

Angular Learn:

4 - 9	פונקציות בסיסיות	פונקציות מחרוזת פונקציות תאריך פונקציות מספרים פונקציות המרות , Null
10 - 16	אופרטור השוואה בסיסיים	Where אופרטור IN אופרטור Not Between אופרטור Like Is Null / Is Not Null Or / And
17 - 18	מבוא	פעולות בסיסיות מיון תצוגה בסיסי
19 - 28	Join	Inner Join Equi Join Equi Join 3+ Non Equi Join Outer Join Right Outer Join Left Outer Join Full Outer Join Self - Join
29 - 33	פונקציות קבוצה , מיון וקוד קרייא	Group By Having Desc & Asc sort , AS
34 - 42	תת שאלתה	תת שאלתה - מבוא Single Row Subquery Multiple Row Subquery
43 - 48	Views	Views - Into פישוט שאלות מורכבות
49 - 52	Union	Union - Into Union All

53 - 57	פעולות טבלה	Select Distinct Insert Create Update Delete
58 - 63	כלים לשיפור העבודה בSQL	SQL Injection SQL Case Stored Procedure Cloning or Copying a Table SQL Temporary Table Cross Join Operation
64 - 73	התכונות והכליים החדשים בSQL	הגדרת משתנים , if הגדרת לולאה , select Cursor Merge OutPut
74 - 78	Index's	מבוא Create Index DROP INDEX Command יתרונות וחסרונות
	Advanced	
79 - 86	פונקציות לשיפור שאלות וטבלאות	Limit Dates Alter Table Review - Index's Foreign Key Order By Select Distinct 2
87 - 92	tabniot le-pitach matkadem	עדכן מונה בטבלה CRUD With Json Join Pivot חלופות

93 - 101

סקירה – Review

ביטוי רגולרי

Like Wildcard**View**

תת שאלתה

פונקציות מקבצות

Union**Join**

102 - 120

תכנון בסיס נתונים

עקרונות וטיפים לסייע

יחסים בין טבלאות

קשרים בין הישויות

נרטול טבלאות



פונקציות מחרוזת:

RIGHT	הצגת מספר מוגדר של תווים מצד ימין של המחרוזת	<pre>1 SELECT RIGHT('hello' , 2) 2 -- Result: 'lo'</pre>
LEFT	הצגת מספר מוגדר של תווים מצד שמאל של המחרוזת	<pre>1 SELECT LEFT('hello' , 2) 2 -- Result: 'he'</pre>
LEN	הצגת כמה תווים עברו מחרוזת	<pre>1 SELECT LEN('hello') 2 -- Result: 5</pre>
RTRIM	חיטוך רווחים מצד ימין של המחרוזת	<pre>1 SELECT RTRIM(' hello ') 2 -- Result: ' hello'</pre>
LTRIM	חיטוך רווחים מצד שמאל של המחרוזת	<pre>1 SELECT LTRIM(' hello ') 2 -- Result: 'hello'</pre>
REPLACE	החלפת תווים בתוך המחרוזת	<pre>1 SELECT REPLACE('hello' , 'e' , '\$') 2 -- Result: 'h\$ll\$'</pre>
REVERSE	הצגת המחרוזת בסדר הפוך	<pre>1 SELECT REVERSE('hello') 2 -- Result: 'olleh'</pre>
SUBSTRING	חיטוך תווים מחרוזת	<pre>1 SELECT SUBSTRING('hello' , 2,3) 2 -- Result: 'ell'</pre>
LOWER	הצגת מחרוזת באותיות קטנות	<pre>1 SELECT LOWER('HELLO') 2 -- Result: 'hello'</pre>
UPPER	הצגת מחרוזת באותיות גדולות	<pre>1 SELECT UPPER('hello') 2 -- Result: 'HELLO'</pre>

```

SELECT last_name,
       RIGHT(last_name, 2) AS 'rt_last_name' ,
       LOWER(LEFT(last_name, 2)) AS 'lt_last_name' ,
       RIGHT(last_name, 2) + LOWER(LEFT(last_name, 2)) + '@gmail.com' AS 'email',
       LEN(last_name) AS 'ln_last_name',
       REPLACE(main_phone_num, '.', '-') AS 'new_main_p_number',
       SUBSTRING(last_name, 2, 4) AS 'sbstr_last_name'
FROM customers

```

last_name	rt_last_name	lt_last_name	email	ln_last_name	new_main_p_number	sbstr_last
Smith	th	sm	thsm@gmail.com	5	567-867-3945	mith
Jones	es	jo	esjo@gmail.com	5	520-336-8373	ones
Taylor	or	ta	orta@gmail.com	6	522-501-6331	aylo

5

```

SELECT last_name,
       RIGHT(last_name, 2) AS 'rt_last_name' ,
       LOWER(LEFT(last_name, 2)) AS 'lt_last_name' ,
       RIGHT(last_name, 2) + LOWER(LEFT(last_name, 2)) + '@gmail.com' AS 'email' ,
       LEN(last_name) AS 'ln_last_name' ,
       REPLACE(main_phone_num, '.', '-') AS 'new_main_p_number' ,
       SUBSTRING(last_name, 2, 4) AS 'sbstr_last_name'
FROM customers
WHERE LEN(last_name) = 7
  
```

Results							
	last_name	rt_last_name	lt_last_name	email	ln_last_name	new_main_p_number	sbstr_last_name
	Roberts	ts	ro	tsro@gmail.com	7	549-569-1762	ober
	Johnson	on	jo	onjo@gmail.com	7	527-138-3311	ohns
	Jackson	on	ja	onja@gmail.com	7	557-460-8507	acks

CONCAT () –

כשיש לנו **מספר עמודות** שאנו **רוצה להציג ביחד**, במסגרת **עמודה אחת**

במידה **ונרצה להוסיף מש浩ו שיחבר בין העמודות** דוגמת **רחוב או מילת קישור, נוסיף אותם לשאליתה**

כדי לחת שם יפה לעמודה שייצרנו ונשתמש **בAS**

```

SELECT CONCAT(`workers_name`, `workers_city`)
FROM workers
  
```

וכך נראה העמודה המאוחדרת:

CONCAT(`workers_name`, `workers_city`)
MosheNahariya
YekhezkelRamat Hasharon
YirmiyahuBat Yam
GershonBeer-sheva
AsherJerusalem
MetushelahNatanyahu

```

SELECT CONCAT(`workers_name`, " from ", `workers_city`)
AS 'united_column'
FROM workers
  
```

וזו התוצאה:

united_column
Moshe from Nahariya
Yekhezkel from Ramat Hasharon
Yirmiyahu from Bat Yam
Gershon from Beer-sheva
Asher from Jerusalem
Metushelah from Netanyahu

הפונקציה TRIM()

פונקציה מועילה במיוחד היא **TRIM()** אשר מסירה את המרווחים סביבה המחרוזת.

לדוגמה:

```

SELECT TRIM(workers_name) AS name
FROM workers
LIMIT 3;
  
```

בעוד **TRIM()** מסירה את הרוחים משני צידי המחרוזת **LTRIM()** מסירה משמאלי
-**RTRIM()** מסירה מימין.

פונקציות תאריך:

שם הפונקציה	מטרת הפונקציה	דוגמאות
DATEADD	הוספה חלקית לתאריך ימים חדשים שניים	<pre> 1 SELECT DATEADD (mm, 3 , '2010-01-01') 2 -- Result: '2010-04-01' 3 4 SELECT DATEADD (dd , 3 , '2010-01-01') 5 -- Result: '2010-01-04' 6 7 SELECTDATEADD (yy , 3 , '2010-01-01') 8 -- Result: '2013-01-01'</pre>
DATEDIFF	הצגת הפרש בין תאריכים שונים	<pre> 1 SELECT DATEDIFF (yy, '2009-01-01' , '2010-01-01') 2 -- Result: 1 3 4 SELECT DATEDIFF (mm,'2009-01-01' , '2010-01-01') 5 -- Result: 12 6 7 SELECT DATEDIFF (dd,'2009-01-01' , '2010-01-01') 8 -- Result: 365</pre>
DAY	הצגת היום מותך למועד	<pre> 1 SELECT DAY('2010-01-01') 2 -- Result: 1</pre>
MONTH	הצגת החודש מותך למועד	<pre> 1 SELECT MONTH('2010-01-01') 2 -- Result: 1</pre>
YEAR	הצגת שנה מותך למועד	<pre> 1 SELECT YEAR('2010-01-01') 2 -- Result: 2010</pre>
(GETDATE	הצגת התאריך הנוכחי	<pre> 1 SELECT GETDATE() 2 -- Result: (current date)</pre>

לפי השרת



```

-- 2. DATEADD
SELECT GETDATE(),
DATEADD(DAY, 10, GETDATE()),
DATEADD(MONTH, 10, GETDATE()),
DATEADD(YEAR, 10, GETDATE())

```

פונקציות מספריים :

שם הפונקציה	מטרת הפונקציה	דוגמא
FLOOR	עיגול מספר כלפי מטה	<pre> 1 SELECT FLOOR(59.9) 2 -- Result: 59 </pre>
CEILING	עיגול מספר כלפי מעלה	<pre> 1 SELECT CEILING(59.1) 2 -- Result: 60 </pre>
ROUND	עיגול מספר באופן משתנה	<pre> 1 SELECT ROUND(59.9, 0) 2 -- Result: 60 3 4 SELECT ROUND(59.1, 0) 5 -- Result: 59 </pre>

-- 3. ROUND

`SELECT ROUND(34.3,0) -- 0-4
SELECT ROUND(34.6,0) -- 5-9`

ייה עיגול למטה ->

ייה עיגול למעלה ->

כמה ספרות אחרי
נקודה עשרונית ->

The screenshot shows a MySQL command-line interface. The 'results' tab is selected, displaying the output of the following SQL statements:

```

SELECT ROUND(34.3,0) -- 0-4
SELECT ROUND(34.6,0) -- 5-9

```

The results are:

```
(No column name)
34.690
```

פונקציות המרת `Null`

שם הפונקציה	מטרת הפונקציה	דוגמאות
<code>CAST</code>	המרת סוג ערך	<pre> 1 SELECT CAST(getdate() AS varchar) 2 -- Result: Jan 11 2013&amp;amp;amp;amp;nbsp; 4:22PM </pre>
<code>CONVERT</code>	המרת סוג ערך	<pre> 1 SELECT CONVERT(varchar,getdate(),103) 2 -- Result: 11/01/2013 </pre>

שם הפונקציה	מטרת הפונקציה	דוגמאות
<code>ISNULL</code>	המרת ערך NULL בערך אחר	<pre> 1 SELECT ISNULL(NULL,'Somevalue') 2 -- Result: Somevalue </pre>

-- 4. Customers Demo

```

SELECT last_name + ' ' + CAST(join_date AS VARCHAR),
       first_name + ' ' + CONVERT(VARCHAR, monthly_discount)
FROM customers

```

Results

	(No column name)	(No column name)
1	Smith Aug 23 2006 12:00AM	Alvin 74.00
2	Jones Jun 10 2008 12:00AM	Jose 36.00
3	Taylor Dec 9 2005 12:00AM	Amado 73.00

כשנרצה לשרשר בין ערכים של משתנים
מסוגים שונים (למשל מספר ומחרוזת)
נקבל שגיאה لكن נעשה המרה

– עובדת בכל הפלטפורמות, רשות
`Cast`
– קיימת רק ב `server`, מאפשרת
לצין פורמט תצוגה
`Convert`

```

SELECT last_name, pack_id, ISNULL(pack_id , 0), ISNULL(CAST(pack_id AS VARCHAR), 'No Package')
FROM customers

```

– מצפה לקבל ערכים כאלה סוג `isnull`

Results

last_name	pack_id	(No column name)	(No column name)
1 Smith	NULL	0	No Package
2 Jones	NULL	0	No Package
3 Taylor	7	7	7
4 Williams	23	23	23

תנאי השוואה : Where

1 | WHERE שם_עמודה תנאי_לחישוואה ערך

לדוגמא:

```

1 | SELECT *
2 | FROM employees
3 |
4 | WHERE salary = 3000

```

❖ **שם העמודה** – שם העמודה עליה אנו רוצים לבצע את התנינה, בדוגמא זו אנו רוצים להציג את כל אלו אשר שכרם שווה ל- 3000 עד בחרנו את עמודת salary, אם היינו רוצים לבחור את כל אלו שעיר מגוריהם שווה ל 'New York' הימנו בוחרים בעמודת city.

❖ **תנאי השוואה** – קיימים שני סוגי של תנאי השוואה:

- תנאי פשוט – מיוצג ע"י אופרטורים פשוטים כגון: = , < , > , <= , >= .
- תנאי מורכב – מיוצג ע"י אופרטורים מורכבים כגון: IN, BETWEEN, LIKE.

❖ **ערך להשוואה** – אם יכולים לבצע התנינה מול':

מספר – לדוגמא, שכר שווה למספר מסוים.

תאריך – לדוגמא, תאריך העסקה גדול מתאריך מסוים.

מחוזת – למשל, שם משפחה שווה לשם משפחה מסוים.

סוג הערך חייב לתואם לסוג העמודה (לא ניתן לנсотה לבצע את העובדים ששם הפרט גובה מ 4503).

במידה וההשוואה שביקשנו לא תואמת את ערכי הטבלה (לדוגמא, השכר הגובה ביותר בטבלה הוא 12,000 ואמנם

רוצים לבצע את העובדים שਮרויחים יותר מ 20,000) לא תחזיר אלינו אף תוצאה אך גם לא תתקבל שגיאה (ביקשנו מהטבלה ערכים ש מבחינתה לא קיימים).

```

-- * String Value
SELECT customer_id, last_name, first_name, city, monthly_discount
FROM customers
WHERE city = 'Los Angeles'           ↪ לא ניתן לגודל אות

-- * Date Value
SELECT customer_id, last_name, first_name, city, monthly_discount, join_date
FROM customers
WHERE join_date >= '2008-01-01'       ↪ שנה חדש יומן (אפשר מפץ – או /)

```

customer_id	last_name	first_name	city	monthly_discount	join_date
1	Smith	Alvin	New York	28.00	2006-08-23 00:00:00.000

אופרטור IN :

אופרטור IN

אפשר לנו לבצע השוואת עמודה מול מספר ערכים בו זמינים.

```
1 | WHERE ... טריך, טריך) IN שם_עמודה
```

השוואה מול מספרים

או: 80, 50, 80 העובדים אשר מספר מחלקתם שווה ל 90

```
1 | WHERE department_id IN (50, 80, 90)
```

השוואה מול מחרוזות

: כהן או קדם , העובדים אשר שם משפחתם הוא לוי

```
1 | WHERE last_name IN ('Levi', 'Cohen', 'Kedem')
```

- ❖ ערכי המחרוזות חייבים להיות בתוך גרשים ('מחוזת').
- ❖ מבחינת גודל האותיות (LEV vs LEVI) אין צורך בתאימות בין הערך בתוך העמודה לבין הערך שאותו כתבנו בתוך ה IN .

השוואה מול תאריכים

או: 07.09.1982 – 01.01.1990 העובדים אשר תאריך העסקתם הוא

```
1 | WHERE hire_date IN ('1982-09-07', '1990-01-01')
```

! חובה לכתוב את הערך התאריכי בתוך גרשים ("ערך תאריכי") ובאחד מהפורמטים התקינים.

אופרטור NOT :

אופרטור NOT

❖ אופרטור NOT מאפשר לנו להחזיר את התוצאות הפוכות מלאו שאופרטור IN החזיר.

❖ את אופרטור NOT נכתב לפני IN .

או – 80, 50, 80 העובדים אשר מספר מחלקתם שווה ל 90

```
1 | WHERE department_id IN (50, 80, 90)
```

או – 80, 50, 80 העובדים אשר מספר מחלקתם לא שווה ל 90

```
1 | WHERE department_id NOT IN (50, 80, 90)
```

! חייבת להיות תאימות בין סוג העמודה לסוג הערכים בתוך הסוגרים.

```
- SELECT customer_id, last_name, first_name, city, monthly_discount, join_date
  FROM customers
 WHERE join_date IN ('2006-08-23', '2006-03-05', '2010-07-25')
--      * NOT Operator

- SELECT customer_id, last_name, first_name, city, monthly_discount
  FROM customers
 WHERE monthly_discount NOT IN (4,11,13,30)
```

אופרטור : Between

אופרטור BETWEEN

אפשר לנו לבדוק טווח ערכים עבור עמודה מסוימת.

```
1 | WHERE uren_million AND urk_thousand BETWEEN sum_modah
```

בדיקה טווח מול מספרים

– 8000; 5000 העובדים אשר שכרם בטוח בין

```
1 | WHERE salary BETWEEN 5000 AND 8000
```

בדיקה טווח מול מחרוזות

העובדים אשר שם הפרטி בטוח האותיות בין A ל– G (כמו ספר טלפונים) :

```
1 | WHERE first_name BETWEEN 'A' AND 'G'
```

- ❖ ערכי המחרוזות חיבים להיות בתוך גרשים ('מחרוזת').
- ❖ מבחינת גודל האותיות (LEVİ מול LEVI) אין צורך בתאמיות מלאה בין הערך בתוך העמודה לבין הערך שאומנו כתובנו בתוך ה– BETWEEN. למשל, אם כתבנו את טווח האותיות A–G וישנו עובד ששמו הפרטி הוא או, אותו עובד יוצג בתוצאות.

בדיקה טווח מול תאריכים

העובדים אשר תאריך העסקתם נع בין 01.01.1990 ל– 01.01.2000 :

```
1 | WHERE hire_date BETWEEN '1990-01-01' AND '2000-01-01'
```

חובה לנכון את הערך התאריכי בתוך גרשים ("ערך תאריכי") ובאחד הפורמטים התקיימים.

! באופרטור BETWEEN הערך הנמוך תמיד יופיע לפני הערך הגבוה.

! אופרטור BETWEEN כולל ערכים קיצוניים (Inclusive). לדוגמה, כאשר נרצה להציג את העובדים אשר שכרם נע בין 5000 ל– 9000 אנו מקבל את כל העובדים כולל אלו ששכרם שווה ל– 5000 ול– 9000.

אופרטור NOT

- ❖ אפשר לנו להציג את התוצאות הפוכות מלאו לאופרטור BETWEEN החזיר.
- ❖ את אופרטור NOT נכתב לפני BETWEEN.

– 8000; 5000 העובדים אשר שכרם בטוח בין

```
1 | WHERE salary BETWEEN 5000 AND 8000
```

– 8000; 5000 העובדים אשר שכרם לא בטוח בין

```
1 | WHERE salary NOT BETWEEN 5000 AND 8000
```

אופרטור Like

12

אופרטור LIKE

אפשר לנו לאטיר מחרוזות על-סמן תבנית (Pattern) מסוימת.

1 | WHERE last_name LIKE 'תבנית_מסויימת' %

לעתים נרצה לחפש אינפורמציה בטבלה שלא על סמן הערך המדויק של השדה שבתוך העמודה, אלא לפי התבנית, לדוגמה:

- ❖ כל העובדים אשר שם מתחילה ב- 'M' (עמודות שם פרטי).
- ❖ כל הלוקחות אשר הניד שלהם מתחילה ב- 054 (עמודות מספר סלולרי).
- ❖ הרכבים אשר לוחית זהויות שלהם מסתיימת ב- 86 (עמודות מספר לוחית זהות).
- ❖ הודעות בפומם בהן מופיעה המילה 'למכור' (עמודות הודעה).

מבנה התבנית

מבנה LIKE יכול להיות מורכב מהסימנים הבאים:

- ❖ % – מסמל כל מספר של תווים (0 עד אין סוף).
- ❖ _ – מסמל תו בודד.

הדוגמאות הבאות ימחישו את סימני התבנית:

1 | WHERE last_name LIKE '%a'

'%a' – כל השמות אשר מסתיימים באות 'a'. התבנית למעשה אומרת, לא משנה כמה אותיות תהיה בתחילת המילה (כל מספר של תווים), אך התבנית חייבת להסת祢 באות a.

ערכים לדוגמא: 'Akiva', 'Shapira', 'Bouskila'.

באופן תיאורטי גם שם משפחה שערך 'a' היה נכללפה משום ש- % מסמל כל מספר של תווים (גם 0).

1 | WHERE last_name LIKE 'A%'

'A%' – כל השמות אשר מתחילה באות 'a'. התבנית למעשה אומרת, המילה חייבת להתחיל ב- 'a', ולאחר מכן לא משנה מספר האותיות (כל מספר של תווים).

ערכים לדוגמא: 'Avidor', 'Akiva', 'Avraham'.

לשימ לב, בהיעדר הריגשות לגדול אותן גם שם משפחה כגון 'zavidor' יתקבל בחיפוש (החיפוש שלנו מוגדר על 'A' גדולה).

באופן תיאורטי גם שם משפחה שערך 'a' היה נכללפה משום ש- % מסמל כל מספר של תווים (גם 0).

1 | WHERE last_name LIKE '%a%'

'%a%' – כל השמות אשר מכילים את האות 'a'. התבנית למעשה אומרת, המילה יכולה להתחיל בכל מספר של תווים, לאחר מכן צריך להופיע התו 'a' ולאחר מכן שוב יכולים להופיע כל מספר של תווים.

ערכים לדוגמא: 'Harel', 'Gilboa', 'Strauss', 'Avraham'.

באופן תיאורטי גם שם משפחה שערך 'a' היה נכללפה משום ש- % מסמל כל מספר של תווים (גם 0).

1 | WHERE last_name LIKE '_a%'

'_a%' – כל השמות אשר האות 'a' היא האות השנייה בהם. התבנית למעשה אומרת, המילה יכולה להתחיל בתו בודד כלשהו (_ מסמלתו בודד) ולאחר מכן תופיע האות 'a' ולאחר מכן יוכל כל מספר של תווים.

ערכים לדוגמא: 'Dagan', 'Tamari', 'Harel'.

```
SELECT last_name, main_phone_num
FROM customers
WHERE main_phone_num LIKE '__1.6%'
-- 4. NOT Operator
```

last_name	main_phone_num
White	561.654.2679
Wright	541.661.3366

1 | WHERE last_name LIKE '_a%'

'_a%' – כל השמות אשר האות 'a' היא האות השנייה בהם. התבנית למעשה אומרת, המילה יכולה להתחיל בתו בודד כלשהו (_ מסמלתו בודד) ולאחר מכן תופיע האות 'a' ולאחר מכן יוכל כל מספר של תווים.

ערכים לדוגמא: 'Dagan', 'Tamari', 'Harel'.

❖ מוחזיר את התוצאות ההפוכות ממה שאופרטור LIKE החזיר.

