זאת באמצעות הפונקציה AVG).

אבל, מה עושים אם אנחנו רוצים לקבל את ממוצע המחירים עבור כל קטגוריית מוצרים? לדוגמא, עבור מוצרי חשמל המחיר הממוצע הוא X ועבור אלקטרוניקה הוא Y.

במקרה זה נשתמש בפסוקיץ GROUP BY.

SELECT

    category\_id,

        AVG( list\_price )

FROM

    products

GROUP BY

    category\_id;

לאחר ה-GROUP BY נכתוב את העמודה של הקטגוריה.

**אז מה עשינו פה?**

קיבצנו את הנתונים עבור שדה מסויים. בטבלה המקורית, הנתונים היו שמורים עבור כל **מוצר**, והתוצאה של השליפה שלנו נתנה לנו נתונים ששמורים עבור כל **קטגוריית מוצרים**. **כלומר, הרמה שבה הנתונים שמורים היא שונה.**

**הנתונים ששמורים בטבלה המקורית שמורים ברמה יותר מפורטת. יש לנו נתונים עבור כל מוצר, בעוד בשליפה שלנו הנתונים שמורים עבור כל קטגוריה. הקטגוריה היא רמה פחות מפורטת של הנתונים. לכל מוצר יש קטגוריה וכל קטגוריה מכילה כמה מוצרים, אז כשמסתכלים רק על הקטגוריות מסתכלים על נתונים ברמה פחות מפורטת, ברמה עליונה יותר.**

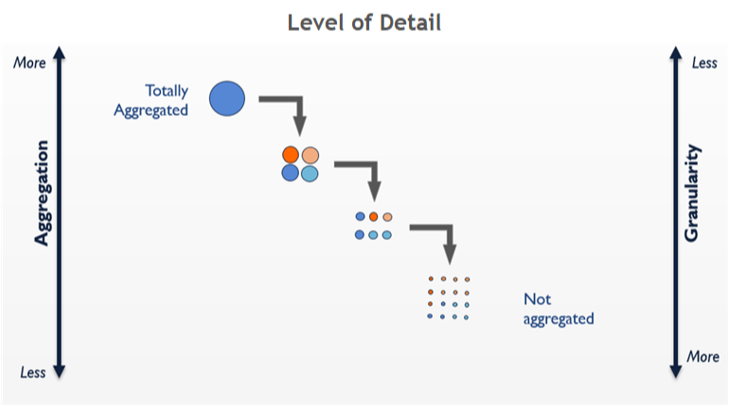
ניקח דוגמא אחרת של תאריכים. ברגע שאנחנו מבצעים פעולות ברמת השנה, אנחנו מבצעים פעולות ברמה גבוהה יותר מאשר אם היינו עושים זאת עבור חודש או יום. הרמה של השנה היא רמה גבוהה יותר מהרמה של החודש (כי השנה כוללת בתוכה חודשים), אך ברמה פחות מפורטת, כי בעוד שיש לנו שנים כמו *2015 ו-2016*, יש לנו את ינואר *2015, פברואר 2015*... והם מפרטים את המידע בצורה יותר טובה. אם מישהו יישאל אתכם מתי נולדתם, ואתם תענו 2015, יש 365 ימים שונים בהם יכולתם להיוולד. אם אתם תענו ינואר 2015 אז יש 31 ימים שונים בהם יכולתם להיוולד. אם תגידו 3 בינואר 2015 אז צמצמתם את הנתון ליום אחד ויחיד. הרמה של השנה היא הרמה הגבוהה ביותר, אך הפחות מפורטת, בעוד הרמה של היום היא הרמה הנמוכה ביותר והכי מפורטת.

לרמה הזאת קוראים **רמה גרנולרית** (**granularity level**) - **כלומר רמת פירוט הנתונים. ככל שהרמה עליונה יותר כך אנחנו ברמת גרנולרית נמוכה יותר כי יש פחות נתונים שמפרטים את המידע (כמו בדוגמא של התאריכים ויום ההולדת). ככל שהרמה נמוכה יותר כך הרמה הגרנולרית גבוהה יותר כי יש לנו נתונים יותר מפורטים.**

**אז איך זה מתקשר לפונקציה כמו AVG?**

ברגע שאנחנו מבצעים פעולה כמוAVG ומקבצים לפי שדה מסויים (GROUP BY), אנחנו יורדים ברמה של הנתונים לרמה פחות מפורטת ויותר גבוהה.

לפונקציה שעושה פעולה מסוג זה קוראים **אגריגציה** (**Aggregation**) – **פונקציה שמקבצת מספר שורות לנתון יחיד.** בדוגמא שלנו, לקחנו את כל המחירים **עבור כל** קטגוריה וחישבנו את הממוצע. אז התוצאה שלנו תהיה **מספר אחד ויחיד** עבור כל קטגוריה (קבוצה של נתונים). לנתון אחד ויחיד זה קוראים Summery Value והוא ערך שנוצר על קיבוץ של מספר נתונים.

לכן – נתון לומר שככל שיורדים ברמה הגרנולרית עולים ברמה של האגריגציה, ולהיפך. **ברגע שאנחנו מצבעים אגריגציה על נתונים, אנחנו יורדים ברמה הגרנולרית שלנו.**

**איזה עוד פעולות אפשר לעשות?**

אפשר לעשות ספירה של ערכים (COUNT), ערך מינימלי (MIN), ערך מקסימלי (MAX) ועוד הרבה יותר! מומלץ להסתכל באינטרנט על הפונקציות השונות.

# HAVING

הפסוקיתHAVING מאפשרת לנו לסנן קבוצות של נתונים, אשר קובצו ב-GROUP BY.

**למה אי אפשר לסנן בתוך ה-WHERE? כי הWHERE- מסנן לנו שורות**. אם אנחנו רוצים לסנן קבוצות, אנחנו רוצים לפעמים לבצע סינון **אחרי הקיבוץ של הנתונים לקבוצות**. לכן הפקודה HAVING קיימת.

לדוגמא:

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

אפשר לראות שב-HAVING סיננו את **הקבוצות של המדינות שעבורן מספר הלקוחות קטן מחמש. זה לא משהו שיכולנו לעשות ב-WHERE כי שוב, ה-WHERE מתבצעת ברמת השורה, בעוד אנחנו מעוניינים לסנן מדינות לאחר שקיבצנו את הנתונים לרמת המדינה.**

# Joins

אולי זה יפתיע חלק מכם, אבל בבסיס נתונים יש יותר מטבלה אחת. בד"כ הנתונים ששמורים בטבלאות השונות מקושרים באיזשהו קשר. אם ניקח את הדוגמא מהפרק של יחסים, יש לנו טבלת סרטים, ששומרת סרטים, יש לנו טבלת שחקנים ששומרת שחקנים, ויש לנו טבלה שמקשרת ביניהם, כך שכל שחקן יוכל לשחק בכמה סרטים וכל סרט יוכל להכיל כמה שחקנים. כלומר, יש לנו 3 טבלאות שונות, וברצוננו לקשר אותן. בטבלה שמקשרת בין סרט לשחקן אין לנו את השם של השחקן או הסרט (משהו שעושים בגלל כל מיני שיקולים עיצוביים של מסדי נתונים רלציונים, אבל זה נושא בפני עצמו). אז אם אנחנו רוצים לקבל את שמות כל הסרטים שישראל ישראלי שיחק בהם, נצטרך דרך לקשר בין 3 הטבלאות הללו.

אבל, כדי שהיה לנו יותר נוח, ניקח דוגמא יותר פשוטה כדי להבין את הקונספט של Join - או בעברית צימוד (איכס, אל תשתמשו במילה הזאת בחיים).

נניח יש לו את שתי הטבלאות הללו:

|  |  |
| --- | --- |
| Employees | |
| Name | DepartmentID |
| Rafferty | 31 |
| Jones | 33 |
| Steinberg | 33 |
| Robinson | 34 |
| Smith | 34 |
| John | NULL |

|  |  |
| --- | --- |
| Departments | |
| DepartmentID | DepartmentName |
| 31 | Sales |
| 33 | Engineering |
| 34 | Clerical |
| 35 | Marketing |

הטבלה הראשונה שומרת נתונים על עובדים ובאיזה מחלקה הם עובדים. הטבלה השנייה שומרת את שמות המחלקות. כמובן שאנחנו משתמשים ב-ID ולא רק בשמות בגלל שיקולי עיצוב.

עכשיו, כדי לקבל את **שם** המחלקה שבה כל עובד עובד בה, עלינו לבצע פעולת Join בין שתי הטבלאות.

להלן ה-Syntax:

SELECT \*

FROM employees

INNER JOIN department

ON employee.DepartmentID = department. DepartmentID;

ניתן לראות שהשתמשנו בשתי פקודות חדשות: INNER JOIN ו-ON. הפקודה INNER JOIN מגיעה אחרי הפקודה FROM ואחריה נכתוב את שם הטבלה שאיתה נרצה לבצע JOIN עם הטבלה מה-FROM. במקרה שלנו, אנחנו משתמשים ב-FROM בטבלת העובדים וב-INNER JOIN בטבלת המחלקות.

בפקודת ה-JOIN אנחנו נשים את התנאי של ה-JOIN. בדוגמא שלנו, התנאי הוא שוויון בין ה-ID של המחלקות מהטבלאות השונות.

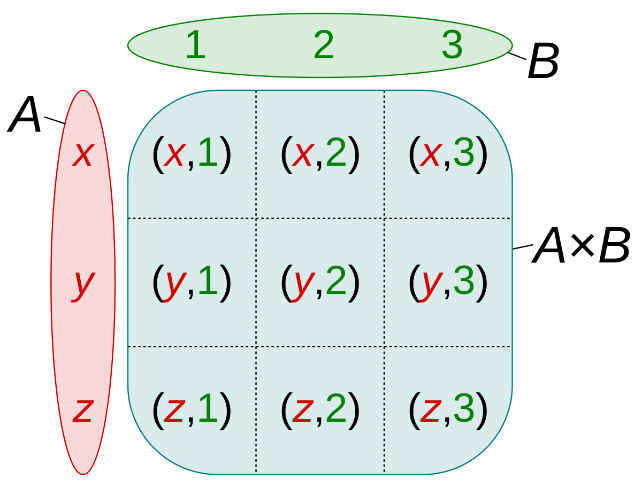
**למה צריך לתת בכלל תנאי?**

אם לא נכתוב תנאי תתבצע פעולה שנקראת **מכפלה קרטזית**. **מכפלה קרטזית מייצרת עבורנו עבור כל שורה בטבלה הראשונה שורות כמספר השורות הטבלה השנייה.** מסובך קצת, נכון. בואו נראה דוגמא:

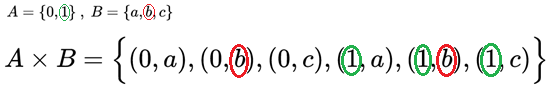
|  |  |  |  |
| --- | --- | --- | --- |
| Name | DepartmentID | DepartmentID | DepartmentName |
| Rafferty | 31 | 31 | Sales |
| Rafferty | 31 | 33 | Engineering |
| Rafferty | 31 | 34 | Clerical |
| Rafferty | 31 | 35 | Marketing |

השורות שלמעלה מציגות לנו מה קורה במכפלה קרטזית **עבור השורה הראשונה בטבלת העובדים (בלבד, כי לא היה לי כוח לעשות את זה לכל הטבלה ☺).** ניתן לראות שיש לנו עבור כל מחלקה בטבלת מחלקות שורה שבה הערך **NAME ו-DepartmentID מטבלת העובדים**. **וזה רק עבור השורה הראשונה. תחשבו איזה טבלה מגעילה נקבל עם היינו עושים מכפלה קרטזית עבור שתי הטבלאות המלאות.**

**כלומר, כמות השורות בטבלת התוצאה תהיה N\*M כאשר N ו-M הם הגודל של שתי הטבלאות.**

**במילים אחרות, תיווצר לנו טבלה שתכיל בתוכה את כל הקומבינציות האפשריות לצימוד של שורות.**

אם נחשוב על כל אחד מ-X,Y,Z,1,2,3 כעל שורה כלשהי (יכולה להכיל כמה שדות) ועל A ו-B כעל הטבלאות, הטבלה שבמרכז היא התוצאה של המכפלה הקרטזית.