❖ כתוב לפני LIKE.

❖ מוחזיר את כל אלו שאינם בשםאות 'a':

```
1 | WHERE last_name LIKE '%a%'
```

❖ מוחזיר את כל אלו שאינם בשםאות 'a':

```
1 | WHERE last_name NOT LIKE '%a%'
```

```

SQLQuery7.sql - D:\LOCEBDF\Ram (34) | SQLQuery8.sql - D:\LOCEBDF\Ram (52) | SQLQuery9.sql - D:\LOCEBDF\Ram (31)
-- General Guidelines
-- * The lower limit must be specified before the upper limit.

SELECT customer_id, last_name, first_name, city, monthly_discount, join_date
FROM customers
WHERE join_date NOT BETWEEN '2006-01-01' AND '2006-12-31'

SELECT customer_id, last_name, first_name, city, monthly_discount, join_date
FROM customers
WHERE join_date BETWEEN '2006-12-31' AND '2006-01-01'
  
```

אופרטור IS Null / IS Not Null

IS NULL / IS NOT NULL

ערך NULL מיין שדה אשר חסר בו ערך. הערך NULL לא שווה לאפס (0) וגם לא שווה לוחץ (' ') יש לראותו כחול ריק. משום שהערך NULL לא שווה לאף ערך, לא ניתן לבצע עלייה השוואות בעזרת האופרטורים '=' או '<>'.

הדוגמאות הבאות לא תקינות.

העובדים אשר אינם מקבלים عمלה:

```
1 | WHERE commission_pct = NULL (לא מקבלים)
```

העובדים אשר מקבלים عمלה:

```
1 | WHERE commission_pct <> NULL ( מקבלים)
```

כדי להתמודד עם השוואה מול ערכי NULL אנו צריכים להשתמש באופרטורים:

= NULL – המקבילה לפועלה IS NULL ❖

<> NULL – המקבילה לפועלה IS NOT NULL ❖

<> <- =!

העובדים אשר אינם מקבלים عمלה:

```
1 | WHERE commission_pct IS NULL
```

העובדים אשר מקבלים عمלה:

```
1 | WHERE commission_pct IS NOT NULL
```

אופרטור : Or / And

האופרטורים AND ו- OR

מטרת אופרטוריים אלו היא לאפשר לנו לבצע סימן על סמך מספר רב של תנאים, לדוגמה :

שם העובד	שכר	מספר מחלקה
חיה	10000	90
דוד	9000	80
יוסי	8000	70
אופיר	7000	70
עמיית	5000	70
רם	4000	40

❖ **AND** מצין כי כל התנאים חיברים להתקיים.

העובדים אשר שכרם גובה מ- 6000 וגם מספר מחלקתם 70.

העובדים אשר עונים על שני תנאים אלו הם: אופיר ויוסי (2 עובדים סה"כ).

❖ **OR** מצין כי לפחות אחד מה坦אים חייב להתקיים.

העובדים אשר שכרם גובה מ- 6000 או מספר מחלקתם 70 :

ו- אצלי יוסי ואופיר שכרם גובה מ- 6000 ומספר מחלקתם 70 – שני התנאים מתקייםים.

ו- אצל חיה ודוד השכרם גובה מ- 6000 – תנאי אחד מתקיים.

5 עובדים עונים לפחות על תנאי אחד.

! משום שאופרטור OR מתקיים גם כאשר רק חלק מה坦אים מתקייםים, אופרטור OR מחזיר יותר ערכים.

העובדים אשר שכרם גובה מ- 6000 וגם מספר מחלקתם 70 :

```

1 | WHERE salary > 6000
2 |
3 | AND
4 |
5 |     Department_id = 70

```

העובדים אשר שכרם גובה מ- 6000 או מספר מחלקתם 70 :

```

1 | WHERE salary > 6000
2 |
3 | OR
4 |
5 |     Department_id = 70

```

! משפט WHERE מופיע פעמי אחת בלבד, לאחר פקודה OR או AND אנו נכתב את התנאי הנוסף ללא ציון נוסף של פקודה WHERE.

סדר קידימות בין תנאים

פקודת AND קודמת לפקודה OR (כשש פעולות הכלפּ קודמת לפעולות החיבור)

```

1 | WHERE Job_Title = 'DBA'
2 |
3 | OR
4 |
5 |     Job_Title = 'Manager' AND Salary > 8000

```

במשפט WHERE זה בחרנו להציג את אלו אשר :

(תנאי שני) התפקיד שליהם הוא Manager **ו גם** הם מרוויחים מעל 8000.

שינוי סדר הקדימות בין תנאים

כasher נרצה לשנות את סדר הקדימות בין OR ל- AND נשתמש בסוגרים (כשם שסוגרים משנים את סדר הקדימות בין חיבור לכפל).

```

1 | WHERE (Job_Title = 'DBA' OR Salary > 8000)
2 |
3 | AND
4 |
5 |     Last_name LIKE '%a%'
```

במשפט WHERE זה בחרנו להציג את אלו אשר :

(תנאי ראשון) התפקיד שליהם הוא DBA **או** השכר שלהם גבוהה מ 8000.

```

SELECT last_name, city, monthly_discount
FROM customers
WHERE city = 'New York' AND monthly_discount > 10
```

(תנאי שני) שם המשפחה שלהם מכיל את האות a.

```

-- 2. OR Operator
-- 4. Order of Precedence
-- 5. Changing the Order of Precedence
```

last_name	city	monthly_discount
Smith	New York	28.00
Hunter	New York	30.00

-- 2. OR Operator

```

SELECT last_name, city, monthly_discount
FROM customers
WHERE city = 'New York' OR monthly_discount > 10
```

-- 4. Order of Precedence

last_name	city	monthly_discount
Smith	New York	28.00
Jones	Los Angeles	12.00
Williams	Houston	17.00

```

SELECT last_name, city, monthly_discount, marital_status
FROM customers
WHERE city = 'New York' OR city = 'Chicago'
AND marital_status = 'Married'

-- 1+2*3
-- (1+2)*3
-- 5. Changing the Order of Precedence

```

Or = +
And = *

results Messages

last_name	city	monthly_discount	marital_status
Smith	New York	28.00	Married
Armstrong	New York	5.00	Divorced
Stone	Chicago	30.00	Married

הו מוגע Where
1 פעם

```

SELECT last_name, city, monthly_discount, marital_status
FROM customers
WHERE (city = 'New York' OR city = 'Chicago' )
AND marital_status = 'Married'

* General Guideline
-- The WHERE keyword appears only once. After the AND or
-- specify the additional condition without specifying

```

results Messages

last_name	city	monthly_discount	marital_status
Smith	New York	28.00	Married
Stone	Chicago	30.00	Married

פונקציות בסיסיות :

```
use Northwind
go
```

תשמש במאסן הנתונים

תבצע את הפקודות הבאות

```
select FirstName, LastName from Employees
```

– תבחר את השדות – **Select**

– מהטבלה – **from**

סימון שורה – איזה פקודות יבוצעו

– כל השדות – *****

```
select FirstName as [First Name], LastName as 'Last Name', City from Employees
```

as – כינוי איך יראה

cotrarת העמודה

First Name	Last Name	City
Nancy	Davolio	Seattle
Andrew	Fuller	Tacoma
Janet	Leverling	Kirkland
Margaret	Peacock	Redmond
Steven	Buchanan	London
Michael	Suyama	London
Robert	King	London
Laura	Callahan	Seattle
Anne	Dodsworth	London

```
SELECT firstName , lastname from Employees
```

אין התוצאות – אין התוצאות לאותיות
גדולות או קטנות

firstName	lastname
Nancy	Davolio
Andrew	Fuller
Janet	Leverling
Margaret	Peacock
Steven	Buchanan
Michael	Suyama
Robert	King
Laura	Callahan
Anne	Dodsworth

```
select FirstName + ' ' + LastName as [Full Name] from Employees
```

– חיבור עמודות – **+**

Full Name
Nancy Davolio
Andrew Fuller
Janet Leverling
Margaret Peacock
Steven Buchanan

```
select ProductName, UnitPrice, UnitPrice * 1.16 as [Price with VAT] from Products
```

– נוכל לעשות פעולות Math
מתמטיות על ערכי עמודות

ProductName	UnitPrice	Price with VAT
Chai	18.00	20.880000
Chang	19.00	22.040000
Aniseed Syrup	10.00	11.600000
Chef Anton's Cajun Seasoning	22.00	25.520000

מיון תצוגה בסיסי :

```
SELECT firstName , lastname from Employees order by LastName
```

```
SELECT firstName , lastname from Employees order by LastName
```

מיון טבלה על פי שדות – Order by

מיון שדה לפי סדר יורד – desc

	firstName	lastname
1	Michael	Suyama
2	Margaret	Peacock
3	Janet	Leverling
4	Robert	King
5	Andrew	Fuller
6	Anne	Dodsworth
7	Nancy	Davolio
8	Laura	Callahan
9	Steven	Buchanan

Results Messages

	firstName	lastname
1	Steven	Buchanan
2	Laura	Callahan
3	Nancy	Davolio
4	Anne	Dodsworth
5	Andrew	Fuller
6	Robert	King
7	Janet	Leverling
8	Margaret	Peacock
9	Michael	Suyama

```
SELECT firstName , lastname from Employees order by LastName desc , FirstName
```

**מיון פנימי – מיון שדה לפי מיון חיצוני של שדה אחר
במקרה של אותו ערך בחיצוני, שם משפחה זהה**

	firstName	lastname
	Michael	Suyama
	Margaret	Peacock
	Janet	Leverling
	Robert	King
	Andrew	Fuller
	Anne	Dodsworth
	Nancy	Davolio
	Laura	Callahan
	Steven	Buchanan

```
select distinct city from Employees
```

**הציג ערכים
פעמיות (יחודית) – distinct**

city
Kirkland
London
Redmond
Seattle
Tacoma

```
select top(3) * from Employees
```

**הציג x שורות
הטבלה – Top(x)**

	EmployeeID	LastName	FirstName	Title
1	1	Davolio	Nancy	Sales Representative
2	2	Fuller	Andrew	Vice President, Sales
3	3	Leverling	Janet	Sales Representative

```
select top(50) percent * from Employees
```

**הציג x אחוז
משורות הטבלה – percent**

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	B
1	Davolio	Nancy	Sales Representative	Ms.	1
2	Fuller	Andrew	Vice President, Sales	Dr.	1
3	Leverling	Janet	Sales Representative	Ms.	1
4	Peacock	Margaret	Sales Representative	Mrs.	1
5	Buchanan	Steven	Sales Manager	Mr.	1

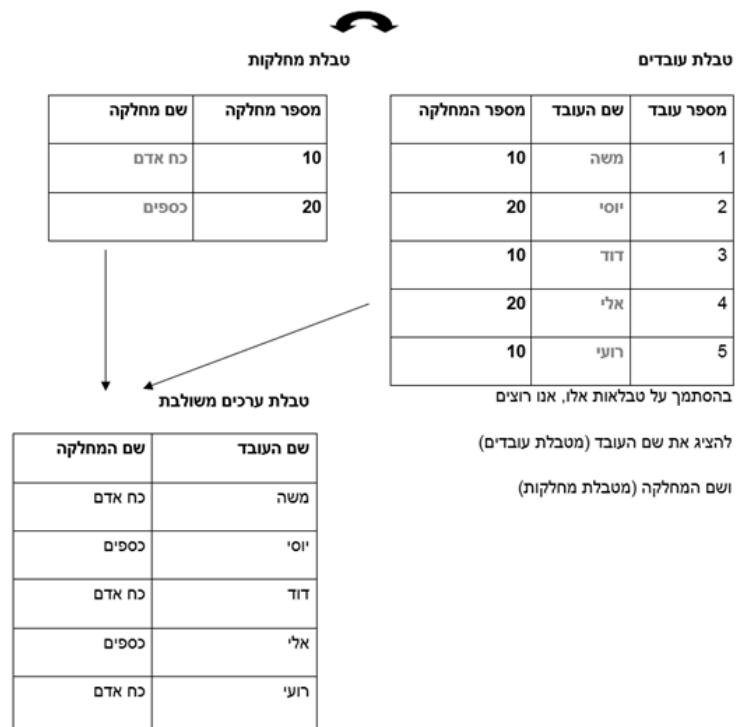
Inner Join:

מדריך SQL | SQL Server – פקודת Join

מדריך SQL זה מתמקד בפקודת `Join` שוחה. מדריך זה הינו חלק מרשימה פוסטיבים הנוגעים לפקודת `JOIN` ב-SQL SERVER. כדי להציג לפוסטיבים נוספים ניתן להעזר בקישורים הבאים:

- **שילוב טבלאות באמצעות Inner** – בשילוב נתונים מטבלאות על סמך ערכים משותפים לשתי הטבלאות.
- **שילוב טבלאות באמצעות Outer** – מתמקד בשילוב נתונים מטבלאות תוך הגזת כל הנתונים מהטבלה האחראית לא קשר לבתונים המוצגים בטבלה האחרת.
- **שילוב טבלאות באמצעות Self Join** – שימוש נתונים מאותו טבלה עצמה.

לעתים, הנתונים אוטם נרצה להציג נמצאים בשתי טבלאות או יותר, לדוגמה:



שני עקרונות חשובים לשילוב בין טבלאות:

❖ על מנת לקשר בין שתי טבלאות או יותר יש צורך בערכים משותפים (מספר מחלוקת בדוגמא זו) או קשר הגיוני מסוים.

❖ על מנת לקשר בין שתי טבלאות יש צורך להחליט על תנאי הקישור. בדוגמא שהוצגה הקשר היה על סמך השוואת מספר המחלוקת בטבלת מחלוקת שווה למספר המחלוקת בטבלת עובדים שירק聯繫 למספרה זו. אם מספר המחלוקת בטבלת מחלוקת שווה למספר המחלוקת בטבלת עובדים הרווק聯繫 למספרה זו.

מכפלה קרטזית – כל הבחירה האפשרי, שילוב כל הקומבינציות האפשרות שתי טבלאות או יותר. מכפלה קרטזית לרבות מתרחשת כתוצאה מפעולות השוואה לא נcona או קשור לוגי שגוי בין העמודות.

Equi Join:

– שילוב 2 טבלאות Equi Join

– מלשון Equal, משומש שהשילוב נעשה על סמך אופרצית השוואה.

```

1 | SELECT شם_עמודה.שם_טבלה_a , شם_עמודה.שם_טבלה_b , ...
2 | FROM شם_טבלה_b JOIN شם_טבלה_a ...
3 |
4 | شם_עמודה.שם_טבלה_a = شם_עמודה.שם_טבלה_b
5 |

```

לדוגמא, אם רוצים לשלב נתונים מטבלת Employees וטבלת Departments, להציג את שם משפחת העובד ומושרכתו מטבלת Employees ולהציג את שם המחלקה של העובד מטבלת Departments :

```

1 | SELECT employees.last_name , employees.salary ,
2 |         departments.department_name
3 |
4 | FROM   employees JOIN departments
5 |
6 | ON   employees.department_id = departments.department_id
7 |

```

- ❖ משפט SELECT מצין את העמודות אותן רוצים להציג לפי הטבלאות השונות. עמודות אלו נכתבו בפורמט של שם_עמודה.שם_טבלה.
- ❖ משפט FROM מצין את הטבלאות מהן אנו רוצים למשוך את הנתונים. במשפט זה אנו מצין את הטבלאות השונות עם מילת המפתח JOIN בינהן.
- ❖ לאחר ציון הטבלאות תגיע המילה ON ולאחריה נציין את התנאי על פי אותו רוצים לבצע את הקישור.

כינויים לטבלאות

כינויים לטבלאות מייעלים את נוחות הכתיבה ואת הקיראות של השאילות JOIN. במקרים לכטוב אחרי כל עמודה ועמודה שם טבלה מלא (דבר שעלול להיות מאוד ארוך ומסורבל) השתמש בכינויים. נבצע שוב את הדוגמא الأخيرة תוך הגדרת כינויים :

```

1 | SELECT emp.last_name , emp.salary , dep.department_name
2 |
3 | FROM   employees emp JOIN departments dep
4 |
5 | ON   emp.department_id = dep.department_id

```

- ❖ את הכינויים לטבלאות נגדיר במשפט FROM.
- ❖ לאחר כל שם טבלה נכתב כינוי.
- ❖ ניתן לנבוע שם שנקבע (למשל, לנכון את טבלת Employees באות A), אך רצוי שלכינויים תהיה משמעות.
- ❖ לאחר שנגדיר כינוי לטבלה, פורמט כתיבת שם העמודה יהיה : שם_עמודה.כינוי_שם_טבלה
- ❖ לאחר שנגדיר את הכינויים לטבלאות, לא יהיה ניתן להשתמש בפורמט העמודה הקודם : שם_עמודה.שם_טבלה
- ❖ הזרה המקובלת לנכונות JOIN בין טבלאות היא תמיד ע"י השימוש בכינויים לטבלאות. מעטה, יתר הדוגמאות תהיאנה עם כינויים לטבלאות.

להלן מספר דוגמאות :

ניתן להוסיף עוד תנאי – להציג את השילוב רק עבור העובדים אשר נמצאים במחלקה מס' 90 :

```

1 | SELECT emp.last_name , emp.salary , dep.department_name
2 |
3 | FROM   employees emp JOIN departments dep
4 |
5 | ON   emp.department_id = dep.department_id
6 |
7 | WHERE emp.department_id = 90

```

ניתן לשנות את סדר המניין – להציג את הנתונים בצורה ממוחינת לפי שכר העובד :

```

1 | SELECT emp.last_name , emp.salary , dep.department_name
2 |
3 | FROM   employees emp JOIN departments dep
4 |
5 | ON   emp.department_id = dep.department_id
6 |
7 | WHERE emp.department_id = 90
8 |
9 | ORDER BY emp.salary DESC

```

Equi Join 3+:

– שילוב 3 טבלאות או יותר

טבלת איזוריים			טבלת מחלקות					טבלת עובדים		
מספר איזור	שם איזור	עיר	מספר איזור	שם מחלקה	מספר איזור	מחלקה	מספר עובד	שם העובד	המחלקה	
600	ת"א		900	כח אדם	10		10	משה		
900	ת"ב	ת"ב	600	כוגים	20		20	יוסי		

מספר עובד	שם העובד	המחלקה
1	משה	
2	יוסי	
3	דוד	
4	אלן	
5	רועי	

כעת נרצה לשלב נתונים שלוש טבלאות: הציג שם העובד מטבלת עובדים, את שם המחלקה שלו מטבלת מחלקות ואת העיר שלו מטבלת איזוריים.

כדי שנוכל לשלב טבלה נוספת לשתי הטבלאות הקשורות שלנו, יש :

❖ להוסיף את שם הטבלה במשפט FROM (ע"י שימוש ב- JOIN נוספת).

❖ למצוא ולכתוב את התנאי המקשר הנוסף במשפט ON.

```

01 | SELECT emp.last_name , emp.salary , dep.department_name ,
02 |
03 | loc.city
04 |
05 | FROM   employees emp JOIN departments dep
06 |
07 | ON emp.department_id = dep.department_id
08 |
09 | JOIN locations loc
10 | ON dep.location_id = loc.location_id

```

❖ אותו רעיון חל גם על שילוב ארבע טבלאות או יותר – הוסף שם הטבלה במשפט JOIN ו כתיבת התנאי המקשר הנוסף במשפט ON.

Non Equi Join:

Non Equi Join

לעתים, נרצה לשילב בין שתי טבלאות אשר אין בינהן עמודה משותפת ולבאורה אף תנאי קשר :

טבלת דרג שכר			טבלת עובדים			
	דרוג שכר	שכר מקסימלי	מספר עובד	שם העובד	שכר	
	A	2000	500	1	משה	1000
	B	4500	2001	2	יօן	2000
	C	6000	4501	3	דוד	3000

מהו דרגת השכר של משה? אמונם אין לנו עמודה משותפת, אך אנו יכולים לדעת כי הוא שייך לדרג שכר A, משום שהשווים את שכרו לנוטונים בטבלת דרג השכר והוא שווה לנמצא בין 500 ל 2000.

כדי לבצע את ההשילוב בין שתי טבלאות אלו נשתמש באופרצית BETWEEN :

```

1 | SELECT emp.last_name , emp.salary , job.grade_level
2 |
3 | FROM employees emp JOIN job_grades job
4 |
5 | ON emp.salary BETWEEN job.lowest_sal AND job.highest_sal

```

שילוב בין כל שתי טבלאות (או יותר) הוא אפשרי כל עוד אנו מוצאים תנאי הגיוני כלשהו אשר יקשר את הטבלאות יחדיו.

	customer_id	last_name	first_name	pack_id
1	1	Smith	Alvin	NULL
2	2	Jones	Jose	NULL
3	3	Taylor	Amado	7
4	4	Williams	Stuart	23
5	5	Brown	Demarcus	13
6	6	Davies	Mark	NULL
7	7	Evans	Merlin	1
8	8	Wilson	Elroy	NULL

	pack_id	speed	start_date	monthly_payment	sector_id
1	1	750Kbps	2005-11-03 00:00:00.000	79	1
2	2	750Kbps	2006-09-04 00:00:00.000	69	1
3	3	750Kbps	2007-01-06 00:00:00.000	59	1
4	4	750Kbps	2008-05-28 00:00:00.000	49	1
5	5	750Kbps	2009-10-10 00:00:00.000	39	1
6	6	750Kbps	2010-01-06 00:00:00.000	29	1
7	7	750Kbps	2005-05-03 00:00:00.000	69	2
8	8	750Kbps	2006-08-19 00:00:00.000	59	2

```
-- 2. Basic JOIN Syntax

SELECT customer_id, last_name, first_name, cus.pack_id, speed
FROM customers JOIN packages
ON customers.pack_id = packages.pack_id

-- 3. Using Table Prefix
-- 4. Using Table Alias
```

Messages

Msg 209, Level 16, State 1, Line 16
Ambiguous column name 'pack_id'.