בואו נראה דוגמא מעט יותר מתמטית: 

ניתן לראות שהערך 1 מופיע 3 פעמים בתוצאה כי יש 3 ערכים ב-B, ו-b מופיע פעמיים כי ב-A יש 2 שורות. עכשיו תחשבו על כל תו כזה כעל שורה בטבלה, ועל צירוף כמו (0,a) **כאל שורה אחת שמכילה את כל השדות של a ואת כל השדות של 0.**

**למה זה קורה?**

**כשאנחנו עושים JOIN אנחנו צריכים לדעת בין איזה שדות לעשות אותו**. בדוגמא שלנו ה-JOIN הוא בין ה-**ID של המחלקה משתי הטבלאות**. התנאי הזה אומר לשאילתה לאחד שתי שורות משתי הטבלאות לשורה אחת **רק אם** התנאי מתקיים. במקרה שלנו, אנחנו רוצים רק את השורות בהם ה-ID של המחלקה שווה בשתי הטבלאות. זאת מפני שאין לנו צורך בשורה כמו:

|  |  |  |  |
| --- | --- | --- | --- |
| Rafferty | 31 | 33 | Engineering |

Rafferty נמצא במחלקה שה-ID שלה הוא **31** ובשורה הספציפית הזאת השם של המחלקה הוא בכלל של המחלקה שה- ID שלה הוא **33** (בגלל המכפלה הקרטזית). **אין לנו צורך בשורה הזאת, והתנאי שאנחנו שמים ב-ON ימנע ממנה להגיע לטבלת התוצאה שלנו.**

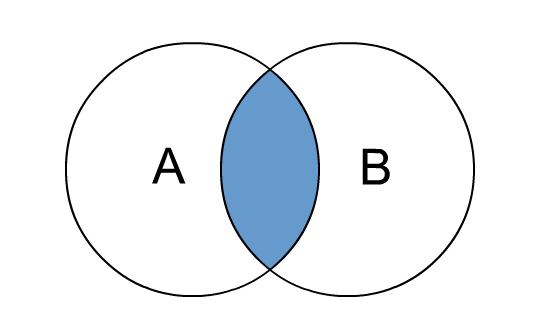
לאחר כל שורות הקוד האלה, נקבל את הפלט הבא:

|  |  |  |  |
| --- | --- | --- | --- |
| Employees.LastName | Employees.DepartmentID | Departments.DepartmentName | Departments.DepartmentID |
| Robinson | 34 | Clerical | 34 |
| Jones | 33 | Engineering | 33 |
| Smith | 34 | Clerical | 34 |
| Steinberg | 33 | Engineering | 33 |
| Rafferty | 31 | Sales | 31 |

לאחר מכן, בתוך ה-SELECT נוכל לבחור במקום בכל השדות רק את השדה של שם העובד ושם המחלקה.

## Inner joins

הדוגמא שהצגנו בפרק הקודם היא דוגמא ל-INNER JOIN.



אם נסתכל על הדוגמא, ב-INNER JOIN אנחנו מקבלים רק את מה שמשותף לשתי הטבלאות, כלומר רק את החלק הכחול. ישנם עוד סוגים של JOIN-ים. בואו נכיר אותם!

## Left join/Left Outer Join

ב-LEFT JOIN אנחנו לוקחים את הרשומות שמקיימים את התנאי שב-ON (מה שנקבל גם ב-INNER JOIN) וגם את שאר הרשומות בטבלה השמאלית. השמאל כאן בא לידי ביטוי בסדר של הטבלאות: הטבלה שרושמים ב-FROM היא השמאלית וזאת שב-LEFT JOIN היא הימנית. להלן דוגמא:

SELECT \*

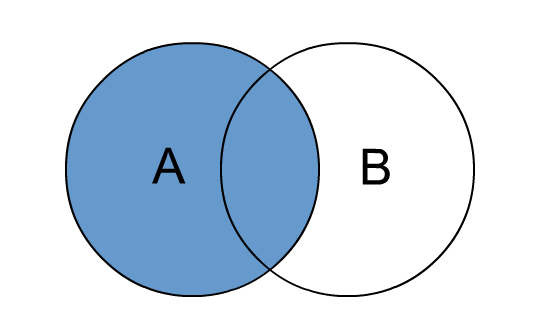
FROM employees

LEFT OUTER JOIN departments

ON employee.DepartmentID = department.DepartmentID;

הפלט של הקוד הזה יהיה:

|  |  |  |  |
| --- | --- | --- | --- |
| Employees.LastName | Employees.DepartmentID | Departments.DepartmentName | Departments.DepartmentID |
| Jones | 33 | Engineering | 33 |
| Rafferty | 31 | Sales | 31 |
| Robinson | 34 | Clerical | 34 |
| Smith | 34 | Clerical | 34 |
| *John* | NULL | NULL | NULL |
| Steinberg | 33 | Engineering | 33 |



חלק מהשורות כאמור לזהות לאלה שנקבל ב-INNER JOIN וחלק לא. השורה שמסומנת בצבע היא השורה שלא קיבלנו בשאילתה של ה-INNER JOIN..

למה לא קיבלנו אותה ב-INNER JOIN? מפני שבטבלה של העובדים ל-John אין ערך בשדה של המחלקה (יש ערך NULL) (תגללו למעלה אם אתם לא מאמינים לי). **בגלל שאין לו ערך התנאי ב-ON שלנו לא מתקיים**, כי בטבלה של המחלקות יש לנו ID של מחלקות, ואין שם למחלקה שלו קיימת.

**אבל, ברגע שהפכנו את ה-JOIN ל-LEFT JOIN אנחנו מקבלים את כל השורות מהטבלה השמאלית, כלומר מטבלת העובדים, ללא קשר אם התנאי ב-ON מתקיים.**

**אפשר לראות שהשאילתה מילאה לנו ערכי NULL בשדות שבטבלה הימנית (מחלקות).**

**חשוב להדגיש ששורות שכן מקיימות את התנאי ב-ON בשתי הטבלאות כן יתאחדו לשורה אחת כמו שקורה ב-INNER JOIN.**

**בנוגע לכותרת- למען הסר ספק: LEFT JOIN = LEFT OUTER JOIN. ה-OUTER נאמר לפעמים כי אכן יש ב-JOIN מסוג זה אלמנט של חיצוניות (לוקחים את כל הצד השמאלי ללא כל תנאי). למעשה, לא חייבים אפילו לכתוב OUTER בקוד שלנו!!!**

## Right join\Right Outer Join

אותו דבר כמו LEFT JOIN **רק הפוך** – לוקחים את כל השורות המשותפות וגם את כל השורות מהטבלה הימנית (זאת שנכתבת לאחר פקודת ה-JOIN).

דוגמא:

SELECT \*

FROM employees

RIGHT OUTER JOIN departments

ON employees.DepartmentID = departments.DepartmentID;

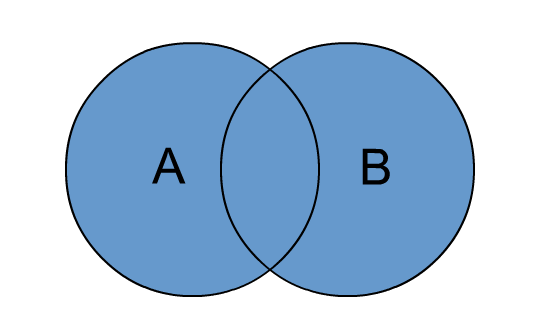
|  |  |  |  |
| --- | --- | --- | --- |
| Employees.LastName | Employees.DepartmentID | Departments.DepartmentName | Departments.DepartmentID |
| Smith | 34 | Clerical | 34 |
| Jones | 33 | Engineering | 33 |
| Robinson | 34 | Clerical | 34 |
| Steinberg | 33 | Engineering | 33 |
| Rafferty | 31 | Sales | 31 |
| NULL | NULL | *Marketing* | *35* |

בשורה החדשה ניתן לראות שיש לנו פרטים על המחלקה Marketing אבל אין לנו פרטים על עובד. **זאת מפני שבטבלת העובדים אין עובד שעובד במחלקה זאת**. אבל, בגלל שאמרנו לשאילתה **לקחת את הכל מהטבלה הימנית (טבלת המחלקות), קיבלנו את השורה הזאת בלי ערכים בשדות של הטבלה השמאלית (עובדים).**

**גם כאן: RIGHT JOIN = RIGHT OUTER JOIN.**

## Full other join

בתוצאה של JOIN מסוג זה נקבל את כל השורות שמקיימים את התנאי + כל השורות מהטבלה השמאלית + כל השורות מהטבלה הימנית, כך שערכי NULL יופיעו בשורות בהן אין התאמה של התנאי של ה-ON.



|  |  |  |  |
| --- | --- | --- | --- |
| Employees.LastName | Employees.DepartmentID | Departments.DepartmentName | Departments.DepartmentID |
| Smith | 34 | Clerical | 34 |
| Jones | 33 | Engineering | 33 |
| Robinson | 34 | Clerical | 34 |
| Steinberg | 33 | Engineering | 33 |
| Rafferty | 31 | Sales | 31 |
| NULL | NULL | *Marketing* | *35* |
| *John* | NULL | NULL | NULL |

## Joins & Aliases

זוכרים את הפיצ'ר המגניב של ה-alias? אז הוא חוזר בענק!!!!

במקרים שיש לנו הרבה תנאים ב-ON לא תמיד יהיה לנו כוח לכתוב כל פעם את שם הטבלה המלא ואז את העמודות.

מה אפשר לעשות?

לתת זהות לטבלאות ב-FROM וב-JOIN בצורה הזאת:

FROM employees e

RIGHT OUTER JOIN departments d

ואז במקום לכתוב:

ON employees.DepartmentID = departments.DepartmentID;

אפשר לכתוב:

ON e.DepartmentID = d.DepartmentID;

תודו לי אחר כך...

## Self Join

כשמו כן הוא, בדובר ב-JOIN שמבצע בין אותה טבלה, כלומר הטבלה מופיעה גם ב-FROM וגם ב-JOIN.

למה בכלל צריך את זה?

הדוגמא הקלסית היא של עובדים ומנהלים: נניח יש לי טבלת עובדים שיש לי בהID של עובד, שם ו-ID של המנהל שלו. המנהל הוא עובד בפני עצמו ולכן הוא גם כן מופיע בטבלה הזאת בעצמו.

לדוגמא:

|  |  |  |
| --- | --- | --- |
| Employee ID | Name | Manager ID |
| 123 | Dave | 321 |
| 321 | Steve | 222 |
| 111 | Chris | 111 |
| 222 | May | NULL |

נניח אנחנו רוצים לשלוף שם של עובד ואת שם המנהל שלו, אז נעשה משהו כזה:

SELECT \*

FROM employees e\_empolyee

INNER JOIN employees e\_manager

ON e\_empolyee.empoyee\_id = e\_manager.Manager\_id;

התנאי ישווה בין ה-ID של העובד לבין ה-ID של המנהל **משתי טבלאות שונות. הן זהות בתוכן שלהן אבל לא מדובר באותה טבלה אלא בשתי מקורות מידע שונים, פעם אחת בשביל העובדים ופעם אחת בשביל המנהלים.**

אם תהיתם לעצמכם, למאי אין מנהל. כנראה היא המנכ"לית.

## Multiples Joins

כמובן שבבסיסי נתונים יש לנו יותר משתי טבלאות, ולכן לפעמים נצטרך לבצע JOIN עבור יותר משתי טבלאות.

במקרים כאלה נכתוב את ה-JOIN-ים אחד אחרי השני. לדוגמא:

SELECT customerName, customercity, customermail, salestotal

FROM onlinecustomers AS oc

INNER JOIN orders AS o

ON oc.customerid = o.customerid

INNER JOIN sales AS s

ON o.orderId = s.orderId;

## WHERE VS. ON

לכעורה כותרת די טיפשית. מה הקשר בין השניים? WHERE מסננת שורות ו-ON מגדיר את התנאי של ה-JOIN.