Ambiguous – כאשר משלבים בין
טבלאות יתכן מצב של **שמות זהים**
בעמודות

```
SELECT cust.customer_id, cust.last_name, cust.first_name, cust.pack_id, pack.speed
FROM customers cust JOIN packages pack
ON cust.pack_id = pack.pack_id
```

Table Alias – כינוי
טבלאות לקיצור,
נستخدم לאורך כל
השאילתת

יותר קרייא, לרשום שם
טבלה לפני כל עמודה

```
SELECT cust.customer_id, cust.last_name, cust.first_name, cust.pack_id, pack.speed
FROM customers cust JOIN packages pack
ON cust.pack_id = pack.pack_id
WHERE pack.pack_id = 7
ORDER BY customer_id
```

customer_id	last_name	first_name	pack_id	speed
3	Taylor	Amado	7	750Kbps
10	Roberts	Rudolph	7	750Kbps
27	Clarke	Bruce	7	750Kbps

```

SELECT pack.speed, AVG(cust.monthly_discount)
FROM customers cust JOIN packages pack
ON cust.pack_id = pack.pack_id
GROUP BY pack.speed

```

Results Messages

speed	(No column name)
1.5Mbps	33.457413
10Mbps	38.056201
12Mbps	38.191191

```

SELECT cust.customer_id, cust.last_name, cust.first_name, cust.pack_id, pack.speed, sec.sector_id
FROM customers cust JOIN packages pack
ON cust.pack_id = pack.pack_id
JOIN sectors sec
ON pack.sector_id = sec.sector_id

```

I

Results Messages

customer_id	last_name	first_name	pack_id	speed	sector_name
3	Taylor	Amado	7	750Kbps	Business
4	Williams	Stuart	23	2.5Mbps	Private
5	Brown	Demarcus	13	1.5Mbps	Private

```

SELECT cust.Last_Name, pack.speed, pack.monthly_payment, sec.sector_name
FROM packages pack JOIN sectors sec
ON pack.sector_id = sec.sector_id
JOIN customers cust
ON pack.pack_id = cust.pack_id
WHERE sec.sector_id = 1
ORDER BY cust.last_name

```

<- On, Join
הרחבה של from

Results Messages

Last_Name	speed	monthly_payment	sector_name
Adams	750Kbps	79	Private
Adams	2.5Mbps	99	Private
Adams	750Kbps	79	Private
Adams	5Mbps	109	Private

Outer Join:

OUTER JOIN מאפשר לנו לשלב נתונים משתי טבלאות (או יותר) כאשר חלק מהנתונים אינם מושתפים לשתי הטבלאות.

טבלת מחלקות			טבלת עובדים		
מספר מחלקה	שם מחלקה	מספר העובד	שם העובד	מספר מחלקה	שם מחלקה
10	כח אדם	1	משה	10	כח אדם
20	כספים	2	יוסי	20	כספים
30	יחסי ציבור	3	דוד	10	דוד
NULL		4	אלן	20	אלן
		5	חגי	10	חגי
		6	יובל	NULL	יובל

שילוב רגילים בין טבלאות אלו יוביל להציג הנתונים הבאים (5 שורות סה"כ):

שם מחלקה	מספר מחלקה	שם העובד	מספר מחלקה	שם מחלקה
כח אדם	10	משה	10	כח אדם
כספים	20	יוסי	20	כספים
כח אדם	10	דוד	10	כח אדם
כספים	20	אלן	20	אלן
כח אדם	10	חגי	NULL	חגי

נשים לב כי יוביל לא מופיעים ומחלקות יחסית ציבור לא מופיעות, הסיבה לכך נועוצה בצורה ההשוואה בין הטבלאות. הרו' CD' להשוות בין שתי טבלאות אלו אנו נכתבו :

```
1 | ON dep.department_id = emp.department_id
```

לומר, כל עוד יש שוויון בין הערכים של عمودת מספר מחלקה בטבלת עובדים לערכיהם של عمודת מספר מחלקה בטבלת מחלקות יש להציג את הנתונים.

* יוביל לא מופיע כי מספר מחלקתו היא NULL ולא ניתן להשוות NULL לאף ערך בעמודת מספר מחלקה בטבלת מחלקות.

* מחלקת ייחס ציבור לא מופיעה ממשום שמספרה הוא 30 ולא ניתן להשוות את ערך זה לאף ערך מעמודת מספר מחלקה בטבלת עובדים.

כדי להציג את כל הנתונים מעד אחד (شمאל או ימין) של הטבלה כולל אלו שאינם להם נתונים להשוואה מצד השני, משתמש Left OUTER JOIN או Right OUTER JOIN ב-

Right Outer Join:

Right OUTER JOIN

טבלת העובדים לפי ההדבמה שלנו מופיעה בצד **הימני** של משפט ה-JOIN :

```
1 | FROM      departments dep RIGHT OUTER JOIN employees emp
2 | (left)                (right)
```

אם נרצה להציג את כל העובדים אשר קיימים בטבלה כולל אלו אשר אין להם מחלקה
נשתמש בביטוי JOIN RIGHT :

```
1 | SELECT emp.last_name , emp.salary , dep.department_name
2 |
3 | FROM      departments dep RIGHT OUTER JOIN employees emp
4 |
5 | ON dep.department_id = emp.department_id
```

שילוב זה בין הטבלאות יוביל להצגת כל העובדים, כולל אלו אשר אין להם מחלקה (6 שורות סה"כ):

שם העובד	מספר מחלקה	שם המחלקה
משה	10	כח אדם
Յօն	20	כספים
דוד	10	כח אדם
אלן	20	כספים
רועי	10	כח אדם
NULL	NULL	NULL

Left Outer Join:

Left OUTER JOIN

טבלת מחלקות לפי ההדבמה שלנו מופיעה בצד **השמאלי** של משפט ה-JOIN :

```
1 | FROM      departments dep RIGHT OUTER JOIN employees emp
2 | (left)                (right)
```

אם נרצה להציג את כל המחלקות אשר קיימות בטבלה כולל אלו אשר אין להן עובדים נשתמש בביטוי Left OUTER JOIN:

```
1 | SELECT emp.last_name , emp.salary , dep.department_name
2 |
3 | FROM      departments dep LEFT OUTER JOIN employees emp
4 |
5 | ON dep.department_id = emp.department_id
```

שילוב זה בין הטבלאות אלו יוביל להצגת כל המחלקות, כולל אלו אשר אין בהן עובדים (6 שורות סה"כ):

EmployeeName	ManagerName
Davolio	Fuller
Fuller	NULL
Leverling	Fuller
Peacock	Fuller

שם העובד	מספר מחלקה	שם המחלקה
משה	10	כח אדם
Յօն	20	כספים
דוד	10	כח אדם
אלן	20	כספים
רועי	10	כח אדם
יהסִ צִבּוֹר	30	NULL

Full Outer Join:

כדי להציג את כל הנתונים משתי הטבלאות (גם את כל העובדים אשר לא שייכים לאף מחלקה וגם את כל המחלקות אשר אין בהן עובדים), נשתמש בביטוי JOIN FULL OUTER.

```
SELECT cust.customer_id, cust.last_name, cust.first_name, cust.pack_id, pack.speed
FROM customers cust LEFT OUTER JOIN packages pack
ON cust.pack_id = pack.pack_id
```

customer_id	last_name	first_name	pack_id	speed
1	Smith	Alvin	NULL	NULL
2	Jones	Jose	NULL	NULL
3	Taylor	Amado	7	750Kbps
4	Williams	Stuart	23	2.5Mbps

כל הלקוחות גם אלי
 שאין להם חבילות
 גישה

```
SELECT cust.customer_id, cust.last_name, cust.first_name, cust.pack_id, pack.speed
FROM customers cust RIGHT OUTER JOIN packages pack
ON cust.pack_id = pack.pack_id
```

customer_id	last_name	first_name	pack_id	speed
10...	NULL	NULL	NULL	750Kbps
10...	NULL	NULL	NULL	1.5Mbps
10...	NULL	NULL	NULL	10Mbps
10...	NULL	NULL	NULL	750Kbps
10...	NULL	NULL	NULL	1.5Mbps
10...	NULL	NULL	NULL	2.5Mbps
10...	NULL	NULL	NULL	1.5Mbps
10...	NULL	NULL	NULL	750Kbps

כל החבילות גם אלי
 שאין להם לקוחות

```
SELECT cust.customer_id, cust.last_name, cust.first_name, cust.pack_id, pack.speed
FROM customers cust FULL OUTER JOIN packages pack
ON cust.pack_id = pack.pack_id
```

customer_id	last_name	first_name	pack_id	speed
1	Smith	Alvin	NULL	NULL
2	Jones	Jose	NULL	NULL
3	Taylor	Amado	7	750Kbps

כל החבילות גם אלי
 שאין להם לקוחות

כל הלקוחות גם אלי
 שאין להם חבילות

Self-Join:

פקודת Self Join

```
SELECT emp.LastName AS 'EmployeeName' , mng.LastName AS 'ManagerName'
FROM employees emp JOIN employees mng
ON emp.ReportsTo = mng.EmployeeID
```

EmployeeName	ManagerName
Davolio	Fuller
Leverling	Fuller
Peacock	Fuller

מספר עובד	שם העובד	מספר מנהל
NULL	משה	
1	יוסי	
1	דוד	
3	אלן	
3	רועי	
5	יובל	

טבלה זו מציגה היררכיה מסוימת של עובדים ומנהליים כאשר מספר עובד מסוים יכול להיות שווה למספר מנהל

(1) משה ← (2) יוסי

(3) דוד ← (4) אלן

(5) רועי ← (6) יובל

ב_TBLAOOT מסווג זה, לעיתים, נרצה להציג נתונים המסתמכים על הקשר הפנימי של הטבלאות (מספר עובד שווה למספר מנהל).

לדוגמא, מהו שמו של המנהל של רועי? כדי לקבל תשובה נבדוק מהו מספר המנהל של רועי (3) ולאחר מכן נבדוק לאיזה מספר עובד מספר מנהל זה מתאים (דוד).

אנו יכולים להשתמש בקשר זה על מנת ליצור טבלה המציג את שם העובד ושם המנהל :

```
1 | SELECT emp.employee_id , emp.last_name , emp.manager_id ,
2 |           mng.last_name
3 |
4 | FROM   employees emp JOIN employees mng
5 |
6 | ON emp.manager_id = mng.employee_id
```

邏輯적으로, משלבים אותה טבלה לעצמה, אנו למעשה קוראים אותה טבלה פעמיים, כל פעם בכינוי אחר. צורת שילוב טבלאות זו נקראת **Self Join**.

במשמעות **ON** אנו מגדירים את הקשר הפנימי, במקרה זה אנו משווים בין מספר המנהל של העובד, לבין מספר העובד של המנהל (בפעם השנייה אנחנו מתיחסים לטבלה כטבלה "מנהל").

Group By

מדריך SQL – פקודת Group By

פקודת Group By מאפשרת לנו לקובץ את תוצאות השאלה שלנו לפי קבוצות מוגדרות. במדריך SQL זה נתמקד בפונקציות הקבוצה ופקודת ה Group By. להבדיל [פונקציות השורה](#) אשר פועלות על כל שורה ושורה, פונקציות הקבוצה פועלות על קבוצות, לדוגמה:

שם העובד	שכר
יוסי	2000
משה	3000
NULL	4000
יוסי	7000

השכר הממוצע של קבוצת עובדים אלו הוא 4000
השכר הגבוה ביותר של קבוצת עובדים אלו הוא 7000

* הקבוצה יכולה להיות בכל הטבלה או בתיקי קבוצות מהטבלה.

פונקציות הקבוצה והערך NULL

! אין כל השכלה מעיר NULL בעמודת שם העובד לפונקציות הקבוצה בדוגמא זו משום של החישובים נעשים על הערכים של عمودת שכר.

! פונקציות קבוצה אינן מתייחסת לערך NULL. לדוגמה, אם שכר אחד מהמעובדים היה NULL והינו רצים להפעיל על עמודת השכר את פונקציית הממוצע, היא הייתה מחברת 3 ערכים ומחלקת ב-3.

פונקציות קבוצה נפוצות

שם הפונקציה	תפקיד	הערות	עובדת על סוג	דוגמא (בהתאם לערכי הקבוצה מהדוגמא הקודמת)
AVG	מחזירה ממוצע	מספרים	מספרים	4000 ← SELECT AVG(sal)
SUM	מחזירה סכום	מספרים	מספרים	16000 ← SELECT SUM(sal)
MAX	מחזירה את הערך הגבוה ביותר	מספרים מחוזות תאריכים	מספרים מחוזות תאריכים	7000 ← SELECT MAX(sal)
MIN	מחזירה את הערך הנמוך ביותר	מספרים מחוזות תאריכים	מספרים מחוזות תאריכים	2000 ← SELECT MIN(sal)

שם הפונקציה	תפקיד	הערות	עובדת על סוג	דוגמא (בהתאם לערכי הקבוצה מהדוגמא הקודמת)
COUNT	מחזירה את כמות הערכים בעמודה מסוימת	מספרים מחוזות תאריכים	מספרים מחוזות תאריכים	3 ← SELECT COUNT(name) 4 ← SELECT COUNT(sal)
(*)COUNT	מחזירה את כמות השורות בטבלה	מספרים מחוזות תאריכים	מספרים מחוזות תאריכים	4 ← (*)SELECT COUNT
COUNT DISTINCT	ספירת את הערכים הייחודיים בטבלה	מספרים מחוזות תאריכים	מספרים מחוזות תאריכים	3 ← (name)COUNT 2 ← (DISTINCT name)COUNT

```

1 | SELECT      שם_עמודה , פונקציית_קבוצה(שם_עמודה)
2 | FROM        שם_טבלה
3 |
4 | WHERE       תנאי
5 |
6 | GROUP BY   שם_עמודה
7

```

עד כה, כל פונקציית קבוצה שהפעלה עלה על הערכים של כל העמודה. ברוב המקרים לא נרצה לקבל את הממוצע של כל טבלת העובדים שלנו, אלא נרצה לראות את ממוצע השכר בטבלה לפי כל מחלקה ומחלקה (מה ממוצע השכר של מחלקת זה אדם, מחלקת מחשב ועוד).

```

1 | SELECT      department_id , AVG(salary)
2 |
3 | FROM        employees
4 |
5 | GROUP BY   department_id

```

קוויים מנחים לעובדה עם GROUP BY

כל עמודה המופיעה ב- **SELECT** חייבת להופיע במשפט **GROUP BY**.

כasher אנו כתובים במשפט **SELECT** פונקציות קבוצה יחד עם עמודות אחרות מהטבלה, כל עמודה הנקתבת במשפט **SELECT** חייבת להופיע במשפט **GROUP BY**, אחרת תתקבל שגיאה.

```

SELECT      department_id , AVG(salary)
            (עמודה)          ↓ (פונקציית קבוצה)
FROM        employees
GROUP BY   department_id

```

כל עמודה הכתובה במשפט **GROUP BY** לא חייבת להופיע ב- **SELECT**.

אין מינעה מלבעז הקבוצות לפי עמודות שונות אך לא להציג במשפט **SELECT** (על אף שההתוצאה לא תצא קרייה).

```

1 | SELECT      AVG(salary)
2 |
3 | FROM        employees
4 |
5 | WHERE       department_id IN (50, 80, 90)
6 |
7 | GROUP BY   department_id

```

משפט **WHERE** מאפשר לנו לסנן תוצאות לפני ההקבוצה.

לדוגמא, יש להציג את הממוצעים עבור מחלקות 50, 80, 90 בלבד :

```

1 | SELECT      department_id , AVG(salary)
2 |
3 | FROM        employees
4 |
5 | WHERE       department_id IN (50, 80, 90)
6 |
7 | GROUP BY   department_id

```

! משפט WHERE למשזה מתבצע לפני פעולת **GROUP BY**, ונראה זאת בהמשך – סדר פעולות מול סדר ביצועים.

ניתן לכתוב במשפט GROUP BY יותר עמודה אחת.

בכל מחלקה יש משרחות שונות (מנהל, פקידים, אנשי תחזקה ועודומה). עד כה, ביקשנו להציג את הממוצע של כל המחלקה, עתה נבקש לראות את הממוצע במחלקה לפי כל משרה ומשרה :

```

1 | SELECT      department_id , job_id , AVG(salary)
2 |
3 | FROM        employees
4 |
5 | WHERE       department_id IN (50, 80, 90)
6 |
7 | GROUP BY   department_id , job_id

```

ראיינו כי ניתן באמצעות פקודת WHERE לבצע סינון תוצאות לפני ההקבוצה. עם זאת, לא ניתן לכתוב פונקציות קבוצה במשפט WHERE – הדבר יוביל לשגיאה.

המחלקות אשר הממוצע שלהם גובה מ- 5000 :

```

1 | SELECT      department_id , AVG(salary)
2 | FROM        employees
3 | WHERE       AVG(salary) > 5000
4 | GROUP BY   department_id
5 | (גיאח)

```

משפט ה HAVING

כדי לسانן תוצאות על סמך פונקציות קבוצה אנו משתמש במשפט HAVING.

```

1 | SELECT      שם_טבילה , פונקציית_קבוצה(שם_טבילה)
2 | FROM        טבילה
3 | WHERE       תנאי
4 | GROUP BY   שם_טבילה
5 | HAVING      תנאי_על_פונקציות_קבוצה

```

המחלקות אשר הממוצע שלהם גובה מ- 5000 :

```

1 | SELECT      department_id , AVG(salary)
2 | FROM        employees
3 | GROUP BY   department_id
4 | HAVING      AVG(salary) > 5000

```

ניתן לשלב בין משפט HAVING למשפט WHERE.

מתוך המחלקות 80, 50, 90, המחלקות אשר הממוצע שלהם גובה מ- 5000 :

```

1 | SELECT      department_id , AVG(salary)
2 | FROM        employees
3 | WHERE       department_id IN (50, 80, 90)
4 | GROUP BY   department_id
5 | HAVING      AVG(salary) > 5000

```

ניתן לבצע סינון על סמך פונקציית קבוצה אחרת מזו אשר מופיעה במשפט SELECT :

```

1 | SELECT      department_id , AVG(salary)
2 | FROM        employees
3 | WHERE       department_id IN (50, 80, 90)
4 | GROUP BY   department_id
5 | HAVING      MAX(salary) > 5000

```

קינון פונקציות קבוצה (Nesting)

פונקציות קבוצה ניתנות לקינון עד שני רמות.

מן ממוצעי השכר במחלקות השונות, ממוצע השכר הגבוה ביותר :

```

1 | SELECT      MAX(AVG(salary))
2 | FROM        employees
3 | GROUP BY   department_id

```

```
SELECT MIN(monthly_discount) FROM customers
SELECT MIN(join_date) FROM customers
SELECT MIN(last_name) FROM customers
```

סופר – Count
ערכים ללא null

```
--  
SELECT COUNT(Customer_Id) FROM customers  
SELECT COUNT(pack_id) FROM customers
```

```
SELECT AVG(ISNULL(pack_id, 0))  
FROM customers
```

כשנרצה לסקום ערכים שיש גם null ולה חשב נחפכם ל 0

```
SELECT martial_status, AVG(monthly_discount)  
FROM customers  
GROUP BY martial_status
```

martial_status	(No column name)
Single	37.524498
Divorced	37.202792
widowed	35.063539
Married	34.905875

```
SELECT city, AVG(monthly_discount)  
FROM customers  
GROUP BY city  
HAVING AVG(monthly_discount) > 40
```

city	(No column name)
Davenport	42.787500
Denton	45.727500
Detroit	46.830000

```
/*  
SELECT  
FROM  
WHERE  
GROUP BY  
HAVING  
ORDER BY  
*/
```

Where מtbody לפני
אינטגרציה

having מtbody אחריו
אינטגרציה

```
SELECT city, AVG(monthly_discount)  
FROM customers  
WHERE city LIKE 'ב%'  
GROUP BY city  
HAVING AVG(monthly_discount) > 40
```

Desc & Asc sort , AS

- Desc – מיון נזקן
- Descending מהגבוה לנמוך

- ASC – מיון נמוך לגבוה
- Ascending מהנמוך לגבוה

```
SELECT gender, martial_status, AVG(monthly_discount)
FROM customers
GROUP BY gender, martial_status
ORDER BY gender
```

gender	martial_status	(No column name)
Female	Divorced	36.986000
Female	Married	35.236080
Female	Single	37.024943

```
-- TOP-N Analysis
SELECT TOP 5 last_name, monthly_discount
FROM customers
ORDER BY monthly_discount DESC
```

last_name	monthly_discount
Davis	99.00
Thomson	99.00
Harris	99.00

```
SELECT DISTINCT state, city
FROM customers
```

state	city
Alabama	Birmingham
Alabama	Huntsville
Alabama	Mobile

ניתן גם לכתוב את הכינוי לעמודה גם ללא ה AS לדוגמא :

```
1 | SELECT last_name, salary, salary + 500 'NEW_SALARY'
2 |
3 | FROM employees
```

:התוצאה תראה כך:

1 Last_name	salary	NEW_SALARY
2 -----	-----	-----
3		
4		
5 Levi	5700	6200

ניתן גם לכתוב את הכינוי לעמודה ללא הגרשיים (')

```
1 | SELECT last_name, salary, salary + 500 new_salary
2 |
3 | FROM employees
```

:התוצאה תראה כך:

1 Last_name	salary	new_salary
2 -----	-----	-----
3		
4		
5 Levi	5700	6200

- ❖ כתיבת כינוי ללא גרשים מונעת מאייתנו לכתוב את הכינוי כשתי מילימ' נפרדות (ולא מחוברות לדוגמא ע"י קו תחתון).
- ❖ כתיבת כינוי ללא גרשים מונעת מאייתנו להשתמש במילימ' שמרורות כינויים.
- ❖ לצורך סדר וקריאות – תמיד להשתמש ב – AS , תמיד להשתמש בגרשיים ותמיד לכתוב צמד מילימ' או יותר בצורה מחוברת ע"י קו תחתון.

תת שאלתה היא שאלתה אשר ניתנת לקין בחלוקת שונים של שפת-SQL ואף בתוך תת שאלתה אחרת. לעתים קרובות אנו מתייחסים לתת השאלתה בשאלתה פנימית באשר השאלתה אשר קוראת לה נקראת השאלתה חיצונית. במדריך SQL זה אנו נרחב על הנושא ונראה כיצד ניתן לישמו.

אחד מטרותיה הנפוצות של תת השאלתה היא לאפשר לנו להתמודד עם שילפה המורכבת מרמות שונות. לדוגמה, מי הם העובדים אשר מרוויחים יותר מעובד מספר 54?

אם נבחן את השאלה האחורונה נשים לב כי זו מרכיבת משתי שאלות (רמות):

1. כמה מרוויח עובד מספר 54?

2. מי מרוויח יותר מעובד מספר 54?