אבל חכו רגע, יש לזה הסבר טוב.

פסוקית ה-WHERE מסננת שורות **אחרי** שפעולת ה-JOIN קוראת (במידה ויש לנו כזאת). לעומת זאת, התנאי שאנחנו מגדירים ב-ON מגדיר איזה שורות נקבל מפעולת ה-JOIN, כלומר גם היא מסננת שורות **לפני** פעולת ה-JOIN.

**אז אם שתיהן מסננות שורות, אז מה הבעיה? לכעורה כל סינון ששמים ב-ON אפשר לשים גם ב-WHERE.**

ב-INNER JOIN אנחנו **לא** נראה הבדל בין סינון שורות ב-WHERE לבין סינון ב-ON. **איפה שאנחנו כן נראה הבדל יהיה בכל סוג JOIN אחר**.

בואו נראה דוגמא שתמחיש לנו את זה יותר טוב:

נניח יש לנו שתי טבלאות:

ספרים:

|  |  |
| --- | --- |
| **BookName** | **BookID** |
| Book 1 | 1 |
| Book 2 | 2 |
| Book 3 | 3 |
| Book 4 | 4 |
| Book 5 | 5 |

וקנייה של ספרים על ידי לקוחות:

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **BookID** | **IDBuy** |
| A | 1 | 1 |
| B | 1 | 2 |
| A | 2 | 3 |
| C | 2 | 4 |
| D | 3 | 5 |

ויש לנו את השאילתה הבאה שמבצעת **LEFT JOIN** ומסננת לקוח מסוים ב-**WHERE**:

SELECT books.BookName, buys.IDBuy

FROM books

LEFT OUTER JOIN buys

ON books.id = buys.BookID

WHERE buys.CustomerID = 'A'

הפלט של השאילתה **ללא פסוקית ה-WHERE** יהיה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **books .BookID** | **books.BookName** | **Buys.IDBuy** | **Buys.BookID** | **Buys.CustomerID** |
| 1 | Book 1 | 1 | 1 | A |
| 1 | Book 1 | 2 | 1 | B |
| 2 | Book 2 | 3 | 2 | A |
| 2 | Book 2 | 4 | 2 | C |
| 3 | Book 3 | 5 | 3 | D |
| 4 | Book 4 | NULL | NULL |  |
| 5 | Book 5 | NULL | NULL |  |

הפלט הסופי עם ה-WHERE יהיה:

|  |  |
| --- | --- |
| **Buys.CustomerID** | **Books.BookName** |
| A | Book 1 |
| A | Book 2 |

לכעורה נראה טוב. קיבלתנו את הקניות של לקוח A.

בואו נראה את אותה השליפה רק עם סינון של הלקוח ב-**ON**:

SELECT books.BookName, buys.IDBuy

FROM books

LEFT OUTER JOIN buys

ON books.id = buys.BookID

AND buys.CustomerID = 'A'

הפלט של השאילתה יהיה:

|  |  |
| --- | --- |
| **Buys.CustomerID** | **Books.BookName** |
| A | Book 1 |
| A | Book 2 |
| NULL | Book 3 |
| NULL | Book 4 |
| NULL | Book 5 |

**הופה.** מה זה השורות החדשות האלה (מסומנות בצהוב)? כשכתבנו את התנאי של סינון הלקוח ב-WHERE לא קיבלנו את השורות האלה?

**למה?**

בשאילתה השנייה שלנו (הסינון ב-ON) יש עשינו **LEFT JOIN**, כלומר קיבלנו את כל השורות מטבלת הספרים, גם אם לקוח A לא קנה אותם.

**אבל עשינו LEFT JOIN גם בשאילתה הראשונה!**

נכון. אבל **בגלל שעשינו את הסינון על הלקוח ב-WHERE אנחנו מסננים החוצה את השורות עם ערכי ה-NULL!! הרי אלה לא שורות עם הלקוח A.**

**המסקנה: כאשר מסננים תנאים ב-WHERE אנחנו יכולים לאבד שורות ולא לקבל תוצאות של שליפה LEFT"-ית אמיתית" (כלומר לקבל את כל השורות מהטבלה השמאלית גם אם התנאים לא מתקיימים).**

**אז למה ב-INNER JOIN זה לא משנה?**

**כי במילא אנחנו לא מקבלים את השורות עם ה-NULL-ים.**

*-רק תדעו שהיה לי ממש כיף לדבר עם עצמי בפרק הזה. אני צריך לעשות את זה יותר.*

*-מחזק.*

## NON-EQUAL JOIN

אם לא יצא לי להגיד את זה עד עכשיו, **אפשר לעשות JOIN שמקיים תנאים שהם לא =. בפרק הבא נראה דוגמא פרקטית לנושא.**

שיא גינס לפרק הקצר ביותר. וזה לא כי אני עצלן.

## TEMPORAL JOIN

אל תדאגו, TEMPORAL JOIN זה לא עוד סוג חדש של JOIN. מדובר בפעולות JOIN רגילות שאנחנו עושים, רק שנעשים בהקשר של **זמן**.

בואו נתחיל ממושג פשוט.

**TEMPORAL DB**

מדובר בבסיס נתונים ששומר מידע שקשור לזמן.

הרבה מאוד מידע מחיי היום יום שלנו הוא תלויי זמן: מקום עבודה, כתובת מגורים ואפילו השם שלנו. את המידע הזה צריך לשמור בצורה תלוית זמן, כלומר שבכל תאריך שנבחר נוכל לדעת מה השם שלו, איפה אנחנו גרים ואיפה אנחנו עובדים.

את זה אפשר להשיג באמצעות הוספת שתי שדות: תאריך תחילה ותאריך סיום. שני התאריכים האלה מייצגים את טווח הזמנים בהם הרשומה הייתה תקפה. כלומר, בין התאריך הזה והזה יוסי גר בירושלים ולאחר מכן בין התאריך הזה והזה הוא גר בתל אביב.

הדבר נכון בטח ובטח בצבא שאנחנו מתמודדים עם מידע על משאבי אנוש ולוגיסטיקה, כלומר ערימות על גבי ערימות של מידע תלויי זמן.

כמו שאולי אתם יודעים, יש 3 סוגים של מידע תלויי זמן:

* **אילוץ זמן 1** – אין חפיפות בין הזמנים ואין פערי זמנים. לדוגמא, לבן אדם לא יכולים להיות שני שמות פרטיים (אין חפיפת זמנים) ולא יכול להיות מצב שאין לו שם (כלומר שיש זמן כלשהו שאין לו שם – פער זמן בין הפעם הקודמת שהייתה לו שם לפעם הבאה).
* **אילוץ זמן 2** – אין חפיפות זמנים אבל יכולים להיות פערי זמן. לדוגמא, בן אדם יכול להיות בחופשה אחת בנקודת זמן מסויימת (אין חפיפה), אבל יכול להיות מאוד להיות שביום מסויים הוא לא בחופש (צריך להתפרנס).
* **אילוץ זמן 3** – יש חפיפות זמנים ויש פערי זמנים. לדוגמא, לחייל יכולים להיות כמה פטורים בו זמנית (פטור משקל, פטור זקן, פטור שמירות – בקיצור חייל עוקץ), ויכול להיות שביום מסויים אין לו אף פטור פעיל (פטוריים זמניים שעבר זמנם וכד').

טוב לדעת את זה, להמשך החיים.

עכשיו, ברוב המקרים אנחנו נצטרך לעשות JOIN בין שתי טבלאות שהן תלויות זמן. לדוגמא, למצוא את הדרגה של החייל ביום כניסתו ליחידה. יש לנו מידע תלוי זמן של הדרגה של החייל, ויש לנו מידע תלויי זמן על היחידות בהן הוא שירת.

כדי למצוא את הדרגה ביום הכניסה ליחידה, נצטרך למצוא את חיתוך הזמנים הנכון.

נניח אלה הטבלאות שלנו:

טבלת הדרגות:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **FROM** | **TO** | **RANK** |
| 1 | 11.12.2018 | 12.4.2019 | טוראי |
| 1 | 13.4.2019 | 31.12.9999 | רב"ט |

(למי שלא יודע, התאריך 31.21.9999 מציין את "סוף העולם". רושמים זאת כאשר אנחנו לא יודעים מתי הרשומה תפסיק להיות תקפה, כלומר מתי החייל יקבל את הדרגה החדשה ויפסיק, במקרה זה, להיות רב"ט עלוב (כמו עבדכם הנאמן בשעת כתיבת שורות אלו).

טבלת היחידות:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **FROM** | **TO** | **ORGANIZATION** |
| 1 | 11.12.2018 | 14.5.2019 | מצפ"ן |
| 1 | 15.5.2019 | 31.12.9999 | שחר |

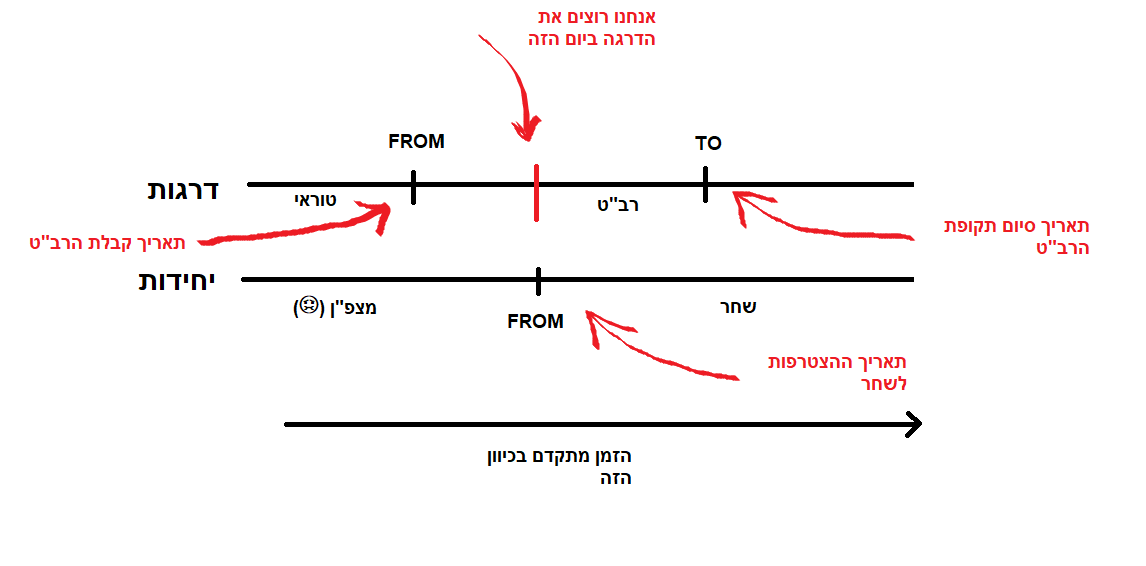
(שימו לב שהחייל עבר מיחידת מצפ"ן לשחר כי הוא הפך מלא מגניב למגניב)

עכשיו, כדי לקבל את הדרגה של החייל ביום הפיכתו למגניב (כלומר הצטרפות ליחידת שחר), נצטרך, מן הסתם, לייצר פה שאילתה עם פעולת JOIN. JOIN זה ייקרא TEMPORAL JOIN.

התנאים שאנחנו צריכים שייתקיימו ב-JOIN הם:

1. ה-ID מטבלת הדרגות = ID מטבלת היחידות (אנחנו רוצים לקבל את הנתונים עבור כל חייל)
2. ה-FROM מטבלת הדרגות **<=** FROM מטבלת היחידות (אנחנו רוצים שהתאריך של קבלת הדרגה יהיה תקף ביום ההצטרפות ליחידה, כלומר **שהדרגה התחילה לפני ההצטרפות ליחידה**).
3. ה-TO מטבלת הדרגות **>=** FROM מטבלת היחידות (אנחנו רוצים שהתאריך הסיום של = הדרגה יהיה תקף ביום ההצטרפות ליחידה, כלומר **שהדרגה** **הסתיימה לפני ההצטרפות ליחידה**).