מבנה תת שאלתה

```

1 | SELECT ...
2 | FROM   טבלה
3 | WHERE  תנאי
4 |
5 |       (SELECT ...
6 |           FROM   טבלה
7 |           )
8 |
9 |

```

קווים מנחים לעובדה עם תת שאלות

```

01 | SELECT first_name , last_name , salary
02 | FROM   employees
03 |
04 | WHERE  salary >
05 |
06 |           (SELECT salary
07 |             FROM   employees
08 |
09 |             WHERE  employee_id = 54)
10 |
11 |

```

❖ תת השאלתה מתבצעת קודם ולאחר מכן התוצאה שהיא החזרה עוברת לשאלתה החיצונית (קודם פועלת השאלתה אשר בודקת כמה עובד מספר 54 מרוויח, ולאחר מכן פועלת השאלתה החיצונית על התוצאה שלה, וכך שscarו עבר לשאלתה החיצונית).

❖ תת השאללה חייבת להיות מוקפת בסוגרים.

❖ על מנת לשפר את הקריאות, מומלץ לכתוב את תת השאלתה בצד ימין של תנאי ההשווה.

❖ לא ניתן להשתמש בתת השאלתה במשפט GROUP BY.

❖ ניתן לחלק את תת השאלות לשני סוגים :

○ תת שאלות אשר מושאות לעמודה ע"י אופרטורים פשוטים

.(Single Row Subquery)

○ תת שאלות אשר מושאות לעמודה ע"י אופרטורים מורכבים

.(Multiple Row Subquery)

Single Row Subquery

Single Row Subquery

הת שאלתה היא תת שאלתה אשר עובדת עם אופרטורים פשוטים (כגון =, <, >).
הת שאלתה זו יכולה להחזיר שדה אחד בלבד לשאלתה החיצונית (הדוגמא שפתחה את הபוטט היא תת שאלתה מסוג זה).
העובדים אשר עובדים במחלקה של עובד שמו Moshe:

```

1 | SELECT last_name , first_name , salary , department_id
2 |
3 | FROM   employees
4 |
5 | WHERE  department_id = (SELECT department_id
6 |
7 |                           FROM employees
8 |
9 |                           WHERE first_name = 'Moshe')

```

- ❖ תת השאלתה חייבת להחזיר שורה בודדת, אם היו כתבים את תת השאלתה ללא משפט WHERE היינו מקבלים שגיאה.
- ❖ תת השאלתה חייבת להחזיר עמודה אחת, אם היו כתבים במשפט SELECT של תת השאלתה יותר מעמודה אחת היו מקבלים שגיאה.
- ❖ אם יכולים לסקם ולומר כי תת שאלתה מסוג זה יכולה לקבל שדה בודד.
- ❖ אילו רצינו להציג את העובדים אשר עובדים במחלקה של עובד שמו Moshe, לא כולל Moshe היינו מוסיפים את המשפט 'And first_name <> 'Moshe' בשורה האחורונה של השאלה (מחוץ לתת השאלתה).

ניתן לשלב בתת השאלתה פונקציות קבועות

העובדים אשר מרוויחים יותר ממוצע השכר במחלקה מספר 60:

```

01 | SELECT  first_name , last_name , salary
02 |
03 | FROM   employees
04 |
05 | WHERE  salary >
06 |
07 |           (SELECT AVG(salary)
08 |
09 |             FROM   employees
10 |
11 |             WHERE  department_id = 60)

```

ניתן לשלב את תת השאלתה במפעט HAVING

המחלקות אשר שכר המוצע גובהה יותר מהשכר המוצע במחלקה 90:

```

01 | SELECT  department_id , AVG(salary)
02 |
03 | FROM   employees
04 |
05 | GROUP BY department_id
06 |
07 | HAVING  AVG(salary) >
08 |
09 |           (SELECT AVG(salary)
10 |
11 |             FROM   employees
12 |
13 |             WHERE  department_id = 90)

```

המחלקה אשר שכרן הממוצע גבוהה יותר מהשכר הממוצע במחלקת 90 :

```

01 | SELECT department_id , AVG(salary)
02 |
03 | FROM employees
04 |
05 | GROUP BY department_id
06 |
07 | HAVING AVG(salary) >
08 |          (SELECT AVG(salary)
09 |           FROM employees
10 |           WHERE department_id = 90)
11 |
12 |
13 |

```

ניתן לעבד עם מספר תתי שאלות בו דמנית

העובדים אשר מרוויחים יותר מעובד מס' 54 וגם נמצאים במחלקת עם עובד שמו Moshe (לא כולל Moshe)

```

01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary >
06 |
07 |          (SELECT salary
08 |           FROM employees
09 |           WHERE employee_id = 54 )
10 |
11 | AND department_id =
12 |
13 |          (SELECT department_id
14 |           FROM employees
15 |           WHERE first_name = 'Moshe')
16 |
17 |           AND last_name <> 'Moshe'
18 |
19 |
20 |
21 |

```

Multiple Row Subquery

Multiple Row Subquery

היא שאלתה אשר יכולה להחזיר יותר משדה אחד לשאלתה החיצונית. תת שאלתה זו עובדת עם האופרטורים **IN, ANY, ALL**.

לצורך הדגמת צורת פעולה של תת שאלתה זו אנו נתychos בדוגמאות הבאות לערכי UPDATED השכר עבור העובדים של מחלקת מס' 80 :

שכר
4300
5200
6700
8200
12500

האופרטור IN מאפשר לנו להשוות עמודה מול רשימה של ערכים החזרים מהתשאילתה.

העובדים אשר שכרם שווה לאחת המשכורות של העובדים במחלקה 80 :

```

01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary IN
06 |
07 |           (SELECT salary
08 |
09 |           FROM employees
10 |
11 |           WHERE department_id = 80)

```

אם נרצה להציג את כל העובדים אשר שכרם שווה לאחת המשכורות של העובדים במחלקה 80 חוץ מהעובדים במחלקה 80 כתוב $80 < > \text{id}$ AND department_id = 80 (מחוץ לתת השאלתה).

האופרטור ANY

האופרטור ANY מאפשר לנו להשוות עמודה מול לפחות אחד מהמערכים החזרים מהתשאילתה.

אנו יכולים לעבוד עם שלוש צורות ההשוואה הבאות כאשר משתמש באופרטור זה :

$< \text{ANY}$, $< \text{ANY}$, $= \text{ANY}$ $<$

$\text{ANY} <$

העובדים אשר שכרם גדול לפחות משכורת העובדים במחלקה 80.

כאשר אנו מבקשים לדעת את הערך המקורי לפחות אחד הערכים בתוך רשימה מסוימת אנו למעשה מבקשים את הערך המקורי מהמיינום (כי אין זה משנה מי הוא יהיה גדול, העיקר שהוא גדול לפחות אחד הערכים).

```

01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary > ANY
06 |
07 |           (SELECT salary
08 |
09 |           FROM employees
10 |
11 |           WHERE department_id = 80)

```

4300

12500

MIN

MAX

$> \text{ANY}$

→

העובדים אשר שכרם קטן לפחות מהתוארכים בתוך רשימה מסוימת אנו למשה מבקשים את הערך .80.

כasher אנו מבקשים לדעת את הערך הקטן לפחות מאחד הערכים בתוך רשימה מסוימת אנו למשה מבקשים את הערך הקטן מהמקסימום (כ' אין זה משנה מי הוא יהיה קטן, העיקר שייהי קטן לפחות מאחד הערכים).

```

01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary < ANY
06 |
07 |           (SELECT salary
08 |
09 |           FROM employees
10 |
11 |           WHERE department_id = 80)

```

4300 12500

MIN MAX

ANY >



ANY =

העובדים אשר שכרם שווה לפחות לאחת המשכורות של העובדים במחלקת .80.

למעשה פקודת ANY = פועלת באותה צורה כמו IN.

```

01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary = ANY
06 |
07 |           (SELECT salary
08 |
09 |           FROM employees
10 |
11 |           WHERE department_id = 80)

```

האופרטור ALL

האופרטור ALL מאפשר לנו להשוות עמודה מול כל הערכים החזורים מהתשאילתה.

אם יוכלים לבצע עם שתי צורות ההשוואה הבאות אשר אנו משתמשים באופרטור זה :

ALL , < ALL <

ALL <

העובדים אשר שכרם גדול מכל המשכורות של העובדים במחלקת .80.

כasher אנו מבקשים לדעת את הערך האגוד מכל הערכים בתוך רשימה מסוימת אנו למשה מבקשים את הערך האגוד מהמקסימום (כ' שערך יהיה גדול מכל הערכים עליו בהכרח להיות גדול מהערך המקסימלי ברשימת הערכים).

39

```
01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary > ALL
06 |
07 |           (SELECT salary
08 |
09 |             FROM employees
10 |
11 |               WHERE department_id = 80)
```

4300 12500

|—————|

MIN MAX

ALL <

→

ALL >

העובדים אשר שכרם קטן מכל המשכורות של העובדים במחלקה 80.

כאשר אנו מבקשים לדעת את הערך הקטן מכל הערכים בתוך רשימה מסוימת אנו למשה מבקשים את הערך הקטן מהמינימום (כדי שהערך יהיה קטן מכל הערכים עליון בהכרח להיות קטן ממהערך המינימלי ברשימה הערכים).

```
01 | SELECT first_name , last_name , salary
02 |
03 | FROM employees
04 |
05 | WHERE salary < ALL
06 |
07 |           (SELECT salary
08 |
09 |             FROM employees
10 |
11 |               WHERE department_id = 80)
```

4300 12500

|—————|

MIN MAX

ALL >

←

האופרטור ALL = מבקש את הערך אשר שווה לכל הערכים אשר חוזרים מתחת לשאליתה (העובד אשר שכו שווה ל-4300 וגם ל-5200 וגם ל-6700 וגם ל-8200 וגם ל-12500). !

משמעותו זה מבקש דבר לא הגיוני לרוב לא משתמש בו

מספר עובד	שם העובד	מספר מנהל
NULL	משה	1
1	יוסי	2
1	דוד	3
NULL	אלן	4
3	רועי	5
5	יובל	6

עבודה עם אופרטור NI – עובד בצורה תקינה

העובדים אשר הם בעצם הם מנהליים:

```
1 | SELECT employee_id , last_name  
2 |  
3 | FROM employees  
4 |  
5 | WHERE employee_id IN (SELECT manager_id  
6 |  
7 | FROM employees)
```

עובדת עם אופרטור NOT – לא מחייב תוצאות

העובדים אשר אינם מנהלים:

```
1 | SELECT employee_id , last_name  
2 |  
3 | FROM employees  
4 |  
5 | WHERE employee_id NOT IN (SELECT manager_id  
6 |  
7 | FROM employees)
```

לאחר מכן, מטרת הבדיקה היא לסייע לאנשי המילוט בפתרון בעיות טכניות או אבטחה.

כאשר מתחדש השאלותה מארילה ערבי | ||| נו אופרטורו IN NOT לא יחייב אף פגיעה

כדי לפתחו עצם ועלינו לדאגו ושקטה בשאיולטה לא מטבח ומכבוי | ||| :

```

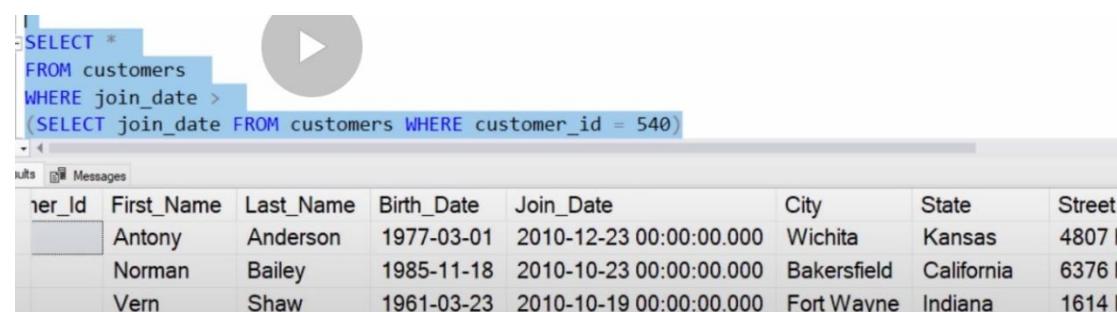
[ ] SELECT *
  FROM customers
 WHERE city = (SELECT city FROM customers WHERE customer_id = 170)

```

```

[ ] SELECT *
  FROM packages
 WHERE sector_id =
 (SELECT sector_id FROM packages WHERE pack_id = 10)
 AND pack_id <> 10

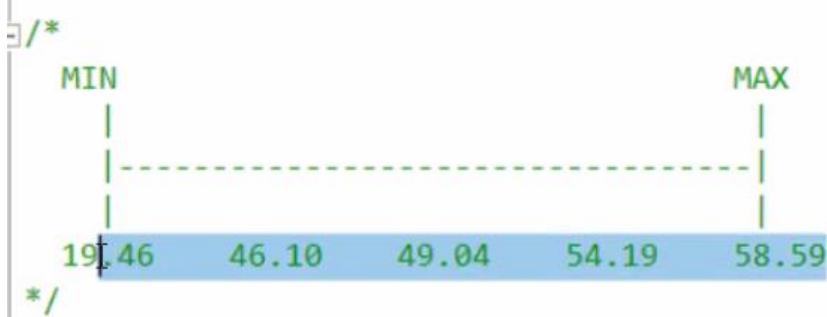
```



```

[ ] SELECT customer_id, last_name, city, monthly_discount
  FROM customers
 WHERE monthly_discount IN
          (SELECT monthly_discount
            FROM   customers
           WHERE  city = 'New York')

```



customer_id	last_name	city	monthly_discount
281	Armstrong	New York	46.10
378	Russell	Gilbert	54.19
424	Carr	Tempe	54.19
378			54.19

אנו נשתמש באלג'ריה
במקומות מסוימים של
שאלות מורכבות, זה
יהי קשה לשנות את תוכן
השאילתת כדי להגיע
ספציפית ל max , מאפשר
לנו להגיע לכל ערכי השדות
בליעות את תוכן
השאילתת

```
SELECT customer_id, last_name, city, monthly_discount
FROM customers
WHERE monthly_discount >ALL
      (SELECT monthly_discount
       FROM   customers
       WHERE  city = 'New York')
```

/*
MIN |-----| MAX
| |-----|
19.46 46.10 49.04 54.19 58.59
*/

Results Messages

customer_id	last_name	city	monthly_discount
89	Stevens	Reno	64.48
98	Kennedy	Gilbert	68.88
111	Pearson	Fontana	60.07

```
SELECT customer_id, last_name, city, monthly_discount
FROM customers
WHERE monthly_discount >ANY
      (SELECT monthly_discount
       FROM   customers
       WHERE  city = 'New York')
```

/*
MIN |-----| MAX
| |-----|
19.46 46.10 49.04 54.19 58.59
*/

View הוא טבלה לוגית אשר מבוססת לרוב על טבלה אחרת, ה-View עצמו לא מכיל נתונים אלא רק מציג את הנתונים אשר שייכים לטבלה המקורי אותה הוא מציג. בפוסט זה אנו נלמד :

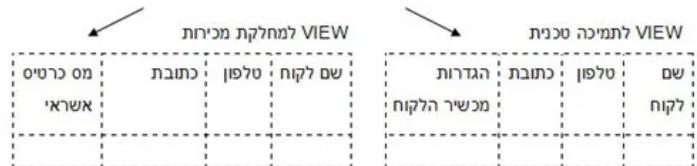
Views - Into

- ❖ מהו VIEW ומהם יתרונותיו.
- ❖ כיצד ניתן ליצור View, לשנות את הגדרותיו, ולמחוק אותו.
- ❖ כיצד ניתן לעדכן את טבלאות המקור ע"י VIEW.

מהו VIEW

נניח כי לפנינו טבלת ליקוחות שהicity לחברה מסוימת :

שם לקוחות	טלפון	כתובת	מספר כרטיסי אשראי	כל חדש	נק חיבטים	למכשיר הלוקה	הגדרות טכניות	ללקוח



על אף שעובדים רבים בחברה יהיו זקנים לגישה לטבלת הליקוחות, לא יהיה הגיוני לתת לכל עובד בחברה גישה לכל הנתונים בטבלה זו, משום שלכל עובד יש את קבוצת הליקוחותআithה הוא עובד ואת הפרטים הרלוונטיים על ליקוחותיו אוטם עליו לדעת. למשל, מצד אחד, צוות התמיכה ליקוחות לא תמיד יצטרך לראות את כל נתוני הליקוח כדוגמת כרטיס אשראי. מצד שני, צוות המכירות לא יצטרך לראות את ההגדירות הטכניות של מכשיר הליקוח וכן הלאה.

במקום זאת, ניתן לחלק את הטבלה למען תתי טבלאות, כל אחד עובד בחברה יראה את הנתונים השיכים רק לו, ולמעשה כל

הגישה לטבלאות תעשה ע"י תתי הטבלאות השונות שלה בלבד.

זהו מהות פועלות ה-View, יצירת מעין "תתי טבלאות" אשר יציגו למשתמש חלק מהנתונים. חשוב לציין כי ה-View עצמו אינו מכיל אף נתון, אנו לא משכפלים את הטבלה המקורי ע"י ה-View, אלא יוצרים מעין "חלון" לנתונים המקוריים.

יתרונות ה View

- ❖ אפשר לנו להגביל את הגישה לנתונים בטבלת המקור.
- ❖ פישוט שאלות מורכבות – אפשר להגדיר View המורכב משאלות מורכבות כדי להמנע מהקלדה בכל פעם שנרצה להשתמש בה.

יצירת View

```

1 | CREATE [OR REPLACE] VIEW
2 |
3 | AS
4 |
5 | SELECT ...
6 |
7 | FROM ...

```

44

```

1 | CREATE VIEW emp_vu
2 |
3 | AS
4 |
5 | SELECT last_name , first_name , phone_number
6 |
7 | FROM employees
8 |
9 | WHERE salary > 10000

```

שליפת נתונים מה VIEW

ה- VIEW מתנהга כמו טבלה לכל עניין, ניתן לבצע עליו שליפות באותה צורה בה ביצענו שליפות מכל טבלה אחרת, ניתן לשלוף ממנו נתונים באמצעות סינון (WHERE), באמצעות מיון (ORDER BY) ואף ניתן לבצע את פקודת ה-DESC עלייה.

מחיקת ה VIEW

כדי למחוק את VIEW אנו משתמש בפקודה :

```
1 | DROP VIEW emp_vu
```

לדוגמא :

```
1 | DROP VIEW emp_vu
```

עדכון מבנה ה VIEW

כדי לעדכן את ה-VIEW ניתן :

- למחוק אותו וליצור אותו בצורה החדשה בה אמו מעוניינים.
- להשתמש בפקודת – OR REPLACE

פקודה זו יוצרת VIEW חדש במידה וה-VIEW עם השם שרצינו אינו קיים.

במידה והוא קיים, היא "דורסת" את מבנה ה-VIEW הקודם ומחליפה אותו

בתחריר החדש.

```

1  CREATE OR REPLACE VIEW emp_vu
2
3  AS
4
5  SELECT department_id , AVG(salary) AS "AVG_SAL"
6
7  FROM employees
8
9  GROUP BY department_id

```

יצירת VIEW עם כינויים לעמודות

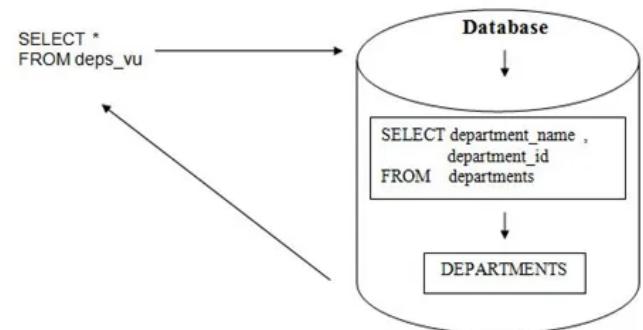
כאשר VIEW מכיל שאלתה המורכבת מעמודות עם חישובים, חובה לתת לכל אחת מעמודות אלן כינוי:

```

01  CREATE VIEW emp_vu_50
02
03  AS
04
05  SELECT UPPER(last_name) AS "LAST_NAME" ,
06
07      UPPER(first_name) AS "FIRST_NAME",
08
09      Salary * 1.3 AS "NEW_SALARY",
10
11      Hire_date
12
13  FROM employees
14
15  WHERE department_id = 50

```

كيفية شرط نتائج من VIEW متباعدة



לעתים אנו מדרשים להריץ שאילתות מורכבות על בסיס יומי. לדוגמה, שיליפת נתונים מסוימת טבלאות יחד עם חישובים מסוימים.

כדי להמנע מכתיבת השאילתות בכל פעם מחדש, אנו יכולים לשמר אותן כ-VIEWS ולקראן להן בעת הצורך ע"י השאילהה פשוטה.

```

01 | CREATE VIEW emp_dep_loc_vu
02 |
03 | AS
04 |
05 | SELECT emp.last_name || ' ' || emp.first_name AS "FULL_NAME" , emp.salary * 1.75 AS "NEW_SAL"
06 |
07 |         dep.department_name , loc.city
08 |
09 | FROM employees emp, departments dep, locations loc
10 |
11 | WHERE emp.department_id = dep.department_id
12 |
13 | AND
14 |
15 |     Dep.location_id = loc.location_id
16 |
17 | AND
18 |
19 |     (emp.salary BETWEEN 5000 AND 15000 OR emp.salary > 20000)

```

לאחר שמירת השאילהה כ-VIEW, בפעם הבאה שנרצה לקרוא לה נוכל לכתוב :

```

1 | SELECT *
2 |
3 | FROM emp_dep_loc_vu

```

פעולות DML על VIEWS

באופן כללי, פעולות DML על VIEWS גורמות לשינוי הנתונים הקיימים ב-VIEW והן בטבלת המקור.

```

1 | CREATE VIEW emp_basic_vu
2 |
3 | AS
4 |
5 | SELECT last_name , first_name , salary , phone_number
6 |
7 | FROM employees

```

: UPDATE VIEW זה, כגון UPDATE

```

1 | UPDATE emp_basic_vu
2 |
3 | SET salary = 1000
4 |
5 | WHERE last_name = 'King'

```

ביאו לשינוי הנתונים הקיימים ב-VIEW והן בטבלת המקור.

הגבלות על פעולות ה DML על VIEWS

במקרים מסוימים, עצם הגדרת VIEW בזורה מסוימת מונעת פעולות DML מסוימות עליו.