תנאים 2 ו-3 הם אלה שייתנו לנו את מה שאנחנו רוצים. מה שבעצם אנחנו עושים זה לתחום את התאריך של ההצטרפות ליחידה בין תחומיי הזמן של הדרגות.

אין כמו ציור פשוט להסביר משהו מסובך:

**אפשר לראות שתחום הדרגה של הרב"ט תוחם את תאריך ההצטרפות לשחר (כלומר מכיל בתוכו את התאריך הזה). זה למה אנחנו לא נקבל את הדרגה טוראי, כי היא אמנם התחילה לפני (תנאי מספר 2) אבל הסתיימה לפני (תנאי מספר 3). אנחנו נקבל רק את הדרגה שתקפה בתאריך המסויים שהגדרנו.**

**דוגמא יותר פשוטה היא קבלת הדרגה של חייל בתאריך מסויים, נניח 16.5.2019. כדי למצוא את הדרגה ביום הזה צריך שתאריך קבלת הדרגה יהיה לפני התאריך הזה, ותאריך סיום הדרגה יהיה אחרי היום הזה.**

**ואז, במקום 16.5.2019 אנחנו לוקחים תאריך אחר מטבלה. התאריך יכול להיות יום ההצטרפות ליחידה ויכול להיות כל דבר אחר. בגלל שהתאריך מגיע מטבלה אחרי אז מדובר פה ב-JOIN. אם היינו רוצים את הדרגה ביום 16.5.2019, היינו יכולים לעשות את התנאי בקלות על טבלת הדרגות בלבד.**

**למי שלא שם לב, זו דוגמא מאוד יפה ל-NON EQUAL JOIN מהפרק הקודם שכלל לא התעצלתי לכתוב אותו (אני יודע שאתם חושבים ככה אז די).**

## ציר זמן אחוד (TEMPORAL JOIN מורכב רצח) - הקדמה

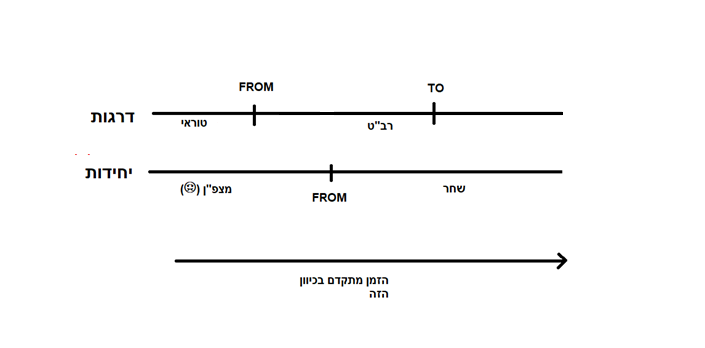
שאלה לי אליכם: אין אנחנו יכולים לאחד את שתי הטבלאות מהפרק הקודם: כלומר את טבלת הדרגות ואת טבלת היחידות?

בואו נחשוב על התוצאה הסופית שלנו ביחד, כלומר על איך הטבלה הסופית שלנו אמורה להיראות:

* אנחנו צריכים שתהיה לנו עמודת **ID**, כדי להבדיל מחייל לחייל.
* אנחנו צריכים עמודה של **דרגה** ועמודה של **יחידה**.
* אנחנו יכולים להוסיף שתי שדות של תאריך תחילה ותאריך סיום (אי אפשר יותר מטווח אחד של זמנים כי רשומה כשלהי תקפה בטווח אחד של זמנים בלבד, מן הסתם).

**עכשיו עולה השאלה: איזה תקופת זמן תאריך התחילה והסיום יציינו? את תקופת הזמן של הדרגה? של היחידה?**

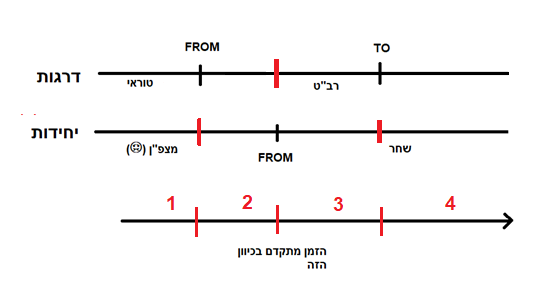
**טווח הזמן הזה לא יכול להיות פשוט טווח זמן מאחד הטבלאות. זאת מפני שבכל טבלה הטווח הזה מציין משהו אחר. ראינו שחייל יכול להצטרף ליחידה חדשה מבלי לקבל דרגה חדשה, כלומר טווח הזמנים מטבלת היחידות לא מכיל לנו את כל טווחי הזמן שרלוונטים לחייל, כמו אלה של הדרגה.**



**תחשבו על זה: אם ניקח את טווחי הזמן מטבלת היחידות, אז לא תהיה לנו דרך לציין מתי החייל קיבל את הדרגה. הוא קיבל אותה בתוך טווח אחר מהטבלה של היחידות, כך שאם פשוט נעתיק את טווחי הזמן מטבלת היחידות נאבד אם המידע על טווח התאריכים של הדרגות.**

אם זה לא היה ממש ברור, קפיצה לפתרון התיאורטי תגרום לכם להבין.

מה שאנחנו צריכים לעשות הוא לייצר שורה עבור כל חיתוך של זמנים:



**אפשר לראות בציור שכל פעם שיש שינוי במידע באחת הטבלאות (כלומר משהו השתנה, דרגה או יחידה) אנחנו נייצר רשומה חדשה עם טווח תאריכים שמתחיל בנקודת זמן הזאת עד התאריך הבא שבו יש שינוי – כל שינוי, יחידה, דרגה וכו'.**

**בציור אפשר לראות שהחייל קיבל דרגה לפני שעבר יחידה, ולכן אנחנו נייצר רשומה (מסומנת ב-1 בציור) עד נקודת זמן הזאת עם הערכים טוראי ומצפן. לאחר מכן, הרשומה ה-2 תהיה עם הדרגה רב"ט והיחידה מצפ"ן, עד שהחייל עבר יחידה. הרשומה הבאה תהיה 3, והיא תכיל את הדרגה רב"ט והיחידה שחר. ככה ממשיכים עד שמקבלים את התוצאה הזאת:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **FROM** | **TO** | **RANK** | **ORGANIZATION** |
| 1 | 11.12.2018 | 12.4.2019 | טוראי | מצפ"ן |
| 1 | 13.4.2019 | 14.5.2019 | רב"ט | מצפ"ן |
| 1 | 15.5.2019 | 31.12.9999 | רב"ט | שחר |

**במג'נטה- תאריך קבלת הרב"ט**

**בציאן – תאריך המעבר לשחר**

**בירוק – תאריך המעבר לשחר פחות אחד (כי הוא היה במצפן עד יום לפני המעבר לשחר)**

**בנזלת – תאריך קבלת הרב"ט פחות אחד (כי הוא היה טוראי עד יום לפני קבלת הרב"ט)**

היישום של הפתרון הוא מעט יותר מסובך, והוא יצריך מאיתנו דברים שעדיין לא למדנו. לכן, ההמשך לפרק וזה והיישום של הטבלה הזאת יהיה בפרק נפרד, בסוף החפיפה.

אבל עדיין אפשר להסביר באופן תיאורטי איך הפתרון יהיה:

ראשית, אנחנו נאחד את כל התאריכים שמופיעים בטבלאות השונות, ונייצר טבלה עם טווח תאריכים חדש שכל טווח יהיה בין שני תאריכים צמודים זה לזה. זה יבטיח לנו שלכל רשומה עם טווח תאריכים מסויים יש מידע שונה מרשומה אחרת.

לאחר מכן נאחד את טבלת הטווחים החדשה עם הטבלאות האחרות שלנו כדי לבלת את המידע שרלוונטי לכל טווח זמן.

**אגיד כבר עכשיו, הפתרון המוצע הוא פתרון למקרים בהם אין חפיפות זמנים וגם אין פערי זמן בין הרשומות השונות.**

# UNION ALL/UNION

בואו נלמד עכשיו משהו יחסית קליל: פעולת UNION.

**ה-UNION מאפשר לנו לאחד תוצאה של שתי פסוקיות SELECT או יותר לטבלה אחת**.

לדוגמא:

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

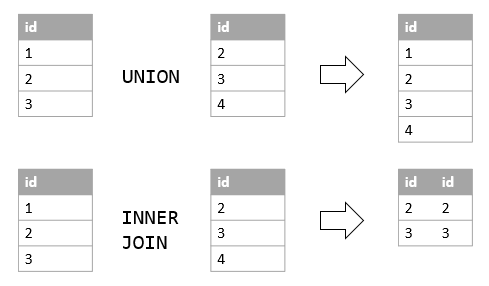
השאילתה הזאת תיקח את כל הערים שמופיעות לנו בטבלת הלקוחות ואת כל הערים שמופיעות לנו בטבלת הספקים ותיתן לנו עמודה אחת עם כלל הערים הללו. **ברגע שיש עיר מסוימת שמופיעה בשתי הטבלאות, אנחנו נקבל רק שורה אחת עם העיר הזאת, כלומר אין כפילויות בערכים.**

אפשר לומר שהפעולה הזאת מאחדת לנו נתונים.

**רגע, אז מה ההבדל בין UNION ל-JOIN?**

מדובר למעשה בשתי פעולות **שונות לחלוטין**.

פעולת ה-JOIN מאחדת לנו **עמודות** משתי טבלאות ועוד פעולת ה-UNION מאחדת לנו **שורות**.



אפשר לראות שה-UNION **פשוט מאחד לנו את השורות לטבלה אחת**, בעוד ה-JOIN מאחד **את העמודות לפי התנאי** (במקרה זה ID = ID) ואנחנו מקבלים שתי עמודות של ID משתי הטבלאות רק עם השורות שמקיימות את התנאי שהגדרנו.

חשוב לשים לב להבדלים האלה. כאמור מדובר בשתי פעולות שונות.

דוגמא לשימוש ב-UNION מהחיים האמתיים: אם יש לנו שתי טבלאות שכל אחת מהן שומרת לנו מכירות בשנה מסויימת, אפשר פשוט לאחד את שתי הטבלאות לטבלה אחת שתכיל את כל המכירות.

חשוב להגיד שאפשר לבצע UNION עם יותר משני שדות.

לדוגמא:

select col1, col2, 'table1' as src

from table1

union all

select col1, col2, 'table2' as src

from table2

הפלט של השאילתה הבאה יהיה:

col1 | col2 | src

t1 | t1 | table1

t2 | t2 | table

שימו לב, שתי שורות מוגדרות ככפילויות אם כלל השדות שלהן שוות (כמו מה שלמדנו ב-DISTICNT).

UNION ALL: מבצעת את אותו הדבר כמו UNION רגיל רק שהיא לא מוחקת כפילויות.

# Sub Query

עכשיו נלמד על משהו ממש מגניב: שליפה בתוך שליפה.

נניח יש לנו טבלה של עובדים ומשכורות. אנחנו רוצים לשלוף את רשימת העובדים שמרוויחים את המשכורת הגבהה ביותר.

אנחנו יודעים למצוא את המשכורת המקסימלית:

SELECT MAX(SALARY)

FROM WORKERS;

לסנן עובדים לפי משכורת מסויימת אנחנו יודעים:

SELECT ID,NAME  
FROM WORKERS  
WHEN SALARY = 25000;

אבל אין אנחנו מחברים בין השניים כדי להגיע למה שאנחנו רוצים?

אפשר במקום לכתוב 25000 לכתוב את התוצאה של השאילתה הראשונה, אבל אז התהליך לא יהיה אוטומטי. ברגע שהנתונים יישתנו והשכר המקסימלי יישתנה, נצטרך לשנות ידנית את השאילתה השנייה.

אז מה עושים? נכניס את השאילתה הראשונה כתת שאילתה של השאילתה השנייה (שיא גינס למשפט עם הכי הרבה "שאילתות" במשפט אחד):

SELECT ID, NAME  
FROM WORKERS  
WHEN SALARY **= (**

**SELECT MAX(SALARY)**

**FROM WORKERS**

**);**

אפשר לראות שהכנסנו את השאילתה הראשונה – תת השאילתא שלו – **לתוך סוגריים** ובמקום לרשום מספר ידנית נשווה את השכר של העובדים לשכר מקסימלי מתת השאילתה ונשלוף את העובדים.