VIEW עם פונקציות קבוצה ומשפט BY

על VIEW אשר מורכב משפט BY GROUP BY ופונקציות קבוצה לא יהיה ניתן לבצע פעולות DML מסוימות עליו.
UPDATE, INSERT, DELETE

```

1 | CREATE OR REPLACE VIEW depts_avg_sal_vu
2 |
3 | AS
4 |
5 | SELECT department_id , AVG(salary) AS "AVG_SAL"
6 |
7 | FROM employees
8 |
9 | GROUP BY department_id

```

על VIEW אשר מורכב מפקודת DISTINCT לא יהיה ניתן לבצע פעולות DELETE, UPDATE, INSERT.

```

1 CREATE OR REPLACE VIEW dis_dep_job_vu
2 AS
3
4 SELECT DISTINCT department_id , job_id
5 FROM employees
6
7

```

VIEW עם המילה השמורה ROWNUM

על VIEW אשר מורכב יחד עם המילה השמורה ROWNUM לא יהיה ניתן לבצע פעולות DELETE, UPDATE, INSERT.

```

1 CREATE OR REPLACE VIEW five_emps_vu
2 AS
3
4 SELECT last_name , first_name , salary
5 FROM employees
6
7 WHERE rownum <= 5
8
9

```

VIEW עם ייחד עם עמודות חישוביות

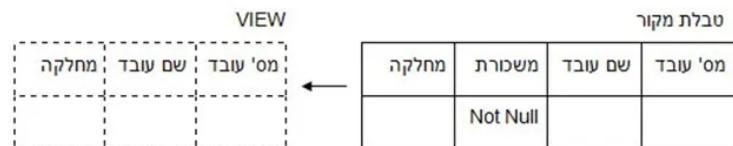
על VIEW אשר מורכב יחד עם עמודות חישוביות לא יהיה ניתן לבצע פעולות UPDATE, INSERT.

```

1 CREATE OR REPLACE VIEW emps_newsal_vu
2 AS
3
4 SELECT last_name , first_name , salary * 1.75 AS "NEW_SAL"
5 FROM employees
6
7

```

VIEW אשר לא כולל עמודות עם אילוץ NOT NULL בטבלת המקור



על VIEW אשר לא מורכב מעמודה עם אילוץ NOT NULL בטבלת המקור, לא יוכל לבצע פעולה INSERT.

```

1 CREATE OR REPLACE VIEW emps_vu
2 AS
3
4 SELECT employee_number , employee_name , department
5 FROM employees_source
6
7

```

מניעת פעולות DML על VIEWS

ניתן למנוע פעולות DML באמצעות יוזם בשני אופנים, בעזרת WITH CHECK OPTION ובעזרת .WITH READ ONLY.

ע"י פקודות WITH CHECK OPTION אנו נוכל למנוע פעולה UPDATE ו- INSERT אשר מוגדמת את התנאי שהצבנו.

1 | WHERE תנאי WITH CHECK OPTION

לדוגמא :

```
1 | CREATE OR REPLACE VIEW emps_60_vu
2 |
3 | AS
4 |
5 | SELECT *
6 |
7 | FROM employees
8 |
9 | WHERE department_id = 60 WITH CHECK OPTION
```

לא נוכל לעדכן לעובדים דרך VIEW זה את המחלקה למספר שונה מ 60 ולא נוכל להכניס דרך VIEW זה עובדים אשר ישוויכו למחלקה אחרת מ 60.

ע"י פקודות WITH READ ONLY נוכל למנוע כל פעולה DML על ה-VIEW שיצרנו :

```
1 | CREATE OR REPLACE VIEW basic_emps_vu
2 |
3 | AS
4 |
5 | SELECT last_name , first_name , salary
6 |
7 | FROM   employees
8 |
9 | WITH READ ONLY
```

פקודות החסובן משמשות לשילוף מידע הקשור לשתי טבלאות (או יותר אם משתמשים בחו"ז).
בדומה לפקודת join.

פקודות החסובן מאחדות שתי שאלות ובסופה של דבר, התוצאה היא טבלת נתונים.

בamar זה, נשתמש לצורך הדגמה בשתי טבלאות, שהשלד שלהם נראה כהה:

תיאור	סוג נתונים	שם
מפתח ראשי	מספר אוטומטי	id
שם המוצר	טקסט	name
מחיר המוצר	מספר	price

פקודת Union

כמשמעותם בפקודה, נשלפות רק רשומות שונות (כמו שימוש בdistinct), והעמודות הנשלפות
צריכות להיות מאותו סוג נתונים.
הפקודה נראה כהה:

```
select <fields> from tb11
UNION select <fields> from tb12
```

דוגמא

שליפת כל המוצרים השונים בשםם ובמחיר שלהם בקטגוריות 1 ו-2.

```
select name, price from products_store1
UNION select name, price from products_store2;
```

פקודת Union all

פקודת all union זהה לפקודת union – גם פה העמודות צריכות להיות מאותו סוג נתונים, אבל יש הבדל אחד – all union שולפת את כל הרשומות, גם אם יש רשומות שוות. הפקודה נראית ככזה:

```
select <fields> from tbl1
UNION ALL select <fields> from tbl2
```

דוגמה

שליפת כל המוצרים בקטגוריות 1 ו-2.

```
select name, price from products_store1
UNION ALL select name, price from products_store2;
```

שאילתתך ראשונה, מוצאת את שמות שלושת העובדים הראשונים לפי ה-id:

```
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` <= 3
```

שאילתתך שנייה, מוצאת את שמות העובדים שה-id שלהם 8 עד 10:

```
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` > 7 AND `workers_id` < 11
```

זו התוצאה:

workers_name	workers_id
Moshe	1
Yekhezkel	2
Yirmiyahu	3
Sheshet	8
Eliezer	9
Shlomo	10

וכדי לאחד בין השאלות נשתמש ב-**UNION**, כך נראית השאלתה המאוחדרת:

```
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` <=3
UNION
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` > 7 AND `workers_id` < 11
```

חשוב לזכור ש-**UNION** מציג כל רשומה (שורה בסיסי הנתונים) רק פעם אחת. לדוגמה, השאלה הבאה שורצוה לשולף את העובדים 1-3, וגם את אילו עובדים בקריות.

התוצאות:		
workers_city	workers_name	workers_id
Hakrayot	Moshe	1
Ramat Hasharon	Yekhezkel	2
Bat Yam	Yirmiyahu	3
Hakrayot	Sheshet	8
Hakrayot	Eliezer	9
Hakrayot	Shlomo	10

```
SELECT `workers_id`, `workers_name`, `workers_city` FROM `workers`
WHERE `workers_id` <= 3
UNION
SELECT `workers_id`, `workers_name`, `workers_city` FROM `workers`
WHERE `workers_city` = 'hakrayot'
```

שים לב, ש-**Moshe** אינו מופיע פעמיים למרוחק שהוא אחד מששת העובדים הראשונים וגם תושב הקריות.

כל משפט SELECT בתוך UNION חייב להיות בעל אותו מספר עמודות לעמודות **חייבות להיות גם סוג נתונים דומים**
העמודות בכל משפט SELECT חייבות להיות באותו סדר

האופרטור UNION בוחר רק ערכים מובהנים כברירת מחדל. כדי **אפשר ערכים כפולים**, השתמש **-UNION ALL**:

שמות העמודות ב-Result-set שווים בדרך כלל **לשמות העמודות במשפט ה-SELECT הראשון**.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

Example

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

הצורה SQL הבאה מחייבת את הערים (רק ערכים נפרדים) הן מהטבלה "LKQHOT" והן מטבלה "SUPPLIERS":

הערה: אם לחלק מהלkommenות או הספקים יש אותה עיר, כל עיר תופיע רק פעם אחת, מכיוון UNION בוחר רק ערכים שונים. השימוש ב- UNION ALL כדי לבחור גם ערכים כפולים!

Example

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

הצורה SQL הבאה מחייבת את הערים (גם ערכים כפולים) מהטבלה "LKQHOT" וגם מהטבלה "SUPPLIERS":

```
SELECT City, Country FROM Customers  
WHERE Country='Germany'  
UNION ALL  
SELECT City, Country FROM Suppliers  
WHERE Country='Germany'  
ORDER BY City;
```

```
SELECT City, Country FROM Customers  
WHERE Country='Germany'  
UNION  
SELECT City, Country FROM Suppliers  
WHERE Country='Germany'  
ORDER BY City;
```

```
SELECT 'Customer' AS Type, ContactName, City, Country  
FROM Customers  
UNION  
SELECT 'Supplier', ContactName, City, Country  
FROM Suppliers;
```

Type	ContactName	City	Country
Customer	Alejandra Camino	Madrid	Spain
Customer	Alexander Feuer	Leipzig	Germany
Customer	Ana Trujillo	México D.F.	Mexico

הבדל בין Union ל Join

אם בהוראות join הוספנו לשורות בטבלה עמודות מטבלה אחרת ע"י חיבור של הטבלאות עם שדה מתאים, ב union אנחנו מחברים את השורות של שתי השאלות ולטבלה אחת.

Select Distinct

השימוש ב- SELECT DISTINCT

הפקודה SELECT DISTINCT נועדה来找出重复的记录。为了避免重复的输出。

במקרה שלנו, ניתן לכתוב את השאלה הבא:

```
SELECT DISTINCT Category
FROM Products
```

ולקבל את התוצאה הבאה:

	Category
▶	1
	2

שhai בא בדיק התוצאה שרצינו לקבל – רשימת קטגוריות המוצרים, כאשר כל קטgorיה מופיעă פעמי אחת בדיק.

Insert

הערות:

- אם עמודה מסוימת מוגדרת כנדרשת (required), חובה לציינה בראשית העמודות ולתת לה ערך.
- אפשר לא לתת ערך לעמודה מסוימת (לא לציינה בראשית העמודות), אם העמודה לא נדרשת.
- ערך שודות מסווג מחוץ, יוקפו בגרשיים ('').
- לא ניתן לתת ערך לשדה מסווג "מספר אוטומטי". שדה זה מקבל את הערך באופן אוטומטי.
- ב- MS ACCESS, על מנת להכניס ערך לשדה מסווג תאריך/שעה, יש לעוטפו בסולניות (#).

לדוגמא, הכנסת רשומה חדשה לטבלת המחברים:

```
INSERT INTO Authors (Author,Year_Born)
VALUES ('Moshe Cohen',1975)
```

בשאילתת הגדרנו את השדות אליון אנו מכניסים נתונים (Author,Year_Born), ואת הערכים בשרצינו לחתם להם ('Moshe Cohen',1975). לשדה ID Au לא ניתן ערך, כי זהו שדה מסווג מספר אוטומטי.

יש להקפיד שהערך הנitin לעמודה מסוימת, תואם את סוג הנתונים לעמודה מכילה!

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

Example

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

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

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

הנתן מספר רשומות

כדי להוציא מספר שורות, נשתמש בתחביר הבא:

```
INSERT INTO workers(workers_name, workers_phone1,
workers_city, workers_date_added)
VALUES ('Moshe', '09-9888887', 'Hakrayot', '2011-06-
07'),
('Yechezkel', '04-4888887', 'Ramat Hasharon', '2011-06-
07'),
('Yirmiyahu', '03-3333337', 'Bat Yam', '2011-06-07')
```

הערכים עברו כל שורה תחומים בסוגרים, ומופרדים בפסיק מהרשומה הקודמת.

Create

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

מודגדר אוטומטי

דנית (ימין – מאיפה, שמאל כמה קפיצה)

```
CREATE TABLE Persons (
    Personid int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Update

UPDATE Multiple Records

It is the `WHERE` clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

Example

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

Update Warning!

Be careful when updating records. If you omit the `WHERE` clause, ALL records will be updated!

Example

```
UPDATE Customers
SET ContactName='Juan';
```

[Try it Yourself »](#)

Delete

SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

-> **מחיקת טבלה (הגדירות הטבלה נשארות)**

(

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

```
DE TRUNCATE TABLE Customers
```

-> **מחיקת טבלה (הטבלה נשארת)**

DROP TABLE Edar1

```
ALTER TABLE Customers
DROP Phone
ALTER TABLE Customers
ADD Address TEXT(100)
```

שינוי תוכן טבלה (drop - למחוק) - Alter table

57

CREATE DATABASE myNewDB

כיצד נגביל את השיליפה רק לרשותות מסוימות?

אתה הדריכם המקובלות ביותר להגביל את השיליפה היא באמצעות WHERE-ב-טבלה, שגבל את התוצאות לשורות המבוקשות.

לדוגמה:

```
SELECT workers_id, workers_name, workers_city
FROM workers
WHERE workers_id = 8
```

התוצאה היא הרשימה שנמצאת בעמודה שבה 8

workers_city	workers_name	workers_id
Hakrayot	Sheshet	8

לדוגמה נוספת:

```
SELECT workers_id, workers_name, workers_city
FROM workers
WHERE workers_name = 'gershon'
```

התוצאה היא הבא:

workers_city	workers_name	workers_id
Beer-sheva	Gershon	4

```
-- Table structure for table `workers`
--

CREATE TABLE `workers` (
  `id` int(11) UNSIGNED NOT NULL,
  `name` varchar(255) DEFAULT '',
  `age` int(2) UNSIGNED DEFAULT 0,
  `gender` enum('female','male','n/a') DEFAULT NULL,
  `phone` varchar(20) DEFAULT '',
  `comment` text,
  `created_at` datetime DEFAULT current_timestamp(),
  `updated_at` datetime DEFAULT NULL ON UPDATE current_timestamp(),
  `active` int(1) UNSIGNED DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=UTF8;

-- 

-- Indexes for table `workers`
-- 

ALTER TABLE `workers`
  ADD PRIMARY KEY (`id`);

-- 
-- AUTO_INCREMENT for table `workers`
-- 

ALTER TABLE `workers`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT;
```

פקודה ליצירת טבלה היא CREATE TABLE, שם הטבלה (workers), ובין סוגים נמצאות כל עמודות הטבלה. לדוגמה, id, name, וכי"ב.

מגדיר את ENGINE=InnoDB המונע של מסד הנתונים. מונע בריתת המחדל הוא MyISAM. לכל אחד יש את היתרונות והחסרונות שלו. אני מעדיף את InnoDB בכלל שהוא תומך בפתרונות מורחבים ובטרנץ אקטזיות.

DEFAULT CHARSET=UTF8 מה שאומר שניין להשתמש בסט תווים מורחב הכול את האותיות העבריות.

העמודה id היא עמודת המספר הסידורי של הרשותות (שורות) בטבלה. לכל רשומה חייב להיות מספר סידורי ייחודי שיזהה אותה בטבלה.

NOT NULL – אסור שהשדה יהיה ריק. לפיכך, בכל רשומה (שורה) שנוספה לטבלה חייב השדה id לקבל ערך. הרשומה הראשונה תהיה 1, השנייה 2, וכו'ב.

PRIMARY KEY – מגדיר את עמודת המספר הסידורי של הטבלה. חשוב ביותר להגדיר primary key מפני שבאמצעותנו אנחנו מזהים את השורה לצורך שילוף הנתונים וערכיהם. שמו לב! יכולה להיות עמודת primary key אחת בלבד בכל טבלה.

סוג הנתונים בכל עמודה – מצד ימין של שם העמודה סוג הנתונים (data type).

ל-id סוג נתונים int(11) שיכול להכיל מספרים שלמים עד סדרי גודל של 10 בחזקת 11, בתנאים שאינם שליליים (UNSIGNED).

עמודות age שיכת לסוג int(2) והוא גם UNSIGNED מה שאומר מספרים בין 0-99.

לעמודות phone ו-name השתמשתי ב-varchar, שמתאים למחרוזות טקסט קצרות, עד 255 תווים, כשבסוגרים מופיעים מספר התווים.

סוג הנתונים datetime מאפשר הדנת נתונים בפורמט: ss:hh:mm:ss-MM-DD-YYYY. לדוגמה, 2019-09-14 16:30:01. בריתת המחדל עבור הסוג datetime הוא current_timestamp מה שיזין את נתוני הזמן העכשוויים.

עמודות updated_at מתעדכנת אוטומטית בזמן העדכן של רשומה בטבלה הוזלת ל-(ON UPDATE current_timestamp). את העמודה comment הגדרתי מסוג מידע text כדי שהשדה יוכל להיות מזמן עד 255 תווים (בניגוד/varchar אין מגדירים את אורך המחרוזת בסוגרים).

יכול לקבל אחת מהאפשרויות: "female", "male", "n/a".

SQL Injection <- SQL פריז

כאשר לא תהיה בקירה ואקר יכניס נתונים ש תמיד יהיו נכונים כדי לקבל את כל נתונים המאגר

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

ביטה לוגי 1=1 או " = "

UserId: 105 OR 1=1

SELECT * FROM Users WHERE UserId = 105 OR 1=1;

SQL Injection Based on ""="" is Always True

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ='' + uName + '' AND Pass ='' + uPass + '''
```

User Name:

" or ""=

Password:

" or ""=

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

SELECT * FROM Users; DROP TABLE Suppliers

horאות מחוברים ב: **Batched SQL** ← משפט!

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

User id: 105; DROP TABLE Suppliers

SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

ASP.NET Razor Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = @0";
db.Execute(txtSQL,txtUserId);
```

SQL Parameters for Protection
בודק כל פרמטר כדי לוודא שהוא נכון לערומה
שלו וمتיחסים אליו כפשווטו, ולא חלק מה-
SQL שיש לבצע.

Another Example

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

SELECT STATEMENT IN ASP.NET:

```
txtUserId = getRequestString("UserId");
sql = "SELECT * FROM Customers WHERE CustomerId = @0";
command = new SqlCommand(sql);
command.Parameters.AddWithValue("@0",txtUserId);
command.ExecuteReader();
```

INSERT INTO STATEMENT IN ASP.NET:

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
command = new SqlCommand(txtSQL);
command.Parameters.AddWithValue("@0",txtNam);
command.Parameters.AddWithValue("@1",txtAdd);
command.Parameters.AddWithValue("@2",txtCit);
command.ExecuteNonQuery();
```

INSERT INTO STATEMENT IN PHP:

```
$stmt = $dbh->prepare("INSERT INTO Customers (CustomerName,Address,City)
VALUES (:nam, :add, :cit)");
$stmt->bindParam(':nam', $txtNam);
$stmt->bindParam(':add', $txtAdd);
$stmt->bindParam(':cit', $txtCit);
$stmt->execute();
```

SQL CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

Example

```
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;
```

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

Example

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);
```

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers;
```

- Stored Procedure

פרוצדורה מאוחסנת היא **קוד SQL** מוקן שנייתן לשמור, כך שנitinן לעשות **שימוש חוזר** בקוד שוב ושוב.

אך אם יש לך שאלות SQL שאתה כותב שוב ושוב, שמור **אותה כפרופCEDURA מאוחסנת**, ואך פשוט קרא לה כדי לבצע אותה.

ניתן גם להעביר פרמטרים להליך מאוחסן, כך שהפרופCEDURA המואחסנת יכולה לפעול על סמך ערכי הפרמטרים המועברים.

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers @City = 'London';
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

```
mysql> CREATE TABLE employees_clone LIKE employees;
```

Now, execute another SQL statement which inserts all the records from *employees* table into *employees_clone* table. After executing this statement you'll get the *employees_clone* table which is an exact copy or duplicate of the *employees* table

```
mysql> INSERT INTO employees_clone SELECT * FROM employees;
```

```
CREATE TABLE new_table SELECT * FROM original_table;
```

The following command creates a simple copy of the *employees* table.

```
mysql> CREATE TABLE employees_dummy SELECT * FROM employees;
```



Tip: Use the `CREATE TABLE ... SELECT` syntax to quickly create a simple copy of any table that only includes the structure and data of the source table.

```
mysql> CREATE TEMPORARY TABLE persons SELECT * FROM persons;
```

טיפ: שולחן זמני יכול לקבל אותו שם כמו שולחן בסיס קבוע. אם אתה מציין את השם של טבלה זמנית זהה לטבלה הבסיס הקיימת, אז **טבלת הבסיס הקבועה מוסתרת עד שהטבלה זמנית תישמש**.
ערה: מכיוון **טבלאות זמניות הן ספציפיות להפעלה**, לכן **זמן מבלי להתגש זה עם זה**.

```
mysql> DROP TEMPORARY TABLE persons;
```

- Cloning or Copying a Table

יתכן מצב שבו אתה רק רוצה ליצור עותק או שיבוט **מודיק של טבלה קיימת כדי לבדוק או לבצע משוח מבלי להשפייע על הטבלה המקורית**.

תחילה נוצר טבלה ריקה המבוססת על **ההגדרה של טבלה מקורית**. הוא יכול גם את **תכונות העמודות והאינדקסים שהוגדרו בטבלה המקורית**.

ואז לאכלס את הטבלה הריקה בנתונים מהטבלה המקורית

עם זאת, אם אתה רק רוצה ליצור **טבלה מטבלה אחרת מבלי לחת בחשבן** **תכונות ואינדקסים של עמודות**, אתה יכול **לשימוש במשפט הפשט פשוט של שורה אחת**

- SQL Temporary Table

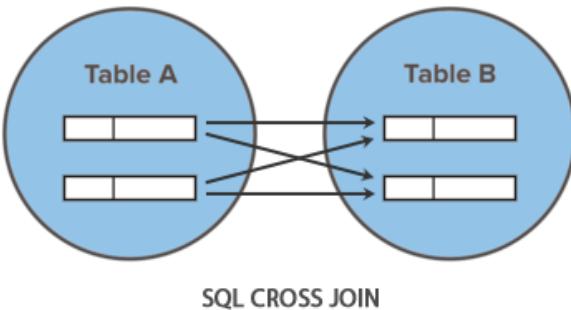
טבלה זמנית היא **טבלה הנראית רק להפעלה הנוכחיית**, והיא **נשמרת אוטומטית כאשר הפעלה** **שהה היא נוצרה נסגרת**.

מכיוון **טבלאות זמניות אין מאוחסנות במסד הנתונים** באופן קבוע, לכן, זה יהיה שימושי במצב שבו **אתה צריך טבלה רק בזמן קצר כדי לבצע או לבדוק משהו**, ולאחר מכן **אתה רוצה שהוא תהיה תיילם אוטומטית**.

טבלאות זמניות יכולות להיות **שימושיות במצבים שבהם אתה רק רוצה לבדוק את שאלות SQL מבלי לשפייע על מסד הנתונים**.

טבלאות זמניות **נשמרות אוטומטית כאשר חיבור מסד הנתונים או הפעלה** **שהם הינם נוצרות נסגרים**.

עם זאת, אם **ברצונך למחוק אותם מבלי לסגור את הפעלה הנוכחיית**, תוכל להשתמש במשפט **DROP TEMPORARY TABLE**, באופן הבא:



- Cross Join Operation

אם אין מציין תנאי צירוף בעת צירוף שתי טבלאות,
מערכת מסדר הנתונים **משלבת כל שורה מהטבלה
הראשונה עם כל שורה מהטבלה השנייה.**
סוג זה של חיבור נקרא חיבור צולב cross join או
מוצר קרטזיани.

דיאגרמת Venn הבאה ממחישה כיצד פועלת חיבור
צולב.

To understand this easily, let's look at the following *employees* and *departments* tables.

emp_id	emp_name	hire_date	dept_id	dept_id	dept_name
1	Ethan Hunt	2001-05-01	4	1	Administration
2	Tony Montana	2002-07-15	1	2	Customer Service
3	Sarah Connor	2005-10-18	5	3	Finance
4	Rick Deckard	2007-01-03	3	4	Human Resources
5	Martin Blank	2008-06-24	NULL	5	Sales

Table: employees

Table: departments

מספר השורות בחיבור צולב
הוא מכפלה של מספר
השורות בכל טבלה.

```

1  SELECT t1.emp_id, t1.emp_name, t1.hire_date, t2.dept_name
2  FROM employees AS t1 CROSS JOIN departments AS t2;
```

emp_id	emp_name	hire_date	dept_name
1	Ethan Hunt	2001-05-01	Administration
2	Tony Montana	2002-07-15	Administration
3	Sarah Connor	2005-10-18	Administration
4	Rick Deckard	2007-01-03	Administration
5	Martin Blank	2008-06-24	Administration
1	Ethan Hunt	2001-05-01	Customer Service
2	Tony Montana	2002-07-15	Customer Service
3	Sarah Connor	2005-10-18	Customer Service
4	Rick Deckard	2007-01-03	Customer Service
5	Martin Blank	2008-06-24	Customer Service
1	Ethan Hunt	2001-05-01	Finance
2	Tony Montana	2002-07-15	Finance
3	Sarah Connor	2005-10-18	Finance
4	Rick Deckard	2007-01-03	Finance
5	Martin Blank	2008-06-24	Finance
1	Ethan Hunt	2001-05-01	Human Resources
2	Tony Montana	2002-07-15	Human Resources
3	Sarah Connor	2005-10-18	Human Resources
4	Rick Deckard	2007-01-03	Human Resources
5	Martin Blank	2008-06-24	Human Resources
1	Ethan Hunt	2001-05-01	Sales
2	Tony Montana	2002-07-15	Sales
3	Sarah Connor	2005-10-18	Sales
4	Rick Deckard	2007-01-03	Sales
5	Martin Blank	2008-06-24	Sales

הגדרת משתנים :

```

SQLQuery2.sql - (L...-LAPTOP\shaha (5))*
DECLARE @str nvarchar(100) = 'Shahar'
PRINT @str
DECLARE @i int = LEN(@str)

PRINT @i
  
```

Messages

15

Shahar

6

```

DECLARE @i int
PRINT @i
SET @i = 30
SET @i = @i / 2
PRINT @i
  
```

```

DECLARE @dt datetime = DATEADD(hour, 100, GETDATE())
SELECT @dt
  
```

(No column name)
1 2017-06-25 20:43:27.560

הגדרת תנאי :

```

DECLARE @i int = 5
DECLARE @str nvarchar(100) = CASE
    WHEN @i < 5 THEN 'Smaller'
    WHEN @i = 5 THEN 'Equal'
    ELSE 'Bigger'
END
PRINT @str
  
```

```

DECLARE @i int = 5
DECLARE @str nvarchar(100)
IF @i > 7
BEGIN
    SET @str = 'Blah'
END
ELSE
BEGIN
    SET @str = 'Blee'
END
PRINT @str
  
```

Messages

Blee

```

Query2.sql - (I...-LAPTOP\shaha (53))*
IF EXISTS(SELECT NULL FROM Users WHERE DisplayName LIKE '%Shahar%' )
BEGIN
    SELECT *
    FROM Users
    WHERE DisplayName LIKE '%Shahar%'
END
ELSE
BEGIN
    PRINT ':-('
END

```

Results

AboutMe		Age	CreationDate	DisplayName	DownVotes	EmailHash	L
50667	NULL	43	2009-01-01 07:47:32.477	Shahar Mosek	3	NULL	
190662	NULL	NULL	2009-10-15 14:37:58.333	Shahar	0	NULL	
319715	NULL	NULL	2010-04-18 14:53:14.623	Shahar	0	NULL	
268049	NULL	36	2010-02-07 07:12:31.943	Shahar	0	NULL	
471528	NULL	NULL	2010-10-10 12:54:03.317	Shahar Mor	0	NULL	

```

Query2.sql - (I...-LAPTOP\shaha (53))*
IF EXISTS(SELECT NULL FROM Users WHERE DisplayName LIKE '%Shahadfghdsfhdf%' )
BEGIN
    SELECT *
    FROM Users
    WHERE DisplayName LIKE '%Shahar%'
END
ELSE
BEGIN
    PRINT ':-('
END

```

```

SQLQuery2.sql - (I...-LAPTOP\shaha (53))*
DECLARE @i int =0
WHILE @i < 100
BEGIN
    IF @i > 50
    BEGIN
        BREAK
    END
    PRINT @i
    SET @i=@i+1
END

```

Messages

:-(

הגדרת לולאה :

161 %

Messages

36
37
38
39
40
41
42
43
44
45

מה זה נוטן לנו?

תאורתית – אנחנו יכולים למש כל אלגוריתם ב- T-SQL (כי היא Turing Complete).

ברמה המעשית – אפשר לנו לפעמים לכתוב קוד ולבנות פעולות בקוד, שלא ניתן להביע באמצעות מניפולציות על result-sets כמו בשאלות שראינו עד עכשין.

בעיה

אנחנו רוצים למלא טבלה מורכבת משתי עמודות:

- מזהה המשתמש

- כתובות כל השאלות שכתב מופרדים ב- |~|

```

DECLARE @str2 nvarchar(max)
SELECT @str2='Shahar'
PRINT @str2
  
```

Messages Shahar

מציע הגדרת ערכי Select
תכונות

```

DECLARE @userId int = 5684416
DECLARE @str nvarchar(max) = ''

SELECT @str = @str + '|~|' + Title
FROM Posts
WHERE PostTypeId=1 AND OwnerUserId = @userId

SELECT UserId=@userId, String = @str
  
```

Userid	String
5684416	~ Scope between methods ~ Bug when displaying images in iOS app ~ Array of rounded buttons ~ App out of phone screen bounds ~ Background color ~ App not displaying properly on iphone6 ~ Reset UIButton background color to storyboard set values ~ Xcode inferred size classes



אנחנו יודעים לעשות עבור משתמש בודד.

בגלל שצורת הפיתרון שלנו עבור משתמש בודד מערבת הגדרת משתנה `string`, אנחנו לא יכולים לשלב את זה עם `OUTER APPLY` מול טבלת המשתמשים.

לכן נ└ר על הפיתרון הבא:

נגידיר משתנה שיציג לנו את טבלת התוצאות

נעבור בולולאה על כל מזהי המשתמשים שכתבו שאלה ב-SO

ニיצר עבור כל אחד מהם את ה- `string` המבוקש ומוסיף למשתנה של טבלת התוצאות

ונחזר את המשתנה של טבלת התוצאות

Cursor: מצביע לשורה הבאה
בתשובה השאלתה

איך עוברים בולולאה על ערכים של שאלה?

מגדירים את השאלה שלנו, ומגדירים CURSOR מולה

ה- CURSOR זה למעשה "iterator" על השאלה.

הוא מצביע כל פעם על שורה בודדת ואנחנו יכולים להוציא ממנה ערכים

עשימים לולאה כל עוד ה- `cursor` לא הגיע לסוף

בגוף הולולאה עושים את הפעולה על השורה הבודדת

שה- `cursor` מצביע אליה

סוגרים את ה- `cursor` ומשחררים משאים

```
DECLARE @results table (UserId int, DisplayName nvarchar(200), SpecialString nvarchar(max))

INSERT @results(UserId, DisplayName, SpecialString)
VALUES (1, 'A', 'B')

INSERT @results(UserId, DisplayName, SpecialString)
VALUES (2, 'A', 'B')

SELECT *
FROM @results
```

בנייה טבלה זמנית

השלבים

המטרה: להזמין result-set שכולל שורה עבור כל משתמש שפירס שאלת עם זה,/sl/, שם, וכותרת השאלה שפירס מופרdata ב-/-/.

ਪੰਤੇ

הגדרת משתנה טבלי שיכיל את התוצאה

הגדרת Cursor מיל שאלת שביבאה את ה- `Id` וה- `DisplayName` של כל המשתמשים שריסו שאלת

פתיחת ה- `Cursor`

הגדרת משתנים שמכילים את הערכים של שורה בודדת הבאת השורה הראשונה

בלולאה – כל עוד הצלחתי להביא מידע ביצוע פעולה שנקוט רצים יצירתי `string`-ה- `string`-ה- שארח רצים (שרשור הקותרה) הכנסת התוצאה למשתנה של התוצאות

הבות השורה והאה (קדום לולאה)

סגירת ה- `Cursor` ושהרכות שליפה של התוצאות

Results	Messages	
Userid	DisplayName	SpecialString
1	A	B
2	A	B

ਪੰਤੇ

השלבים

לזהרן result-set שכול שורה עבר ל משתמש שפיטס שאלת עם זה – של, שם, וכותרת השאלות שפירסם מופרדות ב- | – |

לא מעניין אותנו העמודות
שיחסרו רצים רק לבדוק אם
קיים

גדרת משתנה טבלי שיכיל את התוצאות
אדרת Cursor מיל שאלת שבסאה את ה- Id והוא DisplayName של כל
משתמשים שפיטס שאלת Cursor –
תינחת ה- Cursor
ובדרת משתנים את הערכים של שורה בודדת

ולא – כל עוד הצלחן להביא מידע
בציע העהלה שאחט וויזית ה- string-sharch שאנחן רצים (שרות הלקוחה)
הכנסת התוצאה למשתנה של התוצאות
הבטה והשורה הבהה (קדום הלילה)

אגירת ה- Cursor ושרורו
וליפה של התוצאות

יעילות – הליכה על טבלה עם
פחות שדרות

מטרה: להציג result-set שורה עבר ל משתמש שפיטס שאלת עם זה – של, שם, וכותרת השאלות שפירסם מופרדות ב- | – |

שיטה:

הגדרת משתנה טבלי שיכיל את התוצאות
הגדרת Cursor מול שאלת שבסאה את ה- Id והוא DisplayName של כל
משתמשים שפיטס שאלת Cursor –
פתחת ה- Cursor
הגדרת משתנים שמילים את הערכים של שורה בודדת
בתה השורה האחסונה
בלאלה – כל עוד הצלחן להביא מידע
בציע העהלה שאחט וויזית ה- string-sharch שאנחן רצים (שרות הלקוחה)
הכנסת התוצאה למשתנה של התוצאות
הבטה והשורה הבהה (קדום הלילה)

סיגרת ה- Cursor ושרורו
וליפה של התוצאות

הכנסת ערכים מהשורה
הבא למשתנים שהגדרכנו

ICHIZIR 0 במקורה יהיה
עוד תוצאות שקריא

הכנסת ערכים לטבלה
הזמןית – Insert Into

	Results	Messages
1	Id DisplayName	
1	5735116 Bbuck	
2	5735117 Daniel Irias	
3	5735123 sanjay sasidharan	
4	5735124 ysrome	
5	5735127 NU2Rails	
6	5735128 king2002	
7	5735130 DudeGuy	
8	5735138 newbieDeveloper	
9	5735139 ruhnet	

```
--1
DECLARE @results table (UserId int, DisplayName nvarchar(200), SpecialString nvarchar(max))

--2
SELECT Id, DisplayName
FROM Users
WHERE (Users.CreationDate BETWEEN '2016-01-01' AND '2016-01-05') AND
      EXISTS (SELECT NULL FROM Posts WHERE PostTypeId=1 AND Posts.OwnerUserId = Users.Id)

--3
OPEN usersCursor

--4
DECLARE @currentUserId int, @currentUserDisplayName nvarchar(200)

--5
FETCH NEXT FROM usersCursor INTO @currentUserId, @currentUserDisplayName

--6
WHILE @@FETCH_STATUS = 0 --6
BEGIN
    --6.1
    DECLARE @str nvarchar(max) = ''
    SELECT @str = @str + '|~|' + Title
    FROM Posts
    WHERE PostTypeId=1 AND OwnerUserId = @currentUserId

    --6.2
    INSERT INTO @results(UserId,DisplayName, SpecialString)
    VALUES(@currentUserId, @currentUserDisplayName, @str)

    --6.3
    FETCH NEXT FROM usersCursor INTO @currentUserId, @currentUserDisplayName
END

--7
CLOSE usersCursor
DEALLOCATE usersCursor

SELECT *
FROM @results
```

	UserId	DisplayName	SpecialString
1	5735116	Bbuck	~ Combining two tables with some blank informati...
2	5735117	Daniel Irias	~ How to read a JSON array using Volley and jso...
3	5735123	sanjay sasidharan	~ How to connect to Azure file share from Visial S...
4	5735124	ysrome	~ New web server instance is routing to old publi...
5	5735127	NU2Rails	~ Rails - Trying to check if a column is empty bef...
6	5735128	king2002	~ less CSS parse error: media definitions require b...
7	5735130	DudeGuy	~ C# - Array of a class that holds arrays of anothe...
8	5735138	newbieDeveloper	~ Min max value in iswaceous, no errors showing

MERGE

- .
שילוב של מידע מקורות שונים
- .
צורת עבודה כללית:
- .
מגדירים טבלת בסיס
- .
מגדירים את הנתונים החדשניים – אלה שאמורים להשתלב
- .
מגדירים איך מתבצעת ההצלבה בין ה-target (מה שאמור להשתלב) לבין טבלת הבסיס (ה-source)
- .
מגדירים מה לעשות אם יש התאמה (כלומר זה עדכון לשורה קיימת) ומה לעשות אם אין התאמה (שורה חדשה? שורה שאמורה להימחק?)

Close
Locate

:Marge

שילוב ועדרון נתונים באמצעות הצלבת טבלאות

- **לאיזה מקור יהיה As Target**
- **באיזה ערכים להשתמש Using**
- **מאייזה מקור יהיה Source**
- **תנאי הצבה on**
- **כשהתקיימה התאמה When matched**
- **כשלה התקימה התאמה When not matched**

MERGE

```

MERGE Students AS target
USING (
    VALUES
        (6, 'Shahar', 'LName1'),
        (7, 'Shahar', 'LName2')
    ) AS source (ID, FirstName, LastName)
ON (target.ID = source.ID)
WHEN MATCHED THEN
    UPDATE SET target.FirstName = source.FirstName,
               target.LastName = source.LastName
WHEN NOT MATCHED THEN
    INSERT (FirstName, LastName)
        VALUES (source.FirstName, source.LastName)
OUTPUT $action;

```

– דוגמא נוספת – MERGE

נבנה טבלה שמכילה עבור כל משתמש שפיריםם פוסט את התאריך והשעה של הפוסט הראשון והאחרון שפיריםם

נתיחה רק לנ נתונים עד יוני 2016.

```

CREATE TABLE UsersFirstLastPost
(
    UserId int,
    FirstPost datetime null,
    LastPost datetime null
)

GO

INSERT INTO UsersFirstLastPost(UserId, FirstPost, LastPost)
SELECT OwnerUserId, MIN(CreationDate), MAX(CreationDate)
FROM Posts
WHERE CreationDate < '2016-06-01'
GROUP BY OwnerUserId

```

נניח שאחרי היצירה הראשונית אנחנו רוצים לעדכן את הטבלה זאת מול טבלת ה-*Posts*.

כלומר, רוצים ללקחת דטלאות של מידע (יתכן אם חפייה) ולמזג אותם לטבלה הכללית.

```

MERGE UsersFirstLastPost AS target
USING (
    SELECT OwnerUserId, MIN(CreationDate), MAX(CreationDate)
    FROM Posts
    WHERE CreationDate >= '2016-05-01' AND CreationDate < '2016-06-03'
    GROUP BY OwnerUserId
) AS source (OwnerUserId, MinPublished, MaxPublished)
ON (target.UserId = source.OwnerUserId)
WHEN MATCHED AND source.MaxPublished > target.LastPost OR source.MinPublished < target.FirstPost THEN
    UPDATE SET target.LastPost = CASE WHEN source.MaxPublished > target.LastPost
        THEN source.MaxPublished
        ELSE target.LastPost
    END,
    target.FirstPost = CASE WHEN source.MinPublished < target.FirstPost
        THEN source.MinPublished
        ELSE target.FirstPost
    END
WHEN NOT MATCHED THEN
    INSERT (UserId,FirstPost, LastPost)
    VALUES (source.OwnerUserId, source.MinPublished, source.MaxPublished);

```

הוספה מספר שורות בבבאת אחת

71

```
INSERT INTO [dbo].[Grades]
([StudentID]
,[CourseID]
,[Grade]
,[Weight]
,[ExamDate])
VALUES
( 1, 1, 90, 20, GETDATE())
( 1, 1, 90, 20, '2016-01-01')
```

INSERT SELECT



```
CREATE TABLE UsersFirstLastPost
(
    UserId int,
    FirstPost datetime null,
    LastPost datetime null
)
```

GO

```
INSERT INTO UsersFirstLastPost(UserId, FirstPost, LastPost)
SELECT OwnerUserId, MIN(CreationDate), MAX(CreationDate)
FROM Posts
GROUP BY OwnerUserId
```

הכנס עבור משתמש את הפוסט
שכתב הכי מוקדם והכי מאוחר

	UserId	FirstPost	LastPost
1	NULL	2016-01-01 00:01:27.340	2016-12-10 10:44:36.007
2	-1	2016-01-01 20:16:23.060	2016-12-10 18:18:29.373
3	4	2016-09-23 16:37:27.113	2016-10-18 15:00:54.810
4	13	2016-02-02 00:58:03.243	2016-02-04 03:29:52.737
5	25	2016-02-18 05:31:01.880	2016-02-18 05:31:01.880
6	33	2016-03-23 11:39:08.453	2016-11-17 10:55:14.103
7	36	2016-07-05 19:15:12.150	2016-07-05 19:15:12.150
8	39	2016-08-09 06:51:50.633	2016-08-10 18:08:24.720
9	48	2016-02-05 00:07:45.920	2016-04-25 14:36:31.897
10	51	2016-07-07 20:50:47.430	2016-09-29 12:47:52.530
11	56	2016-05-11 09:40:11.050	2016-09-15 12:58:08.290

קבלת ערכים בחזרה עם OUTPUT

```
INSERT INTO [dbo].[Grades]
([StudentID]
,[CourseID]
,[Grade]
,[Weight]
,[ExamDate])
OUTPUT inserted.ID, inserted.StudentID
VALUES (
    1, --value for StudentID
    1, --value for CourseID
    90, -- value for Grade
    20, -- value for Weight
    GETDATE() -- value for ExamDate
)
```

:Output

הדף שעובדנו איתם

UPDATE



```
|UPDATE UsersFirstLastPost  
SET LastPost = DATEADD(hour, 1, UsersFirstLastPost.LastPost)  
FROM  
(  
    SELECT TOP(100) *  
    FROM UsersFirstLastPost  
    JOIN Users ON Users.Id = UsersFirstLastPost.UserId  
    WHERE Location='Israel'  
    ORDER BY Users.Id  
) d  
WHERE UsersFirstLastPost.UserId = d.UserId
```

:Update - 2

עדכן ערכים באמצעות from

```
with cte as (  
    SELECT TOP(100) *  
    FROM UsersFirstLastPost  
    JOIN Users ON Users.Id = UsersFirstLastPost.UserId  
    WHERE Location='Israel'  
    ORDER BY Users.Id  
)  
UPDATE cte  
SET cte.LastPost = DATEADD(hour, 1, cte.LastPost)
```

UPDATE



```
with cte as (  
    SELECT TOP(100) *  
    FROM UsersFirstLastPost  
    JOIN Users ON Users.Id = UsersFirstLastPost.UserId  
    WHERE Location='Israel'  
    ORDER BY Users.Id  
)  
UPDATE cte  
SET cte.LastPost=DATEADD(hour, 1, cte.LastPost)  
OUTPUT inserted.UserId,  
      inserted.LastPost AS newVal,  
      deleted.LastPost AS prevVal
```

	Userid	newVal	prevVal
1	130635	2017-06-19 20:39:34.177	2017-06-19 19:39:34.177
2	133568	2017-06-19 20:39:34.177	2017-06-19 19:39:34.177
3	139300	2017-06-19 20:39:34.177	2017-06-19 19:39:34.177
4	140937	2017-06-19 20:39:34.177	2017-06-19 19:39:34.177
5	141408	2017-06-19 20:39:34.177	2017-06-19 19:39:34.177
6	141947	2017-06-19 20:39:34.177	2017-06-19 19:39:34.177

DELETE

:Delete - 2

מחיקת ערכים באמצעות in

```
;with cte as (
    SELECT TOP(100) UsersFirstLastPost.*
    FROM UsersFirstLastPost
    JOIN Users ON Users.Id = UsersFirstLastPost.UserId
    WHERE Location='Israel'
    ORDER BY Users.Id
)
DELETE FROM UsersFirstLastPost
WHERE UserId IN (SELECT UserId FROM cte)
```

```
DELETE FROM UsersFirstLastPost
OUTPUT deleted.UserId
WHERE FirstPost BETWEEN '2016-01-01' AND '2016-01-03'
```

מבוא :

אינדקסים הם טבלאות חיפוש מיוחדות שצריכות לשמש את מנוע החיפוש של מסד הנתונים כדי להאיץ את אחסון הנתונים.

אינדקס הוא פשוט התייחסות לנתונים בטבלה. אינדקס מסד נתונים דומה לאינדקס בחלק האחורי של יומן. המשמשים אינם יכולים לצפות בו ורק להשתמש בו כדי להאיץ את הגישה למסד הנתונים.

לדוגמה, על מנת להתייחס לכל העמודים בספר העווקים בנושא מסוים, עוברים תחילת לאינדקס, המפרט את כל הנושאים בסדר אלפביתי, ולאחר מכן עוברים למספר עמודים ספציפיים אחד או יותר.

אינדקסים מוגנים ערכיהם קבועים בעמודה או בשילוב העמודות שעלייהן הוא נוצר. לאחר אנדקס SQL הם בערך כל ביצועים, הם שימושיים ביותר כאשר מסד נתונים גדול בגודלו.

האינדקס המקבץ - clustered index : הוא אחד מסוגי האינדקסים הפופולריים ביותר הנתמכים על ידי שרת SQL . סוג זה של אינדקס נוצר אוטומטית עם מפתח ראשי .