ניתן לשים תת שאילתה גם בפסוקית ה-FROM. דוגמא למקרה כזה הוא מצב שבו אנחנו רוצים לבצע אגריגציה על אגריגציה (או בהגה המקצועית nested aggregation).

נניח אנחנו רוצים לשלוף את הכמות של עובדים בכל מחלקה, ואז לעשות על הכמות הזאת ממוצע – כלומר לשלוך את ממוצע כמות העובדים במחלקות.

אז השאילתה הפנימית שלנו תהיה:

SELECT DEPTPARTMENT\_ID, COUNT(EMPLOYEE\_ID) CNT  
FROM DEPARTMENTS  
GROUP BY DEPARTMENT\_ID;

ואז בשאילתה החיצונית נבצע את הממוצע:

SELECT AVG(COUNTER.CNT)  
FROM **(**

**SELECT DEPTPARTMENT\_ID, COUNT(EMPLOYEE\_ID) CNT  
FROM DEPARTMENTS  
GROUP BY DEPARTMENT\_ID**

**) COUNTER**;

סתם חשוב שתשימו לב שגם כאן אפשר לעשות alias. קראנו לשאילתה הפנימית בשם COUNTER ובתוך הפונקציה AVG התייחסנו לעמודה של הכמות מהטבלה הזאת (במקרה שבו אין לנו יותר מטבלה אחת לא צריך לכתוב את שם הטבלה, נקודה ואז את שם העמודה אלא אפשר לכתוב יש את שם העמודה ולא לעשות alias).

אפשר לעשות תת שאילתה גם בתוך פסוקית ה-SELECT. נניח אנחנו רוצים להציג את ההזמנה והמחיר שלה, ולצד זה להציג עמודה שתכיל את המחיר הממוצע להזמנה.

SELECT OrderID, TotalPrice,

**(**

**SELECT AVG(TotalPrice)**

**FROM Sales**

**)** AS AvgPrice

FROM   Orders;

מן הסתם כל הערכים בעמודה הזאת בכל השורות יהיו זהות, אבל אפשר לקחת את הממוצע הזה ולחסר אותו מהמחיר של ההזמנה ולראות את ההבדל בין המחיר למחיר הממוצע, לדוגמא.

**שימו לב: יש מקרים שבהם ניתן להגיע לאותה תוצאה של שליפה גם בשימוש ב-JOIN וגם בשימוש בתת שאילתה. ברוב המקרים, השימוש ב-JOIN יהיה יותר יעיל מאשר בשימוש בתת שאילתה. אבל, תת שאילתה יכולה בחלק מהמקרים להיות יותר קריאה. אז השימוש באופציה זו או אחרת תלוית מקרה.**

# ALL ו-ANY

ALL ו-ANY הן פקודות שאפשר להשתמש בהן בפסוקית ה-WHERE או בפסוקית ה-HAVING. באמצעותן אפשר לסנן ערכים שמתקבלים מתת שאליתה.

ANY מחזיר לנו אמת אם **לפחות אחד** הערכים בתת שאילתה מקיימים תנאי מסויים. לדוגמא:

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

בשאילתה הזאת אנחנו שולפים את כל המוצרים שבוצעו להם הזמנות עם כמות של 10. התת שאילתה מחזירה את את המוצרים האלה מטבלת ההזמנות, ובשאילתה הראשית אנחנו משווים את ה-ID של המוצר מטבלת המוצרים ל-ID מתת השאילתה, כך שאם ה-ID של המוצר מופיע בתת שאילתא, אנחנו נציג את שם המוצר.

ALL מחזיר לנו אמת אם **כל** הערכים בתת שאילתה מקיימים תנאי מסויים. לדוגמא:

SELECT

    first\_name, last\_name, salary

FROM

    employees

WHERE

    salary > ALL (SELECT

            MAX(salary)

        FROM

            employees

        WHERE

            department\_id = 2)

ORDER BY salary;

במקרה זה אנחנו נשלוף את שמות העובדים והשכר שלהם אם השכר שלהם גבוה מהשכר המקסימלי של העובדים במחלקה מספר 2.

# פקודות DML

כזכור , פקודות DML הן פקודות לניהול הנתונים בטבלאות. עכשיו ניכנס יותר לעומק שלהן.

# INSERT

INSERT היא פקודה שמאפשרת הכנסה של שורות חדשות לטבלה.

לדוגמא:

INSERT INTO Customers (ContactName, Address, City, Country)  
VALUES ('Tom B. Erichsen', 'Skagen 21', 'Stavanger', 'Norway');

כאשר הטבלה מכילה עמודות שם, כתובת, עיר ומדינה. שימו לב שסדר ההכנסה ב-VALUES צריך להיות לפי הסדר שכתוב בסוגריים של העמודות

אפשר גם לכתוב

INSERT INTO Customers   
VALUES ('Tom B. Erichsen', 'Skagen 21', 'Stavanger', 'Norway');

כלומר בלי לכתוב את השדות לאחר שם הטבלה.

אפשר גם להכניס ערכים רק לחלק מהעמודות:

INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');

**במקרה זה חייבים לציין את שמות העמודות.**

# UPDATE

UPDATE היא פקודה שמאפשרת לעדכן ערכים ברשומות שמקיימות תנאי מסויים.

לדוגמא:

UPDATE Customers  
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'  
WHERE CustomerID = 1;

במקרה זה אנחנו מעדכנים את השם והעיר של הלקוח שה-ID שלו הוא 1. אם לא נגדיר WHERE כל הערכים בטבלה שלנו יתעדכנו ולכולם יקראו אפלרד מהעיר בלה בלה.

# DELETE

DELETE מאפשרת לנו למחוק רשומות מטבלה שמקיימות תנאי מסויים.

לדוגמא:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

במקרה זה נמחק את הלקוח ששמו הוא השם הגרמני הזה שאין לי כוח לרשום את שמו שוב. **אם לא נציין ב-WHERE תנאי אז כל הטבלה שלנו תימחק!**

# DISCLAIMER

**השימוש בפקודות DML מצריכות מאתנו זהירות מקסימלית ומחשבה רבה. אפשר בקלות להרוס את המידע שיש לנו!!**

# Window Functions

פונצקיות מסוג אלה הן פונקציות שמבצעות חישובים על שורות. מדובר באותם סוגי חישובים שפונקציות אגריגציה יכולות לעשות כמו AVG,COUNT וכד'. אבל, בשונה מפונקציות אגריגציה, window functons לא גורמות לצמצום של רשומות לשורה אחת. כלומר, מתבצעת פעולת החישוב האגריגטיבית מבלי לשנות את תוכן השורות הקיימות.

בואו נסתכל על התחביר הבסיסי:

window\_function OVER ( [ PARTITION BY partition\_list ] [ ORDER BY order\_list] )

במקום window\_function נכתוב את הפונקציה, כמו סכום, ממוצע וכדומה.

הפקודה OVER מגדירה את ה"חלון" שעליו הפונקציה תעבוד.

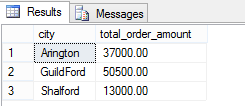
בתוך הפקודה OVER יש את הפקודה PARTITION BY שמגדירה בתכלס על מה הפונקציה תעבוד, כלומר על איזה סט של שורות היא תעבוד. אנחנו נראה בהמשך שהיא ממד דומה ל-GROUP BY.

בואו נראה דוגמא לשאילתה שלא משתמשת בסוג זה של פונקציה:

SELECT city, SUM(order\_amount) total\_order\_amount

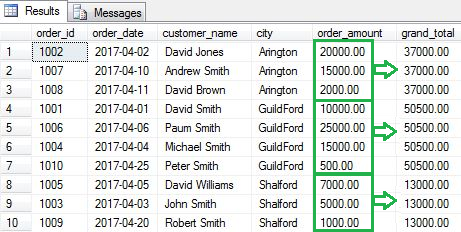
FROM Orders GROUP BY city

השאילתה תחזיר לנו את כמות ההזמנות עבור כל עיר:



אפשר לראות שאנחנו מקבלים שורה עבור כל עיר.

אבל, מה היינו עושים אם היינו רוצים לקבל את הכמות הזאת **מבלי לקבץ את הנתונים לרמת עיר?** כלומר משהו בסגנון:



**אפשר לראות שעדיין יש לנו את השורות של כל ההזמנות, ועדיין יש לנו עמודה של הכמות הכוללת. ניתן לראות שעבור כל עיר, אנחנו מקבלים כמות כוללת שונה.**

אז איך עושים את זה?

SELECT order\_id, order\_date, customer\_name, city, order\_amount

,SUM(order\_amount) OVER(PARTITION BY city) as grand\_total

FROM Orders

ניתן לראות ששלפנו את כל השדות, **בנוסף הוספנו שדה חדש** שיכיל את הערך הכולל **עבור כל עיר**.

במקרה זה, אנחנו רוצים לסכום את כמות ההזמנות, לכן נכתוב SUM(order\_amount).

לאחר מכן אנחנו רוצים שהסכום ייתבצע עבור כל עיר. כלומר, בדומה ל-GROUP BY, אנחנו רוצים לקבץ את הנתונים לרמת עיר ולספור את כמות ההזמנות עבור כל עיר. **אבל כאמור, בשונה מ-GROUP BY, לא מתבצעת פה פעולת קיבוץ. השורות יישארו כמו שהן ורק העמודה החדשה תתווסף.** את ההגדרה של לפי מה "לקבץ" עושים ב-PARTITION BY.

בדומה ל-GROUP BY, גם כאן אפשר "לקבץ" (המילה בסוגריים כי החישוב מבוצע על ידי קיבוץ, אבל הנתונים עצמם **לא** מתקבצים) לפי יותר משדה אחד.



לדוגמא:

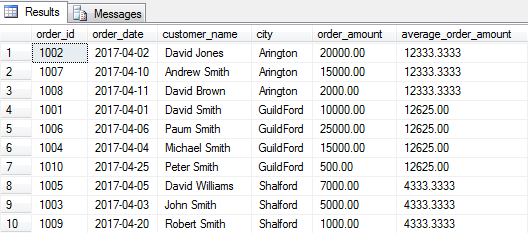
SELECT order\_id, order\_date, customer\_name, city, order\_amount

,AVG(order\_amount) OVER(PARTITION BY city, MONTH(order\_date)) as   average\_order\_amount

FROM Orders

במקרה זה אנחנו רוצים לשלוף את כל הפרטים של ההזמנה, ולחשב את ממוצע כמות ההזמנות לפי עיר וחודש, וזאת שוב, מבלי באמת לקבץ את הנתונים.

התוצאה תהיה משהו כזה:



שוב, ניתן לראות שהשורות התקבלנו כמו שהן, ורק הוספנו עוד עמודה.

קיימות עוד כל מיני פונקציות מגניבות שאפשר להשתמש בהן בתחביר הזה. בואו נכיר כמה מהן.

## RANK, DENSE\_RANK

הפונקציה RANK מאפשרת לנו לתת לכל שורה דרגה מסויימת שתציין את דרגת המיון שלה.

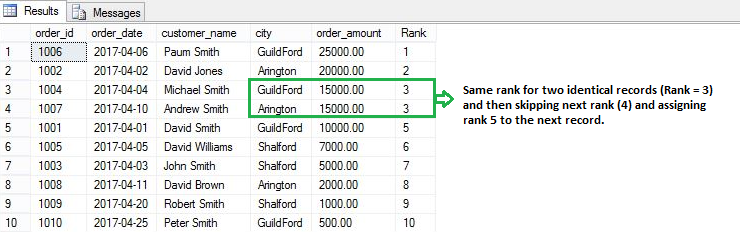
לדוגמא, אם יש לנו נתונים כמו על כמויות, אז לייצר עמודה חדשה שתציין האם מדובר בשורה עם הכמות הגבוהה ביותר, השנייה הגבוהה ביותר וכן הלאה. כלומר – האינדקס של השורה לפי הסדר שהוגדר:

SELECT order\_id,order\_date,customer\_name,city,

RANK() OVER(ORDER BY order\_amount DESC) [Rank]

FROM Orders

אנחנו נקבל את התוצאה הבאה:



ניתן לראות אנחנו מקבלים עמודה שמכילה את האינדקס של השורה בהתאם לכמות שלה. השורה עם הכמות 25000 היא עם האינדקס 1 כי 25000 זה הכמות הכי גדולה שהוזמנה.