הfonקציות השימושיות בעבודה

אתם הם :

CREATE INDEX Command

DROP INDEX Command

הפקודה CREATE INDEX

פקודה זו משמשת **ליצירת אינדקס הטבלה ב-SQL** באמצעות ביטוי **index build**. בזמן יצירת הטבלה, זה אפשר לך לקבל את הנתונים המשוכפלים על הטבלה.

יצירת אינדקס הופך עמודות לשאלתה מהירה יותר על ידי יצירת מצביעים למקום מאוחסן הנתונים במסד נתונים .

נניח שאתה צריך **לאתר פיסת נתונים ספציפית** במסד נתונים ענק. כדי **לאחד נתונים אלה** ממשד הנתונים, המאפשר יחסם בכל שורה עד שתמצא אותם. **השאלה** זו **יקח הרבה זמן** לפעול אם הנתונים **שאתה מחפש הם קראת הסוף**.

Syntax:

CREATE INDEX name_of_Index ON name_Of_Table (Attribute1, Attribute2,);

Code:

```
create INDEX ind_1 on studentdet(renamed);
```

הסבר:

אנחנו יכולים **ליצור רשימות ממוינות** באמצעות **אינדקסים ב-SQL** **במקום** **צורך ליצור טבלאות ממוינות חדשות**, שיתפסו הרבה מקום אחסון.

INDEX - היא מילת המפתח **לציין אינדקס**.

Ind_1 הוא **שם האינדקס**.

Studentdet הוא **שם הטבלה**.

בדוגמה שלמעלה, **Ind_1** נוצר עם השדות של **dept regno** **studentdet** **טבלה**.

Single-Column Indexes

רק **עמודת טבלה אחת** משמשת **לבנייה אינדקס עמודה חד כיווני**.

Syntax: `CREATE INDEX name_of_Index ON name_Of_Table (Attribute1);`

Code: `create INDEX ind_reg on studentdet(regno);`

Explanation:

כאן `ind_reg` הוא האינדקס שנוצר עם התוכנה `regno`.

Unique Indexes

על פי אינדקס ב-**SQL**, **אינדקסים ייחודיים** משמשים **לשימוח הנתונים כמו גם לדיווק**. אינדקס ייחודי **מנע** הذנת ערכים כפולים **לטבלה**.

Syntax: `CREATE UNIQUE INDEX name_index on tables_na (Attribute_name);`

Code: `CREATE UNIQUE INDEX sidindex on Student_DB (L_Name);`

Explanation:

שם הטבלה הוא `Student_DB` עם השדה `L_Name`. ה-`אינדקס הייחודי` שנוצר עבור **הטבלה** `Student_DB` עם **השדה** `L_Name`.

Composite Index

לפי אינדקס ב-**SQL**, **אינדקס על שתי עמודות או יותר** של טבלה ידוע בתור **אינדקס מורכב - composite index**. זה עשוי **לייצר** את אותו אינדקס עם **מספר שונה של עמודות**.

Syntax: `CREATE INDEX Name_index ON db_table (column1, Column2.....);`

Code: `CREATE INDEX stu_index ON Student_DB(S_ID, L_Name);`

Explanation:

שם הטבלה הוא `Student_DB` עם השדות `L_Name , S_ID`. ה-`אינדקס המורכב` שנוצר עבור **הטבלה** `Student_DB` עם **השדות** `L_Name , S_ID`.

Implicit Indexes

ניתן להפיק את הרשימות הממייניות באמצעות אינדקסים במקומם יוצרת טבלאות ממייניות חדשות, שיש בהן הרבה מקומות אחסן. כאשר עמודה מסוימת כמפתח ראשי או מיוחד, האינדקסים המרומיינים נוצרים באופן אוטומטי. על עמודות אלה, אנחנו לא צריכים לבנות אינדקס.

Syntax: CREATE INDEX IN_Name ON TN_Name(Attribute1..);

Code: CREATE INDEX ind_stu_det ON Student_DB(Regno);

Explanation:

לטבלה student_db יש תכונות כמו regno, name, dept וכן הלאה. העמודה או התכונה עשויים לזרות את אילוצי המפתח הראשי או אילוצים ייחודיים אחרים שהאינדקסים נוצרים באופן אוטומטי ללא כל הפרעה.

DROP INDEX Command

ניתן להשתמש בפקודת SQL **DROP** כדי להסיר אינדקס. בעת הורדת מדד, היזהר כי הביצועים עשויים להאט או להשתפר. פקודה זו משמשת בדרך כלל למחיקת האינדקס בטבלת הנתונים שצינה.

Syntax: DROP INDEX name_of_index on Table_name; **DROP INDEX index_name;**

Code: DROP INDEX ind_stu_det on Student_DB

Explanation:

לטבלה student_db נסיר את האינדקס ind_std_det

```
1 ALTER TABLE workers
2 DROP INDEX email_index
```

מהו יש להימנע מאיינדקסים

אם נמנם אינדקסים נועדו לשפר את היעילות של מסד נתונים, אך ישנו מקרים שבהם יש להימנע מהם, למשל:

- ❖ על טבלאות קטנות
- ❖ טבלאות שמקובלות הרבה עדכוני אצווה - big batch גודלים או הוספה
- ❖ עמודות עם מספר גדול של ערכי אפס – null
- ❖ עמודות שעוברות מניפולציות - manipulated תכופות
- ❖ שבו עמודות עוברות מניפולציות באופן קבוע.
- ❖ כאשר התוכנה או השדה מתעדכנים לעיתים קרובות
- ❖ התוכנות אינן משתמשות לעיתים קרובות במצב שאליטה

יתרונות אינדקס ב-SQL

- ❖ האז את שאלת הבירהה
- ❖ עוזר להפוך שורה למינוחדת או **לא כפליות (primary, unique)**
- ❖ נוכן לבדוק מול ערכי מחרוזת רחבה אם האינדקס מוגדר לאינדקס של טקסט מלא ולמצוא מילה ממשפט.

החרוננות של אינדקס ב-SQL

- ❖ אינדקסים לוקחים יותר מקום בדיסק.
- ❖ DELETE , UPDATE ו- INSERT מושרים כלום על **ידי אינדקסים**, אבל UPDATE מואץ אם לתנאי WHERE יש שדה באינדקס. מכיוון שיש לשנות את האינדקסים עם כל תחלה, DELETE-INSERT ו-UPDATE נועשים **איטיים** יותר.

סיכום

SQL Server - משמש בעיקר לאחסון רשומות בסכמת מסד נתונים בצורה של דפים. לרוב, גודל העמוד של ערכיהם הוא **C8B**. לכל מסד נתונים יש לפחות **שני קבצים**: אחד עבור הנתונים, עם סוג הקובץ המוגדר כברירת מחדל של **.mdf**, ואחד עבור היומן, עם סוג הקובץ המוגדר כברירת מחדל של **.ldf**. יש עמוד אחד או יותר עבור כל טבלה במסד הנתונים. SQL Server משתמש באוסף מיוחד של דפים הנקרא **IAM** (מפתח הקצאת אינדקס) כדי לעקוב אחר דפים אלו.

<https://iw.tutorialcup.com/interview/sql-interview-questions/indexes-sql.htm>

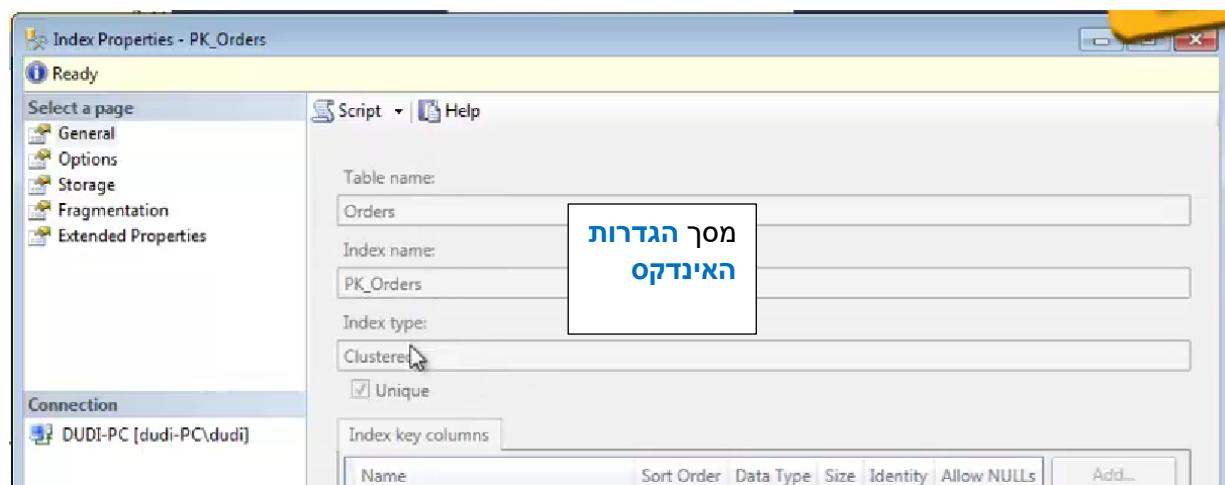
78

איך נמצאים האינדקסים

dbo.Orders

- Columns
- Keys
- Constraints
- Triggers
- Indexes
 - CustomerID (Non-Unique, Non-Clustered)
 - CustomersOrders (Non-Unique, Non-Clustered)
 - EmployeeID (Non-Unique, Non-Clustered)
 - EmployeesOrders (Non-Unique, Non-Clustered)
 - OrderDate (Non-Unique, Non-Clustered)
 - PK_Orders (Clustered)**
 - ShippedDate (Non-Unique, Non-Clustered)
 - ShippersOrders (Non-Unique, Non-Clustered)
 - ShipPostalCode (Non-Unique, Non-Clustered)

מצביע – מוביל
אינדקס אוטומטי, אינדקס מצוין



Limit:

הגבלת מספר הפריטים שמוסיפים בשאליטה נעשית באמצעות מילת המפתח **LIMIT**

במידה ולא מעוניינים שהפריט הראשון שיוחזר יהיה הראשון בסט התוצאות, מוסיפים **להגדרת ה LIMIT- את הפריט הראשון.**

```
SELECT `workers_id` , `workers_name`
FROM `workers`
LIMIT 5
```

חזר את חמישת התוצאות הראשונות בטבלה:

workers_name	workers_id
Moshe	1
Yechezkel	2
Yirmiyahu	3
Gershon	4
Asher	5

בדוגמה הבאה, מעוניינים להציג את התוצאה הריבועית עד השבעית מס' התוצאות. כך תראה השאלה:

```
SELECT `workers_id` , `workers_name`
FROM `workers`
LIMIT 3 , 4
```

או התוצאות:

workers_name	workers_id
Gershon	4
Asher	5
Metushelah	6
Yoshiahu	7

```
SELECT `workers_id` , `workers_name`
FROM `workers`
LIMIT 0 , 4
```

workers_name	workers_id
Moshe	1
Yechezkel	2
Yirmiyahu	3
Gershon	4

```
SELECT UNIX_TIMESTAMP('2020-04-20 17:30:00')
```

1587393000

כדי לקבל את חותמת הזמן ברגע זה:

```
SELECT UNIX_TIMESTAMP()
```

1588261483

ניתן לקבל חותמת זמן בפורמט :

```
SELECT NOW()
```

2020-04-20 17:31:33

גם המתודה **SYSDATE** נותנת את חותמת הזמן עכשווית:

```
SELECT SYSDATE(6)
```

2020-04-20 17:32:04.450769

بعد המתוודות **NOW**-**UNIX_TIMESTAMP** רצוט פעם את כל הילך (שאליטה או פרוצדורה), SYSDATE רצה בכל פעם שקיים לה ויכול לסייע להזמין זוראי בקבוק אשר מעכבים את ביצוע הקזון.

Dates:

פורמט הזמן

פורמט DATE (**שנה, חודש ויום**) הוא שונה ב-**MySQL** מהנוהג בישראל

לדוגמא: 2020-04-20

השנה מקדימה את החודש, והיום נמצוא בסוף.

הפורט **TIME** יכול החלק שנים עד רמת דיווק של 6

ספרות,
17:30:00.123456 hour:minutes:seconds.split_seconds

הפורט **TIME** מחבר את **TIME** ו-**DATE** ל-**DATETIME**

2020-04-20 :TIME DATE DATETIME

17:30:00.123456

חותמת זמן

חותמת הזמן היא מספר השניות מראשית הספירה עבור מחשבים שחל

בתאריך 1970-01-01 00:00:00, **זמן epoch**.

כדי לקבל את חותמת הזמן עבור תאריך ספציפי:

ההפרש בין שני תאריכים

נשתמש ב-**DATEDIFF** כדי לקבל את ההפרש בין שני תאריכים.

```
SELECT DATEDIFF('2020-04-27', '2020-04-20')
```

7

- ההפרש הוא בימים אלא אם צוין אחרת.

הסדר משנה:

```
SELECT DATEDIFF('2020-04-20', '2020-04-27')
```

-7

כדי לראות את ההפרש ביחידות אחרות (חודש, שבוע, שנות, דקוט, וכו') נעביר פרמטר נוסף. לדוגמה, **MONTH**:

```
SELECT TIMESTAMPDIFF(MONTH, '2020-04-06', '2020-07-31')
```

3

תוספת והפחתה של זמנים

כדי להוסיף זמן. לדוגמה, 7 ימים:

```
SELECT ADDDATE('2020-04-20', INTERVAL 7 DAY)
```

2020-04-27

כדי להפחית זמן. לדוגמה, 3 חודשים:

```
SELECT SUBDATE('2020-04-20', INTERVAL 3 MONTH)
```

2020-01-20

איך ליצור תאריכים?

כדי ליציר **TIME** מפורט מעה?

```
SELECT MAKETIME(17,30,3)
```

17:30:03

כשמייצרים **DATE** צריך לנקח בחשבון שהפורמט משתנה בין ארצות ו שימושים.

בפורמט ISO משתמשים מסדי נתונים:

```
SELECT GET_FORMAT(DATE, 'ISO')
```

%Y-%m-%d

```
SELECT GET_FORMAT(DATETIME,'ISO')
```

%Y-%m-%d %H:%i:%s

כדי להציג תאריכים בהתאם לניהוג באירופה וישראל:

```
SELECT GET_FORMAT(DATE, 'EUR')
```

מתודת נוספת לטיום

כדי לבדוק על פי תאריך מתייך עמודה מסווג **DATETIME** משתמש במתודה **DATE()**

```
SELECT * FROM tablename WHERE DATE(datetime) = '2020-01-20'
```

ALTER TABLE:

מבוא

81

```
ALTER TABLE users
MODIFY id int(11) AUTO_INCREMENT;
```

וכבשו האינדקס יתחל למספר מ-1, וכל רשותה שנוסיף תקבל ערך "יחודי" הגבוה ב-1 מאינדקס הרשותה הקודמת. הרשותה הראשונה תקבל אינדקס 1, הרשותה השניה תקבל באופן אוטומטי אינדקס 2, הרשותה השלישית תקבל אוטומטית אינדקס 3 וכו'ב.

לעתים, נרצה להתחילה מערך שנה מ-1. לדוגמה, אם יש לנו כבר טבלה עם 99 רשותות נרצה להתחילה את הספירה של האינדקס מערך הגבוה מס'רשות הרשותה הקיי'וות. אז אנחנו יכולים להוראות MySQL להמשיך מאינדקס 100:

```
ALTER TABLE users
AUTO_INCREMENT = 100;
```

נבדוק ע"י כך שנכנים רשותה ונראה איזה אינדקס היא תקבל:

```
INSERT INTO users (id, first_name, birthdate)
VALUES (NULL, 'Moshe','1993-05-07');
```

מצ' בטבלה כדי לראות מה קיבלנו:

+ Options		← T →	▼	id	first_name	birthdate
				100	Moshe	1993-05-07 00:00:00

נשתמש בדירקטיבה **AFTER** כדי להוסיף עמודה נוספת לעמדת מס'רשות. לדוגמה, אחרי העמודה `first_name`:

```
ALTER TABLE users
ADD COLUMN approved tinyint(1) NOT NULL
DEFAULT 0 AFTER first_name;
```

הוספה עמודה
נוסיף עמודה בשם <code>approved</code> לטבלה <code>users</code>

```
ALTER TABLE users
ADD COLUMN approved tinyint(1) NOT NULL
DEFAULT 0;
```

שינוי התכונות של עמודה

סוג הנתונים `tinyint` יכול לקבל ערכים שליליים. כדי למנוע קבלת ערכים שליליים: **UNSIGNED** (attribute) ששםה שוניה את העמודה על ידי שונייפ להכנה

```
ALTER TABLE users
MODIFY COLUMN approved tinyint(1) UNSIGNED
NOT NULL DEFAULT 0;
```

```
ALTER TABLE users
ADD COLUMN approved tinyint(1) NOT NULL
DEFAULT 0 FIRST;
```

- השאלתה גרמה להוספה העמודה "approved" לסוף רשימת העמודות.
- כדי להוסיף עמודה לתחילת רשימת העמודות השתמש בדирקטיבה **FIRST**.

כדי לשנות את שמה של עמודה נוספת נוסף לפקודה **ALTER TABLE** את הפקודה **.CHANGE**

נשנה את שם העמודה מ-**approved** ל-**agreed**:

שינוי שם הטבלה ומחיקה

נשתמש בפקודה **RENAME** כדי לשנות את שם הטבלה:

```
RENAME TABLE users TO people;
```

נשתמש בפקודה **DROP TABLE** כדי למחוק טבלה:

```
DROP TABLE people;
```

```
ALTER TABLE users
```

```
CHANGE approved agreed tinyint(1) UNSIGNED NOT
NULL DEFAULT 0;
```

מחיקת עמודה

כדי למחוק עמודה נוספת נוסף לפקודה **ALTER TABLE** את הפקודה **DROP** את הפקודה **.COLUMN**

נמחוק את העמודה **agreed**:

```
ALTER TABLE users
```

```
DROP COLUMN agreed;
```

שים לבו当你尝试在 MySQL 中立即删除列时，可能无法执行此操作。直到你先更改列名，才能删除它。

Re - Index's:

מבוא

אינדקסים של SQL משמשים **לאיתור מהיר של רשומות בטבלה** במסד הנתונים.

לא שימוש באינדקס השאיתה צריכה להתחל **לקראאתחלת הטבלה** ואז לעבור **שורה** - שורה על כל הטללה **עד שהיא מוצאת את המידע המבוקש**. לעומת זאת, כאשר **משתמשים באינדקס המנוע** של מסד הנתונים **שולף** את **המידע ישיר מהרשומות הרלוונטיות**. התוצאה היא **חסכון עצום במשאבים** ודמן טעינה נמוך בהרבה.

למה להשתמש באינדקסים?

- ❖ כדי **להיאץ את מהירות השליפה** של הנתונים **כאשר משתמשים ב-WHERE**.
- ❖ **איחוד טבלאות** באמצעות **JOIN** געשה על **עמודות האינדקס**.

מס'יע **לאופטימיזר optimizer** **לבחר** את **אופן הביצוע החסכוני** **bijouter במשאבים**.
האופטימיזר הוא פונקציה של MySQL המיעיל את **השאלות** וכאשר קיימים גוברים הסיכוי שהוא יבחר בו כדי **להיאץ את השאלתה**.

איך עובד אינדקס?

אינדקס **מכיל רשימה של מצביעים pointers** שככל אחד מהם הוא **קישור קצר לרשותה בטבלה המקורית**.

בגלל **שהמצביע הוא קצר בהרבה מהמידע** ברשומות המקוריות הוא **מקל על מסד הנתונים** **לאתר את המידע המבוקש**.

על מה צריך לחשוף כמשמעותים אינדקס?

אינדקס **יעיל** **צריך להיות קצר ויחודי**:

אינדקס **قصير** **מקל על** **מנוע מסד הנתונים** **לסנן את התוצאות הרצויות**. לכן, **נעדייף להוסיף אינדקסים על** **עמודות מספריות** (וגם INT) **ולא על** **עמודות המכילות טקסטים ארוכים**.

עדיף להגדיר אינדקס ייחודי כדי **להקל על השליפה**.

אומנם **יצירת אינדקסים** **מאייצה את החיפוש אבל מפריעה לעדכן וכתייה** לפיכך כדאי **לשימוש באינדקס רק כשਬאתם חיבטים**.
לדוגמא, כדאי **לגדיר אינדקס** את **עמודת המזהה הייחודי** (id) **בכל טבלה**.

אילו אינדקסים ישם בטבלה?

הפקודה SHOW INDEX FROM ביצורף שם הטבלה מציג את האינדקסים השיכלים לטבלה:

```
SHOW INDEX FROM table_name;
```

כיצד מוחקים אינדקס מטבלה?

הפקודה DROP INDEX ON מוחקת אינדקס מטבלה:

```
ALTER TABLE table_name
DROP INDEX index_name
```

איך להוסיף אינדקס לטבלה?

```
CREATE INDEX index_name
ON table_name(column_name)
```

רצוי מאוד להשתמש באינדקס על عمودה בעלת ערכים ייחודיים, דוגמת عمودת המזהים הייחודיים של הטבלה id. כדי להגדיר את האינדקס כבעל ערכים ייחודיים משתמש במילת המפתח UNIQUE.

```
CREATE UNIQUE INDEX index_name
ON table_name(column_name)
```

נוסיף טבלה בשם cars כדי שונכו להציג על ה-id שלו מהטבלה שלנו:

```
CREATE TABLE IF NOT EXISTS `cars`
(
  `cars_id` int(6) NOT NULL AUTO_INCREMENT,
  `cars_model` varchar(20) DEFAULT NULL,
  `license_renewal` date DEFAULT NULL,
  PRIMARY KEY (`cars_id`)
);
```

מין נתונים לטבלה cars:

```
INSERT INTO `cars`(`cars_model`, `license_renewal`)
VALUES
('Talbot', '2013-10-01'),
('Sussita', '2014-01-01');
```

נוסיף עמודה חדשה של מפתחות זרים לטבלה המקורית. שמה של העמודה refer_cars_id, וחוسب מאווד שהיא תואם בדיק את התוכנות של העמודה שהיא מצביעה עליה (סוג נתונים, קיוד וכו'ב), ושבירית המחדל תהיה NULL.

```
ALTER TABLE workers
ADD COLUMN refer_cars_id int(6)
DEFAULT NULL;
```

אחריו שיצרנו את העמודה של המפתחות הזרים, נגידר על אייזו עמודה היא מצביעה.
באופן הבא:

```
ALTER TABLE workers
ADD FOREIGN KEY (refer_cars_id)
REFERENCES cars(cars_id)
```

foreign key

פתח זר(foreign key)

נניח שתהיה טבלה (**workers**) שתכיל את שמות העובדים בחברה ופרטים נוספים.