ניתן לראות שכאשר מספר ההזמנות הוא שווה, **שתי שורות מקבלות את אותו אינדקס. יותר מזה: אפשר לראות שהשורה הבאה היא 5, כלומר בגלל שיש לנו שני 3, ישר מדלגים לחמש.  
אם אנחנו לא רוצים שהדירוג ידלג מתי שיש כפילויות, אפשר להשתמש בפונקציה DENSE\_RANK, שהיא כמו RANK רק שהיא לא מדלגת על אינדקסים גם אם יש כפילויות בין דירוגים של שורות.**

## ROW\_NUMBER

הפונקציה ROW\_NUMER מייצרת לנו אינדקס לפי מספר השורה.

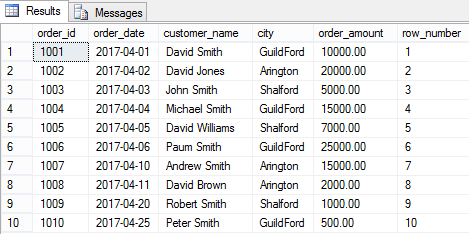
לדוגמא:

SELECT order\_id,order\_date,customer\_name,city, order\_amount,

ROW\_NUMBER() OVER(ORDER BY order\_id) [row\_number]

FROM Orders

הפלט יהיה:



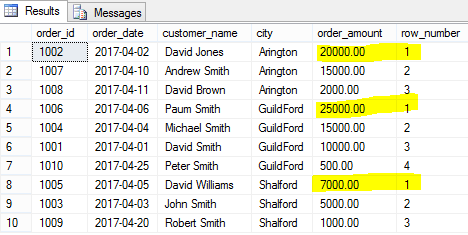
**שימו לב שלא נעשה שימוש ב-PARTITION BY. אם היינו משתמשים בזה, השאילתה הייתה נראית כך:**

SELECT order\_id,order\_date,customer\_name,city, order\_amount,

ROW\_NUMBER() OVER(PARTITION BY city ORDER BY order\_amount DESC) [row\_number]

FROM Orders

והתוצאה הייתה נראית כך:



**ניתן לראות שבגלל שעשינו PARTITION BY לעיר המספור שורות יהיה עבור כל עיר, וכאשר עוברים לעיר הבאה המספור מתאפס.**

## NTILE

הפונקציה NTILE מאפשרת לנו לחלק את המידע לקבוצות ולתת אינדקס לכל שורה שמסמנת את הקבוצה שהיא שייכת אליה.

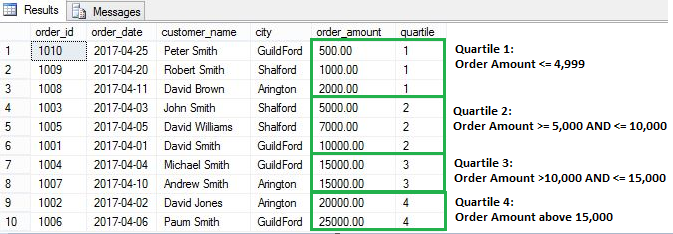
לדוגמא, אם אנחנו רוצים לחלק את המכירות ל-4 קבוצות (באנלית נהוג לקרוא להם Buckets) לפי כמות המכירות:

SELECT order\_id,order\_date,customer\_name,city, order\_amount,

NTILE(4) OVER(ORDER BY order\_amount) [row\_number]

FROM Orders

הפלט יהיה:



ניתן לראות שהערכים של העמודה החדשה הינם בין 1 ל-4 כך שכל שורה מסווגת לקבוצה אליה היא שייכת לפי כמות המכירות שלה.

חדי העין מבינכם שמו לב שיש 3 שורות עבור בקבוצה הראשונה והשנייה ו-2 עבור השאר. זאת מפני שיש לנו 10 שורות ב-DB, וביקשנו לחלק ל-4 קבוצות, כך שלפי מתמטיקה של כיתה ד', בכל קבוצה אמורה להיות 2.5 שורות. בגלל שזה לא אפשרי, הפקודה מחלקת את השורות בצורה כזו שמכניסים את השורות העודפות בקבוצות הראשונות.

## LAG ו-LEAD

הפונקציה LAG נותנת לנו גישה לשורות אחרות ב- DBשמופיעות לפני השורה שלנו. כלומר, השימוש ב-LAG בכל שורה מאפשר לנו לגשת לשורות במופיעות לפניה לפי סדר מסויים.

לדוגמא:

SELECT

    employee\_id,

    fiscal\_year,

    salary,

    LAG(salary) OVER (

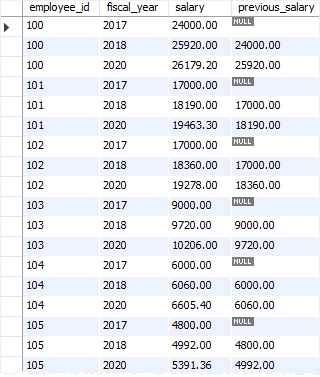
        PARTITION BY employee\_id

        ORDER BY fiscal\_year) previous\_salary

FROM

    basic\_pays;

התוצאה של השליפה הבאה תהיה הוספת עמודה חדשה שתכיל את השכר של כל עובד **בשנה שעברה , זאת על ידי קיבוץ הנתונים לפני עובדים ומיון על פי שנה. פונקציית ה-LAG לוקחת עבור כל employee\_id את השכר של השנה שלפני השנה של השורה הנוכחית.**



**ניתן לראות שעבור כל עובד, אנחנו מקבלים בעמודה החדשה את השכר של השנה הקודמת.**

**ניתן לראות שיש ערכי NULL בעמודה, וזאת משום שלשורה הראשונה עבור כל עובד, כלומר השנה הראשונה שלו בחברה, אין שכר של שנה שעברה כי הוא לא היה בחברה בכלל.**

**אם היינו רוצים את השכר של שנתיים אחורה, היינו רושמים במקום:**

LAG(salary)

את:

LAG(salary,2)

כלומר הפונקציה תיקח 2 שורות אחורה, ולא שורה אחת (שזה ברירת המחדל אם לא רושמים את המספר).

אם בנוסף לזה נכתוב:

LAG(salary,2, 500)

אז במקום ערכי ה-NULL שיופיעו בשליפה יופיע המספר 500.

**שימו לב: לא ניתן לתת מספר שלילי בכמות השורות של "קפיצה לאחור".**

**אז אם אתם שואלים את עצמכם: אין אני קופץ קדימה?**

את זה עושים באמצעות פונקציית LEAD. היא פועלת בדיוק אותו דבר רק שהיא קופתת קדימה במקום אחורה. גם כאן, אי אפשר לתת מספר קפיצות שלילי, אז אל תנסו בכלל סקרנים שכמותכם.

**פונקציות אלה שימושיות במיוחד בנתונים תלויי זמן. אם לדוגמא יש לנו נתונים על מקומות עבודה של אנשים, אפשר עבור כל אדם לראות מה היה תאריך סיום העסקה הקודם שלו, וכך לחשב כמה זמן כל אדם היה בין עבודות (ההפרש בין תאריך העסקה הנוכחי לתאריך סיום ההעסקה הקודם שלו).**

# CTE

CTE (Common Table Expression) היא אופציה שמאפשרת לנו להגדיר טבלאות זמניות בשביל שימוש בפסוקיות כמו SELECT, INSERT, UPDATE וכו'.

בואו נקפוץ ישר לדוגמא:

WITH cte\_sales\_amounts (staff, sales, year) AS (

-- CODE OF CTE HERE

)

SELECT

    staff,

    sales

FROM

    cte\_sales\_amounts

WHERE

    year = 2018;

המילה WITH מגדירה לנו שמדובר פה בטבלה זמנית. לאחר מכן אנחנו נותנים שם לטבלה הזמנית ואת רשימת העמודות. לאחר מכן רושמים AS ואז בתוך **סוגריים עגולים** רושמים את השליפה עצמה שהתוצאה שלה תשמר בטבלה הזמנית שלנו.

לאחר מכן אפשר לראות שמופיעה לנו שליפה באמצעות SELECT. **אנחנו נוכל להשתמש בטבלה הזמנית שלנו רק בשליפה הזאת ולא בשום שליפה/אופרציה אחרת.** אפשר לראות שקראנו לטבלה הזאת בפסוקית ה-FROM שלנו.

אפשר לאחר הסוף של הסוגריים העגולים, לכתוב פסיק ולהגדיר עוד טבלה זמנית. כלומר, כל הטבלאות הזמנית ייכתבו בפסוקית ה-WITH שלנו.

**רגע, אז מה זה שונה מ-SUBQUERY?**

אז זהו, שכשמשתמשים בזה כמו שראינו עכשיו, אין באמת הבדל. במקום להסתבך עם הטבלה הזמנית הזאת אפשר להכניס אותה לסוגרים עגולים בפסוקית ה-FROM.

אבל, הכוח של הטבלה הזמנית הזאת מגיע בשתי סיטואציות:

* סיטואציות בהן אנחנו רוצים להשתמש בתת שליפה יותר מפעם אחת
* שליפה רקורסיבית

הסעיף הראשון הוא די ברור: אם אנחנו רוצים להשתמש בתת שאילתה יותר מפעם אחת בשליפה.

הסעיף השני הוא כבר יותר מסובך, אבל הרבה יותר מגניב.

בואו נראה דוגמא פשוטה, ואז אחריה דוגמא פרקטית.

WITH cte\_numbers(n, weekday)

AS (

    SELECT

        0,

        DATENAME(DW, 0)

    UNION ALL

    SELECT

        n + 1,

        DATENAME(DW, n + 1)

    FROM

**cte\_numbers**

    WHERE n < 6

)

SELECT

    weekday

FROM

    cte\_numbers;

אפשר לראות שבפסוקית ה-FROM שלאחת הגדרת ה-CTE אנחנו שולפים פשוט עמודה מהטבלה הזאת. החלק המעניין מגיע בתוך ה-CTE.

החלק הציאני פשוט שולף לנו שתי עמודות (**שהשם שלהן הוא n ו-weekday ; חשוב להמשך**) עם ערכים 0 והשם שם היום עם האינדקס 0, שהוא Monday או Sunday (תלוי בהגדרות ה-DB עצמו. אצלנו תחילת השבוע הוא יום ראשון ואצל הנוצרים הוא יום שני). הפונקציה DATENAME מקבלת בהתחלה פרמטר DW שאומר DayWeek, שמגדיר לנו את מרווח הזמן לימות השבוע (יכול להיות גם שנים, רבעונים וכו').

לאחר מכן אנחנו מאחדים באמצעות UNION ALL את השליפה הציאנידית עם השליפה הירוקה – **פעולה שתתבצע לאחר שכל השורות שלנו יווצרו**.

**ניתן להגיד שהשורה הראשונה היא שורת הבסיס שלנו, כלומר זאת שורה במתבצעת פעם אחת ואין בה אלמנט רקורסיבי כלשהו.**

**החלק הירוק שולף לנו מידע מתוך הטבלה הזמנית עצמה,** כפי שאפשר לראות בפסוקית ה-FROM**.** בתוכה, אנחנו **שולפים את הערך של העמודה n ומוסיפים לו אחד ואת השם של היום בשבוע עם האינדקס n+1. וכל זאת קורה כל עוד n קטן מ-6.**

**אז בואו נראה מה קורה בריצה הראשונה שלנו. בריצה הראשונה שלנו הטבלה נראית כך:**

|  |  |
| --- | --- |
| **n** | **weekday** |
| 0 | Monday |

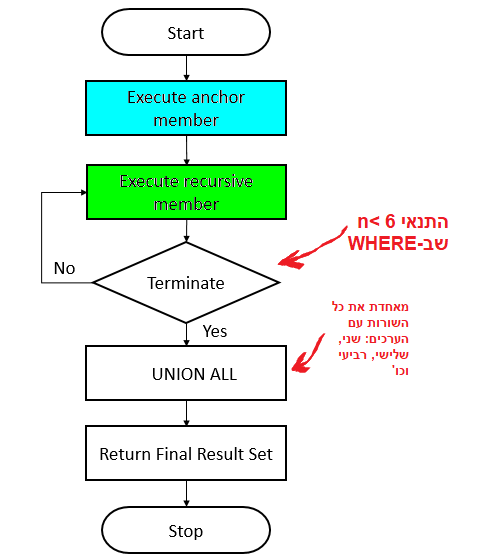
זאת התוצאה של ה-SELECT בצבע הציאנידי**.**

**השליפה הזאת שולפת מידע מהטבלה הזמנית, שבה כרגע יש שורה אחת עם הערכים 0 ו-Monday. השליפה הזאת לוקחת את הערך ב-n ומוסיפה לו אחד, ומחשבת את היום בשבוע של האינדקס הזה. השליפה הזאת תכיל את הערך 1 ואת Tuesday . בגלל שאנחנו שולפים את זה מתוך הטבלה עצמה אנחנו נכנסים בתוך השליפה הזאת לתוך שליפה נוספת. השליפה הזאת תיקח את ה-n מהשליפה הקודמת (כלומר 1) ותחשב את היום עבור 1+1. וכך שוב ושוב ושוב.**

**מה שיש לנו פה בעצם היא שליפה בתוך שליפה בתוך שליפה, שכל שליפה לוקחת את הערך n של השליפה שקדמה לה, מוסיפה 1 ומחשב את היום הבא.** (שיא גינס למשפט עם הכי הרבה "שליפות" במשפט אחד.

**מה מונע מהלוגיקה הזאת להימשך לנצח? כלומר, יש לנו פה מצב שהשליפה קוראת מעצמה וזה יכול להגיע לתהליך אינסופי.**

התנאי WHERE n < 6 הוא זה שיימנע מאתנו לגרום למחשב שלנו להתחרפן (בלעדי התנאי מדובר בלולאה אינסופית). אנחנו עוצרים את החגיגה מתי ש-n הופך להיות שווה ל-6 וכך **בסוף אנחנו מאחדים את כל השורות ביחד לטבלה אחת באמצעות UNION ALL.**



**עכשיו הדוגמא הפרקטית:**

זוכרים את הדוגמה עם הטבלה של העובדים, שבה יש לנו עמודה למנהל של העובד, ואז באמצעות SELF JOIN שלפנו את שם העובד ושם המנהל זה לצד זה?

אז אפשר לעשות את אותו דבר, והרבה יותר מזה, באמצעות שליפה רקורסיבית בטבלה זמנית.

נניח זאת הטבלה שלנו:

|  |  |  |
| --- | --- | --- |
| EmployeeId | EmployeeName | ManagerId |
| 25 | Salman | NULL |
| 26 | Ranbeer | 25 |
| 27 | Hrithik | 25 |
| 28 | Aamir | 27 |
| 29 | Shahid | 28 |
| 30 | Sidharth | NULL |
| 31 | Varun | 30 |
| 32 | Kabeer | 30 |
| 33 | Raj | 29 |

וזאת השאילתה הרקורסיבית שלנו:

WITH EMP\_CTE AS

(

SELECT EmployeeId, EmployeeName, ManagerId, '' AS ManagerName, 0 AS EmployeeLevel

FROM tbEmployee

WHERE ManagerId IS NULL

UNION ALL

SELECT T.EmployeeId,T.EmployeeName, T.ManagerId, C.EmployeeName,ManagerName, EmployeeLevel + 1 AS EmployeeLevel

FROM tbEmployee AS T

INNER JOIN **EMP\_CTE**  AS C

ON C.EmployeeId=T.ManagerId

)

SELECT \* FROM EMP\_CTE

אפשר לראות שבשליפת הבסיס שלנו אנחנו שולפים את כל אלה שאין להם מנהל, כלומר שהם המנכ"לים. לכל אחד מהם אנחנו נוסיף עמודה של הרמה בהיררכיה של החברה, ועבור המהלים הרמה היא אפס כי הם המנכ"לים. לאחר מכן אנחנו שולפים בצורה רקורסיבית את שאר העובדים. איך אנחנו עושים את זה?

בכל שלב ברקורסיה אנחנו שולפים את העובדים שהמנהלים שלהם מופיעים כבר ב-CTE. לכל קבוצה כזאת אנחנו ניקח את הרמה של המנהלים שלהם ונוסיף לזה אחד, כי הם ברמה יותר עמוקה בתוך ההיררכיה של החברה.

**רגע, איפה התנאי של הרקורסיה שמונע ממנה להמשיך לנצח?**

הרקורסיה תפסיק ברגע שלא יהיו לה עוד רשומות לשלוף מטבלת העובדים.

התוצאה של השליפה בסוף תהיה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EmloyeeId | EmployeeName | ManagerId | ManagerName | EmployeeLevel |
| 25 | Salman | NULL |  | 0 |
| 30 | Sidharth | NULL |  | 0 |
| 31 | Varun | 30 | Sidharth | 1 |
| 32 | Kabeer | 30 | Sidharth | 1 |
| 26 | Ranbeer | 25 | Salman | 1 |
| 27 | Hrithik | 25 | Salman | 1 |
| 28 | Aamir | 27 | Hrithik | 2 |
| 29 | Shahid | 28 | Aamir | 3 |
| 33 | Raj | 29 | Shahid | 4 |

# ציר זמן אחוד (TEMPORAL JOIN מורכב רצח) **-** יישום

זוכרים את ציר הזמן האחוד שלנו?

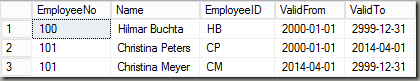
אז הוא חוזר בענק!!

עכשיו כשאנחנו למדנו את כל שאר הדברים – אפשר סוף סוף ליישם אותו.

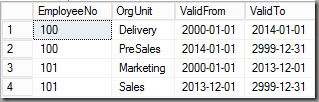
**אזכיר שוב: הפתרון המוצע הוא פתרון למקרים בהם אין חפיפות זמנים וגם אין פערי זמן בין הרשומות השונות.**

בואו נסתכל על דוגמא שיש לי בשבילה צילומי מסך מוכנים כך שלא אצטרך לעבוד כל כך קשה:

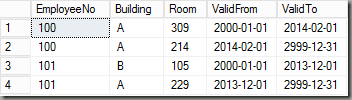
טבלתה העובדים:



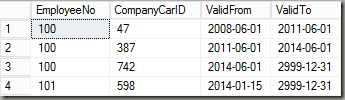
טבלת היחידות:



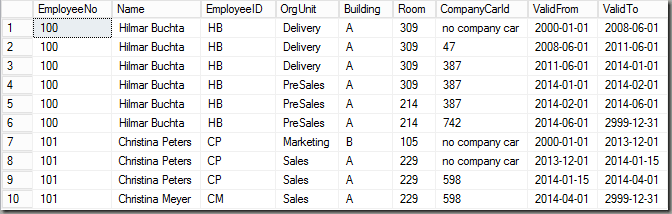
טבלת המיקומים (באיזה חדר כל עובד עובד):



טבלת המכוניות (רכב חברה)



רק אזכיר שהמטרה שלנו הייתה לייצר **שורה עבור כל שינוי שיש לנו במידע**. בואו נראה את התוצאה הסופית הרצויה:



השלב הראשון יהיה לייצר רשימה של כל התאריכים מטווחי התאריכים שלנו:

select EmployeeNo, ValidFrom as Date from Employee

union

select EmployeeNo, ValidTo from Employee

union

select EmployeeNo, ValidFrom from OrgUnit

union

select EmployeeNo, ValidTo from OrgUnit

union

select EmployeeNo, ValidFrom from Location

union

select EmployeeNo, ValidTo from Location

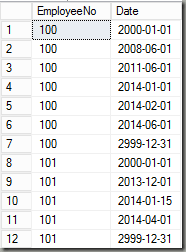
union

select EmployeeNo, ValidFrom from Car

union

select EmployeeNo, ValidTo from Car

השליפה הזאת תיתן לנו את הפלט הבא:



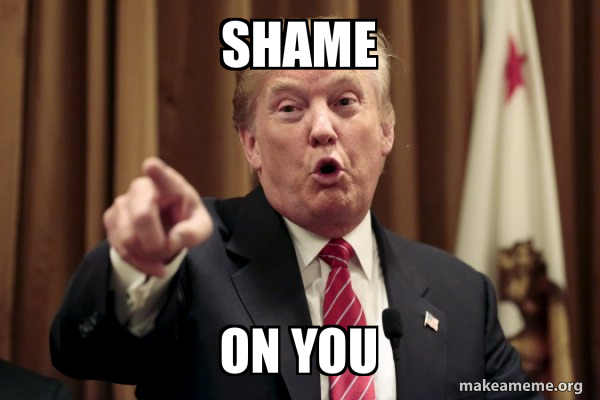
כלומר, לכל עובד יש לנו רשמה של כל התאריכים שמופיעים אצלו בנגררת. **שימו לב שאיחדנו גם את תאריך התחילה וגם את תאריך הסיום. זאת כי כל תאריך סיום הוא תאריך תחילה של הרשומה הבאה.**

**שלושה דברים חשובים:**

1. **אפשר לראות שבדוגמא הזאת "סוף העולם" הוא בשנת 2999 ולא כמו אצלנו (אנחנו יותר מגניבים)**
2. **אצלנו גם אם המידע הוא רציף, כלומר אין פערי זמנים, תאריך הסיום שווה לתאריך התחילה פחות אחד, כלומר יום לפני. לכן, כשאנחנו ניישם את זה לצרכים שלנו אנחנו ניקח את תאריך הסיום ועוד יום במקום את תאריך הסיום המקורי.**
3. **השתמשנו פה ב-UNION ולא בUNION ALL כי אנחנו לא מעוניינים בכפילויות.**

את כל התאריכים האלה אנחנו נכניס לטבלה זמנית, כלומר CTE. **למה טבלה זמנית? כי אנחנו עומדים לייצר עוד טבלאות שישתמשו בטבלה הזאת וזה יהפוך את כל הסיפור להרבה יותר פשוט וקריא.** אבל, עדיין אפשר לעשות את זה ללא כל שימוש ב-CTE, כך שאם אתם רמאים ולא קראתם את הפרק הקודם, הפעם ניצלתם.

רק שתדעו שכתבתי את הפרק עלCTE במיוחד בשביל ציר הזמן האחוד, כדי שיהיה לכם נוח ושלא תהיה לכם בחילה.



with

ValidDates as

(

select EmployeeNo, ValidFrom as Date from Employee

union

select EmployeeNo, ValidTo from Employee

union

select EmployeeNo, ValidFrom from OrgUnit

union

select EmployeeNo, ValidTo from OrgUnit

union

select EmployeeNo, ValidFrom from Location

union

select EmployeeNo, ValidTo from Location

union

select EmployeeNo, ValidFrom from Car

union

select EmployeeNo, ValidTo from Car

)

לאחר שהכנסנו את התאריכים לטבלה הזמנית, **אנחנו צריכים לייצר את טווח הזמנים החדש עבור כל עובד.**

**איך נעשה את זה? באמצעות פונקציית LEAD. באמצעותה נוכל עבור כל שורה לקחת את התאריך הבא וכך להרכיב את טווח התאריכים החדש שלנו (שכאמור בו, כל שינוי יקבל שורה חדשה משל עצמו).**

with

ValidDates as … --THE TABLE WE DID EARLIER

,

ValidDateRanges1 as

(

select EmployeeNo, Date as ValidFrom, lead(Date,1) over (partition by EmployeeNo order by Date) ValidTo

from ValidDates

)

,

ValidDateRanges as

(

select EmployeeNo, ValidFrom, ValidTo from ValidDateRanges1

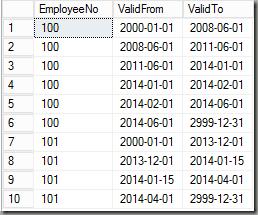
where ValidTo is not null

)

**אפשר לראות שב- ValidDateRanges1 אנחנו לוקחים את התאריך הבא עבור כל עובד במיון לפי התאריכים שלו.**

**ב- ValidDateRangesאנחנו לוקחים את כל התאריכים מ- ValidDateRanges1 שהם לא NULL. זאת מפני שבשורה האחרונה אין לנו תאריך שבה אחריו ולכן אין לנו צורך בשורה הזאת. ואל תתבלבלו, אנחנו עדיין נקבל את סוף העולם. פה אנחנו רק מוצאים שורה מיותרת שאין בה כלל מידע רלוונטי.**

**זאת תהיה התוצאה של הטבלה הזמנית ValidDateRanges:**



**טוב, עכשיו יש לנו את הטווחים המעודכנים. כל מה שנשאר לנו הוא לאחד את המידע עצמו מהטבלאות השונות. עבור כל שורה ניקח את המידע (יחידה, חדר ורכב) שרלוונטי בטווח הזמנים שלה.**

with

--THE TABLES WE DID EARLIER

ValidDates as …

, ValidDateRanges1 as …

, ValidDateRanges as …

select

      E.EmployeeNo

    , E.Name

    , E.EmployeeID

    , isnull(OU.OrgUnit,'unknown') OrgUnit

    , isnull(L.Building,'unknown') Building

    , isnull(L.Room,'unknown') Room

    , isnull(C.CompanyCarId,'no company car') CompanyCarId

    , D.ValidFrom, D.ValidTo

from Employee E

inner join ValidDateRanges D

on E.EmployeeNo=D.EmployeeNo and E.ValidTo>D.ValidFrom and E.ValidFrom<D.ValidTo

left join OrgUnit OU

on OU.EmployeeNo=D.EmployeeNo and OU.ValidTo>D.ValidFrom and OU.ValidFrom<D.ValidTo

left join Location L

on L.EmployeeNo=D.EmployeeNo and L.ValidTo>D.ValidFrom and L.ValidFrom<D.ValidTo

left join Car C

on C.EmployeeNo=D.EmployeeNo and C.ValidTo>D.ValidFrom and C.ValidFrom<D.ValidTo

**מזל טוב!**

יש לכם טבלה עם ציר זמן אחוד.

# תרגול

## סביבת עבודה

כדי לתרגל, אנחנו נשתמש במסד נתונים שנקרא sqlite. הוא ידוע בתור מסד נתונים שקל לעבוד אתו ובמיוחד להתקין אותו, והוא מושלם לצורך התרגול שלנו.

כדי להוריד אותו, תעקבו אחר ההוראות שבקישור הבא:

<https://www.sqlitetutorial.net/download-install-sqlite/>

## נתונים

אנחנו נשתמש בסכמת נתונים שמוצעת גם כן מהאתר sqlitetutorial המגניבים.

תעקבו אחר ההוראות שבקישור:

<https://www.sqlitetutorial.net/sqlite-sample-database/>

מדובר באמת בחמש דקות של התקנה והתחברות למסד הנתונים.

לאחר שסיימתם לעקוב אחר המדריכים יש עוד כמה דברים קטנים שצריך לעשות ולדעת:

* תריצו את הפקודה:

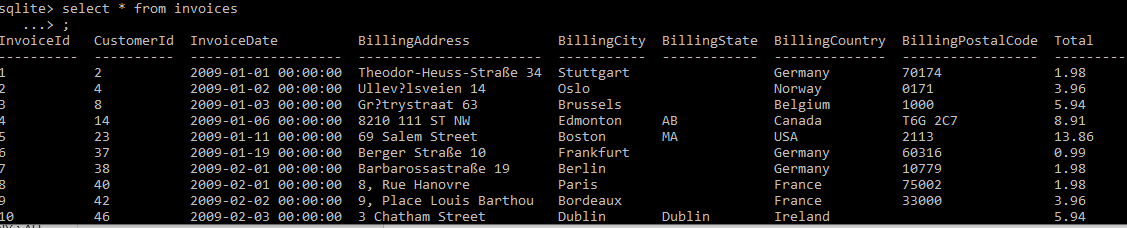
.headers ON

זה יאפשר לנו בזמן שאנחנו שולפים לראות את הכותרות של העמודות שלנו.

* תריצו את הפקודה:

.mode column

הפקודה תפרמט לנו את הנתונים שיוצגו בטבלה יפה כמו זאת:



* אם קורה שאתם רואים שהעמודה נחתכת באמצע, לדוגמא אם שם של מישהו לא מוצג במלואו, אפשר להשתמש בפקודה הבא:

.width 1,100,2

כאשר זה יגדיר את העמודה הראשונה להיות באורך של 1, את השנייה בגודל של 100 וכו'. נשתמש בה תלוי מקרה, כלומר פעם אחת העמודה של השם תהיה ראשונה, ופעם אחרת חמישית.

בכל מקרה, כדי להחזיר למצב ההתחלתי/דיפולטי אפשר לכתוב:

.width auto

* אם אתם רוצים לראות רשימה של עמודות של טבלה מסויימת, תכתבו:

pragma table\_info(invoices);

* אם אתם רוצים לראות רשימה של הטבלאות, תכתבו:

.tables

* אם אתם רוצים יש בקישור <https://sqlite.org/cli.html> עוד כל מיני אפשרויות מתקדמות ונחמדות.
* אם בא לכם לשלוף בטבלה רק n רשומות כמו לדוגמא 5 רשומות תרשמו:

select \* from tracks limit 5;

אני מציין זאת כי מדובר בתחביר שונה משאר מסדי הנתונים (וכי יהיה על זה תרגיל אז תתכוננו)

* אפשר לנקות את המסך (אחרי שהוא יהיה עמוס) עם הפקודה:

.shell cls

* כשהשליפות הופכות להיות מסובכות, אתם יכולים לכתוב את השמירות במקום אחר כמו Word (אני ממליץ על notepad++) ואז להעתיק ולהדביק ל-cmd.
* **אם אתם מסתכלים על הדיאגרמה שבקישורים, שימו לב שהוא לא מציג את כל העמודות שבטבלה אלא רק חלק. בנוסף, בטבלה invoice\_items לעמודות ב-DB יש שמות אחרים ממה שכתוב בדיאגרמה. השתמשו במה שלמדנו כדי לקבל את שמות העמודות הנכונים.**

## תרגילים

לאחר שהצלחתם לעשות את כל זה, נעבור לתרגול עצמו. לכל שאלה יש קובץ CSV עם הפלט המלא של השאילתה.

### 1. בסיס (SELECT, אופרטורים בסיסיים, CASE, LIKE, Aliases, ORDER BY)

1. שלפו את כל תכולת טבלת האומנים.
2. שלפו את רשימת שמות כל האומנים. תשנו את שם העמודה למשהו אחר.
3. שלפו את השם המלא של הלקוח בעמודה אחת, הכתובת המגורים שלו וכתובת המייל שלו.
4. שלפו את העובדים שנולדו בין השנים 1968 ל-1973. אתם יכולים להיעזר בקישור הבא: <https://www.sqlite.org/lang_datefunc.html> אך זה לא חובה לפתרון התרגיל.
5. שלפו את פרטי העובדים שגם שמם הפרטי וגם שם המשפחה שלהם מתחיל באות A.
6. שלפו את רשימת העובדים שמתגוררים ב- Jasper Ave (רמז: השדה של הכתובת)
7. שלפו את פרטי העובדים לפי סדר של שנת העסקתם בסדר עולה.
8. שלפו את רשימת הלקוחות ממוינת לפי מדינה (State של ארה"ב) לפי סדר יורד ולפי מספר טלפון בסדר יורד.
9. שלפו את רשימת השירים שאורכם בין 2 ל-3 דקות (שימו לב שהנתונים שמורים במילי שניות) ממויין לי אורך השיר בסדר עולה.
10. שלפו את פרטי הלקוח רק במקום להציג את המדינה תייצרו עמודה חדשה של יבשת.

### 2. אגריגציות בסיסיות

1. שלפו את מספר הלקוחות בכל עיר.
2. שלפו את מספר החשבוניות עבור כל לקוח.
3. שלפו את מספר השירים בכל פלייליסט
4. ספרו את כמות החשבוניות בכל חודש ושנה (כלומר ינואר 2009, פברואר 2009 וכד').

### JOIN .3

1. שלפו את שמות האלבומים לצד שמות האומנים שהשתתפו באלבום.
2. שלפו את שם השיר לצד שם הז'אנר שלו. אם תשנו את סוג ה-JOIN לסוג JOIN אחר האם יהיה שינוי בתוצאות במקרה זה?
3. שלפו עבור כל לקוח את שמו, ואת שמות השירים שהוא קנה, לצד תאריך הקנייה שלהם ועיר הקנייה.
4. שלפו עבור כל לקוח את שמו, ואת שמות השירים שהוא קנה, לצד ז'אנר השיר ופלייליסטים שבו השיר מופיע. במידה ושיר מופיע ביותר מפלייליסט אחד, תייצרו שורה עבור כל פלייליסט. **שימו לב: יש פלייליסטים עם אותו שם (אבל ID שונה).**
5. שלפו את כלל השירים שבוצעו בכל הזמנה שהאורך של כל השירים הוא בין שתי דקות לחמש דקות. אל תשתמשו בפסוקית ה-WHERE כלל.

### 4. אגריגציות + JOIN

1. שלפו את מספר השירים עבור כל סוג מדיה לצד שם המדיה.
2. שלפו את מספר השירים שכל לקוח קנה עבור כל ז'אנר. תציגו את שם הלקוח, שם הז'אנר ומספר השירים שקנה בז'אנר הזה.
3. הצג את שם הלקוח ואת סוגי המדיה של השירים שהוא קנה ללא כפילות של שורות.
4. הציגו את הלקוחות שלא קנו פריטים לעולם.

### 5. אגריגציות + תת שליפות

1. שלפו את מספר השירים הממוצע בפלייליסטים.
2. שלפו את ממוצע אורך של אלבום (אורך של אלבום = סכום אורכי השירים שלו). האורך צריך להיות בדקות.
3. שלפו את הכמות הממוצעת של שירים שהוזמנו בהזמנות (כלומר, שדהQuantity הוא הכמות לשיר ספציפי. צריך לסכום את הכמות הזאת לכל הפריטים בהזמנה ולעשות על זה ממוצע).
4. שלפו את ממוצע כמות הלקוחות בכל עיר.

### 6. אגריגציות + JOIN + תת שליפות

1. הציגו את שם החברה, ואת ממוצע כמות השירים שכל לקוח (ששייך לחברה מן הסתם) קנה. במידה וללקוח אין חברה, אל תתייחסו אליו.
2. שלפו את ממוצע של "סך כמות המוצרים שקנו פר חברה".

### UNION .7

1. שלפו את כל הכתובות מטבלת העובדים + טבלת הלקוחות. אל תציגו כפילויות.
2. שלפו את כל השמות המלאים של בני האדם ב-DB. אל תציגו כפילויות.
3. איך אפשר לדעת אם יש לנו כפילויות? רמז כאן: <https://www.dofactory.com/sql/subquery>. **התרגיל: שלפו את השורות הכפולות ממשימה 1** (בפרק זה).

### 8. פונקציות WINDOW (ביחד עם כל שאר מה שלמדנו)

1. שלפו את שם המלא של הלקוח ועמודת אינדקס לפי אורך השם לו ללא דילוגים כך שהשם הארוך ביותר יקבל אינדקס 1.
2. שלפו את מספר הלקוחות בכל state. הציגו את שם ה-State, ID הלקוח ומספר הלקוחות ב-State.
3. שלפו עבור כל לקוח את הזמן הממוצע שלו בין קנייה לקנייה. כדי לחשב הפרש בימים השתמשו בקישור הבא:

<https://stackoverflow.com/questions/289680/difference-between-2-dates-in-sqlite>

1. הציגו את שם הלקוח, ID של קנייה, ואת ההפרש בין סכום הקנייה נוכחי לסכום הקנייה הממוצע שלו.