נדמיין מצב שאנו צריכים צרכי מידע נוסף והפעם על המכוניות של העובדים (דוגמת מודל ותאריך חידוש רישיון הרכב), מה שיפוך את הטבלה שלנו למრובת עמודות מדי וקשה לתחזוקה.

פתרונות?

הפתרון, מרכיב משני שלבים:

- הוספה טבלה חדשה (**cars**), שתכיל את פרטי הרכבים.
- הוספה עמודה מפתח זר לטבלה הקיימת (**workers**), שתחזק על המזהה הייחודי (**id**) של הרכב בטבלה (**cars**).

העמודה שתצבי על המזהה הייחודי (**id**) מורכבת מ מפתחות זרים (foreign key), שהם **ערכים מספריים שמצביעים על המזהים הייחודיים** (**foreign key**) בטבלה האחרת.

עכשו ניתן את הערכים לעמודות refer_cars_id בטבלה workers. כשבן את הנתונים לעמודות המפתחות הזרים, נקבע לחסיף רק מהו שקיימים בטבלה שעליה מחייבים. בדוגמה, קיימים רק רכבים שהזירה שלהם הוא 1 או 2, ובהתאם, אילו הערכים היחידים שאינם NULL, שנכל לחסיף לעמודה. מגבילה זו קשורה בכך שמספרם זר מחייב אותם להזין רק את ערכי המזהה הקיימים, ואם ננסה להזין ערכים שאינם קיימים נקבל שגיאה.

מכיוון שהעמודות הקשורות, אנו רצים לעדכן את הנתונים במקרה של שינוי או מחיקה, ועשויים זאת כך:

```
ALTER TABLE `cars`
MODIFY `cars_id`
INT(11) UNSIGNED NOT NULL AUTO_INCREMENT
```

```
ALTER TABLE `workers`
ADD FOREIGN KEY (refer_cars_id)
REFERENCES cars(cars_id)
ON DELETE CASCADE
ON UPDATE CASCADE
```

במסדי נתונים מסוימים, צריכים להגדיר את טור המפתחות הזרים ואת טור המזהים - **UNSIGNED** לפני שניתן להשתמש במפתחות זרים. כך עושים זאת:

```
ALTER TABLE `workers`
MODIFY `refer_cars_id`
INT(11) UNSIGNED NOT NULL
```

```
SELECT * FROM table_name
ORDER BY שם_העמודה DESC
```

ORDER BY

נסדר את התוצאות מהגבוהה לנמוכה או ORDER BY. ה**פוך** באמצעות הפוקודה

שם העמודה מייצג את העמודה שלפיה נסדר את התוצאות. לאחר כן, ניתן לחסיף **DESC** כדי לדרוש תוצאות בסדר יורד. ברירת המחדל היא סדר עולה, וכן ניתן לציין את זה באמצעות **ASC**.

דוגמה לקוד הבאה שולפת מbasics הנתונים את 5 התוצאות האחרונות בסדר יורד:

```
SELECT * FROM workers
ORDER BY workers_id DESC
LIMIT 5
```

והתוצאה היא הבאה:

workers_phone1	workers_name	workers_id
02-2233337	Yehuda	13
02-2888887	Koresh	12
07-7888887	Zrubavel	11
08-8666666	Shlomo	10
03-7333335	Eliezer	9

workers_phone1	workers_name	workers_id
02-2000002	Asher	5
03-7333335	Eliezer	9
0544-444333	Gershon	4
02-2888887	Koresh	12
09-99999998	Metushelah	6

ניתן גם לסדר את התוצאות לפי סדר הא"ב. לדוגמה, השאלה הבאה שולפת את 5 התוצאות הראשונות מבסיס הנתונים לפי סדר ה-abc.

```
SELECT * FROM workers
ORDER BY workers_name ASC
LIMIT 5
```

workers_city	workers_name	workers_phone2	workers_phone1	workers_id
Bat Yam	Eliezer	NULL	03-7333335	9
Bat Yam	Yirmiyahu	NULL	03-3333337	3
Beer-sheva	Gershon	NULL	0544-444333	4
Hakrayot	Moshe	NULL	09-9888887	1
Hakrayot	Sheshet	NULL	09-9777778	8
Hakrayot	Yoshiahu	NULL	09-8999997	7
Hakrayot	Zrubavel	NULL	07-7888887	11
Jerusalem	Asher	NULL	02-2000002	5
Kiryat Gat	Shlomo	NULL	08-8666666	10
Makabim	Yehuda	NULL	02-2233337	13
Natanyahu	Metushelah	NULL	09-99999998	6
Ramat Hasharon	Yechezkel	NULL	04-4888887	2
Rosh haayin	Koresh	NULL	02-2888887	12

ניתן גם לסדר את התוצאות לפי יותר מעמודה אחת. לדוגמה, השאלה הבאה מסדרת את התוצאות לפי שם העיר, ובתוך זה לפי שם העובד.

```
SELECT * FROM workers
ORDER BY workers_city ASC, workers_name ASC
```

התוצאה היא הבאה, קודם סידור לפי שם העיר, ובתוך כך סידור לפי שמות:

```
SELECT DISTINCT workers_city
FROM workers
```

השאילה מה תחזיר כל שם עיר פעם אחת בלבד, אפילו אם שם העיר מופיע במספר שורות שונים.

Select DISTINCT

כשמשתמשים בשאלחת **SELECT** על **עמודת נתונים שעווים לקבל את אותה תוצאה יותר מפעם אחת**.

כדי להציג כל ערך / תוצאה פעם אחת בלבד, משתמשים במילת **DISTINCT**. המפתחה

עדכן מונה בטבלה

אחד הלקוחות שלי ביחס לדעת מה מספר הצפויות בתכנים שהוא מעלה. מכיוון שמספר הצפויות בכל אחד מהתכנים אצלך יכול להגיע למאות אלפי רציתי פתרון פשוט שלא יהיה הכרוך בניהול של טבלה ענקית שבה כל שורה מתעדת צפיה אחת באחד התכנים. הפתרון בו בחרתי הוא של מונה שמתעדכן בכל פעם שימושה נכנס לדף.

```

CREATE TABLE `posts_views` (
    `post_id` int(11) UNSIGNED DEFAULT 0,
    `views` int(11) UNSIGNED DEFAULT 0,
    `created_at` datetime DEFAULT NOW(),
    `updated_at` datetime DEFAULT NULL ON UPDATE
    NOW()
) ENGINE=InnoDB DEFAULT CHARSET=UTF8;

ALTER TABLE `posts_views`
ADD PRIMARY KEY (`post_id`);

ALTER TABLE `posts_views`
MODIFY `post_id` int(11) UNSIGNED NOT NULL;

```

כך נראה השאלה:

```

INSERT INTO posts_views (post_id, views,
created_at, updated_at)
VALUES (1, 1, NOW(), NOW())
ON DUPLICATE KEY UPDATE views = views + 1;

```

במילים: תרשום צפיה 1 למאמר שהזאהה הייחודי שלו, ה-*id* שלו, בטבלת הפוטיטים הוא 1.

אחרי הפעם הראשונה בה הרצית את השאלה קיבלתי:

	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export
					1	1	2020-05-27 15:20:52	2020-05-27 15:20:52	

מספר הצפויות 1.

הרציתי שוב את אותה השאלה בדיק.

	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export
					1	2	2020-05-27 15:20:52	2020-05-27 15:21:45	

ה קיבלתי 2 צפיות כי מונה הצפויות מתעדכן באופן אוטומטי.

	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export
					1	3	2020-05-27 15:20:52	2020-05-27 15:22:16	

פעם שלישית עם אותה השאלה בדיק:

	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export
					1	95	1	2020-05-27 15:22:46	2020-05-27 15:22:46

וראיתי שנוספה רשומה חדשה עם צפיה 1 למאמר שמספרו 95.

הוסףתי צפיה 1 למאמר אחר שה-*id* שלו בטבלת הפוטיטים הוא 95.

	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export
					1	95	1	2020-05-27 15:22:46	2020-05-27 15:22:46

וראיתי שנוספה רשומה חדשה עם צפיה 1 למאמר שמספרו 95.

3 צפיות.

אחרי **שסיימת את העבודה** הלה כך מרצה שהוא **ביקש לישם את המונה על כל סוג הtechnics** (מאמרים, מדריכים, פוסטיםים בבלוג), **אבל הטבלה לעיל לא יכולה לתמוך מענה מכיוון שסוגי הtechnics מפוזרים בין טבלאות שונות** (**טבלה למדריכים, טבלה לפוסטים וטבלה לבלוגים**), **ובטבלאות שונות יכולם להיות technics עם אותו מספר אינדקס**.

הפתרון שמצאתו היה להוסיף עמודה מסוג enum שתכיל את סוג המאמר ולעשות אינדקס משותף ייחודי UNIQUE על שתי העמודות, סוג המאמר ומספר האינדקס שלו.

זה מבנה הטבלה לאחר השינוי:

הນוטי רשותה למאמר מסוים שמספרו 101.

```
INSERT INTO `articles_views`
VALUES (NULL, 'news', 101, 1, NOW(), NOW())
ON DUPLICATE KEY UPDATE views = views + 1;
```

	id	article_type	article_type_id	views	created_at	updated_at
	1	news	101	1	2020-06-27 15:24:23	2020-06-27 15:24:23

המונה הרואה 1

הרצתי את אותה שאלתה בדיק בפעם השנייה:

	id	article_type	article_type_id	views	created_at	updated_at
	1	news	101	2	2020-06-27 15:24:23	2020-06-27 15:25:08

למונה נוספים 1 באופן אוטומטי, וכך עליה מספר הצפיות ל-2.

הນוטי צפיה לבלוג אחר שמספרו 42.

	id	article_type	article_type_id	views	created_at	updated_at
	1	news	101	2	2020-06-27 15:24:23	2020-06-27 15:25:08

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at
	2	blog	42	1	2020-06-27 15:25:41	2020-06-27 15:25:41

	id	article_type	article_type_id	views	created_at	updated_at

<tbl_r

CRUD With Json

89

MySQL מצויד ביכולת לטפל בסוג הנתונים **JSON** שמעניק **גמישות רבה, בדומה לסקמות של NoSQL.**

נוסיף את הטבלה

מוסיף את הטבלה הבאה למסד הנתונים:

```
CREATE TABLE `cars` (
  `id` int(11) UNSIGNED NOT NULL,
  `name` varchar(255) DEFAULT ,
  `attributes` JSON DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=UTF8;

ALTER TABLE `cars`
ADD PRIMARY KEY (`id`);

ALTER TABLE `cars`
MODIFY `id` int(11) UNSIGNED NOT NULL
AUTO_INCREMENT;
```

- סוג הנתונים הוא **JSON**.

- הערך ביריתת המוחלט היחיד שהוא יכול לקבל הוא **NULL** (באוטו-ידולות).

חונה של **JSON** בשיטה הידנית עשויה להיות מייגעת ורგישה לטועינו, לפיכך נעדרף להשתמש בפונקציה **JSON_OBJECT** כדי להזין את המידע באופן שיטתי ומוסדר.

הפונקציה **JSON_OBJECT** מקבלת מפתחות וערכים מופרדים בפסיקים ע"ש התמחיבה:

```
JSON_OBJECT(key1, value1, key2, value2, ...)
```

ומחזירה אובייקט **JSON**.

```
INSERT INTO `cars`(`id`, `name`, `data`)
VALUES
(NULL, 'BMW R80G/S',
JSON_OBJECT(
  "class",
  "dual-sport",
  "wheelbase" ,
  "57.7 in",
  "weight",
  186,
  "is_motorcycle",
  true,
  "steering wheel",
  null,
  "dimensions",
  JSON_ARRAY(
    JSON_OBJECT(
      "length",
      2230,
      "width",
      820,
      "height",
      1150
    ),
    {
      "class": "dual-sport",
      "weight": 186,
      "wheelbase": "57.7 in",
      "dimensions": [
        {
          "width": 820,
          "height": 1150,
          "length": 2230
        }
      ],
      "is_motorcycle": true,
      "steering wheel": null
    }
));
```

הפונקציה **JSON_ARRAY** יוצרת רשימות של פריטים מופרדים בפסיקים.

חשיבות הקפיד על הפורמט

המודרך:

❖ את אובייקט **- JSON** **חובה להקיף בסוגרים מסולסים.**

❖ את **המפתחות** **צריך לשים בין מרכאות כפולות.**

את סוג

true, false, null **חייבים לכתוב באוטו-ידולות.**

❖ את **הסוג מחרוזת מקיפים מרכאות כפולות.**

❖ האובייקט יכול להכיל **מערכות** **כמו גם אובייקטים מקוונים של JSON.**

כיצד להזין רשומה חדשה הכוללת **NOJSON?**

דרך אחת, היא לרשום את ה-JSON על פי הסכמה המדויקת (json.org)

```
INSERT INTO `cars`(`id`, `name`, `data`)
VALUES
(NULL, 'BMW R80G/S', '{"class":"dual-sport","wheelbase":"57.7 in", "weight":186, "is_motorcycle":true, "steering wheel":null, "dimensions":[2230,820,1150]}');
```

לאחר ההזנה, כשאנו שומד עם הסמן בתוך השדה אנו יכולים לראות את ה-JSON שהזנו:

```
{
  "class": "dual-sport",
  "weight": 186,
  "wheelbase": "57.7 in",
  "dimensions": [
    2230,
    820,
    1150
  ],
  "is_motorcycle": true,
  "steering wheel": null
}
```

השאלתה הבאה מדגימה 3 רמות של אובייקטים מקוונים:

```
INSERT INTO `cars`(`id`, `name`, `data`)
```

```
VALUES
```

```
(NULL, 'BMW R80G/S',
```

```
JSON_OBJECT(
```

```
  "class",
```

```
  "dual-sport",
```

```
  "wheelbase" ,
```

```
  "57.7 in",
```

```
  "weight",
```

```
  186,
```

```
  "is_motorcycle",
```

```
  true,
```

```
  "steering wheel",
```

```
  null,
```

```
  "dimensions",
```

```
  JSON_ARRAY(
```

```
    JSON_OBJECT(
```

```
      "length",
```

```
      2230,
```

```
      "width",
```

```
      820,
```

```
      "height",
```

```
      1150
    )
  )
);
```

```
{
  "class": "dual-sport",
  "weight": 186,
  "wheelbase": "57.7 in",
  "dimensions": [
    2230,
    820,
    1150
  ],
  "is_motorcycle": true,
  "steering wheel": null
}
```

```
{
  "class": "dual-sport",
  "weight": 186,
  "wheelbase": "57.7 in",
  "dimensions": [
    {
      "width": 820,
      "height": 1150,
      "length": 2230
    }
  ],
  "is_motorcycle": true,
  "steering wheel": null
}
```

90

```
{
    "class": "dual-sport",
    "weight": 186,
    "wheelbase": "57.7 in",
    "dimensions": {
        "width": 820,
        "height": 1150,
        "length": 2230
    },
    "is_motorcycle": true,
    "steering wheel": null
}
```

כיצד לעדכן רשומה?

נשתמש בפונקציה **JSON_SET** כדי לעדכן שדה מסוג JSON.

נוסיף מפתח חדש בזמן שנעדכן רשומה קיימת:

```
UPDATE `cars`
SET `data` = JSON_SET(
    `data`,
    '$.dimensions.seat_height' ,
    860
)
WHERE `id` = 5
```

כדי לעדכן מפתח קיים:

```
UPDATE `cars`
SET `data` = JSON_SET(
    `data`,
    '$.dimensions.seat_height' ,
    null
)
WHERE `id` = 5
```

כיצד למחוק רשומה?

נשתמש בפונקציה **JSON_REMOVE** כדי למחוק רשומה ע"פ מפתח:

```
UPDATE `cars`
SET `data` = JSON_REMOVE(
    `data` ,
    '$.dimensions.seat_height'
)
WHERE `id` = 5
```

```
INSERT INTO `cars`(`id`, `name`, `data`)
VALUES
(NULL, 'BMW R80G/S',
JSON_OBJECT(
    "class",
    "dual-sport",
    "wheelbase" ,
    "57.7 in",
    "weight",
    186,
    "is_motorcycle",
    true,
    "steering wheel",
    null,
    "dimensions",
    JSON_MERGE(
        JSON_OBJECT("length",2230),
        JSON_OBJECT("width",820),
        JSON_OBJECT("height",1150)
    )
));
)
```

כיצד לקרוא רשומה?

כדי לקרוא רשומה של JSON מסגד הנתונים חשוב קודם להבין את היריעון **path expression**.

הקריאה בפועל מוצעת באמצעות הפונקציה **JSON_EXTRACT** המתקבלת מהביטוי **path .expression**

```
SELECT * FROM `cars`
WHERE
`name` = 'BMW R80G/S'
AND JSON_EXTRACT(`data` , '$.weight') > 180
AND JSON_EXTRACT(`data` , '$.dimensions.width') > 800
AND JSON_EXTRACT(`data` , '$.wheelbase') = '57.7 in'
AND JSON_EXTRACT(`data` , '$.is_motorcycle') = true;
```

את **JSON_EXTRACT** אפשר להחליף בחז' כדי להפוך את השאלתה לקריאה נוספת.

```
SELECT * FROM `cars`
WHERE
`name` = 'BMW R80G/S'
AND `data` -> '$.weight' > 180
AND `data` -> '$.dimensions.width' > 800
AND `data` -> '$.wheelbase' = '57.7 in'
AND `data` -> '$.is_motorcycle' = true
AND JSON_EXTRACT(`data` , '$.class') LIKE '%sport%';
```

Join + Pivot

מתי נשתמש בטבלת ציר? (pivot)

התשובה הפשוטה, **נשתמש בטבלת ציר (pivot) כדי להציג מידע שמקורו במספר טבלאות.**
והתשובה המדויקת היא **שניהם חייבים לשימוש בטבלת ציר במצב שבו נרצה לבטא יחסים של "many to many".**

לדוגמא, כשהרצתה שאותו אדם יכול להיות הבעלים של יותר מכוכנת אחת.
בדוגמה, אנחנו מעוניינים להציג מידע בין טבלת workers וטבלת cars במטרה **לשאול איזה מכונית שייכת לכל עובד.**
כדי לעשות זאת נוצר טבלת ציר שמכילה את **מזהה העובד** ואת **מזהה המכונית** **שמשוכנת אליו.**

שלבי העבודה

1. ייצור 3 טבלאות

טבלת workers שמכליה מידע אודות העובדים

```
-- Table `car_worker`  
  
CREATE TABLE `car_worker` (  
    `id` int(10) NOT NULL AUTO_INCREMENT,  
    `car` int(10) NOT NULL,  
    `worker` int(10) NOT NULL,  
    PRIMARY KEY (`id`)  
);  
  
INSERT INTO `car_worker` (`id`, `car`, `worker`) VALUES  
(1, 1, 1),  
(2, 1, 3),  
(3, 2, 3),  
(4, 2, 5);  
  
-- Table `cars`  
  
CREATE TABLE `cars` (  
    `id` int(10) NOT NULL AUTO_INCREMENT,  
    `name` varchar(80) DEFAULT NULL,  
    PRIMARY KEY (`id`)  
);  
  
INSERT INTO `cars` (`id`, `name`) VALUES  
(1, 'Talbot'),  
(2, 'Sussita'),  
(3, 'Subaru');
```

טבלת car_worker שמצוינה את המידע>About העובדים עם המידע>About המכוניות שבבעלותם

```
-- Table `workers`  
  
CREATE TABLE `workers` (  
    `id` int(10) NOT NULL AUTO_INCREMENT,  
    `name` varchar(80) DEFAULT NULL,  
    PRIMARY KEY (`id`)  
);  
  
INSERT INTO `workers` (`id`, `name`) VALUES  
(1, 'Moshe'),  
(2, 'Yechezkel'),  
(3, 'Yirmiyahu'),  
(4, 'Gershon'),  
(5, 'Asher');
```

2. נכתוב שאלות ששאלות איזה מכונית שייכת לכל עובד

3. נריץ את השאלה

כשריצית השאלתה נקבל את התוצאה הבאה:

car_name	worker_name
Talbot	Moshe
Subaru	Moshe
Sussita	Yirmiyahu
Sussita	Asher

שים לב, משה מחזק 2 מכוניות, ואת זה אנחנו יכולים לבטא בקלות הודות לשימוש בטבלת ציר, pivot.

```
SELECT w.name AS worker_name, c.name AS car_name  
FROM car_worker AS cw  
  
INNER JOIN workers AS w  
ON w.id = cw.worker  
  
INNER JOIN cars AS c  
ON c.id = cw.car
```

- ב-JOIN הראשון נוצרת טבלת car_worker כשהעומודה המשותפת היא id העובד.
- ב-JOIN השני נוצרת טבלת cars לatable car_worker כשהעומודה המשותפת היא id המכונית.
- סה"כ, קיבלנו צירוף של שיטות הטבלאות.

פתרונות

ב MySQL -קיהםת פונקציית , (IF שמקבלת שלושה פרמטרים: הביטוי, מה לעשות במידה והתנאי מתקיים, ומה לעשות אם התנאי נכשל.

כך נראה המחבר:

```
SELECT `workers_name`, `workers_date_added`,
IF(`workers_date_added`<'2011-06-08', "Vatik", "Tzair")
AS Vetek
FROM workers
```

השאילתה יוצרת עמודה חדשה בשם Vetek, ומוצבה בו ערך של "Vatik" אם תאריך ה入职 העובד ("workers_date_added") קטן מ-'2011-06-08', ואם התנאי אינו מתקיים מוצב בעמודה הערך "Tzair".

או התוצאות:

workers_name	workers_date_added	Vetek
Moshe	2011-06-07	Vatik
Yechezkel	2011-06-07	Vatik
Yirmiyahu	2011-06-07	Vatik
Gershon	2011-06-08	Tzair
Asher	2011-06-08	Tzair
Metushelah	2011-06-08	Tzair
Yoshiahu	2011-06-09	Tzair
Sheshet	2011-06-09	Tzair
Eliezer	2011-06-09	Tzair
Shlomo	2011-06-09	Tzair
Zrubavel	2011-06-21	Tzair
Koresh	2011-06-06	Vatik
Yehuda	2011-06-21	Tzair

כדי לבחור בין יותר משתי חלופות משתמשים ב-CASE. המחבר הבא נותן תוצאה דומה ל-IF ELSEIF THEN-JavaScript בֆוטות PHP ו-.

```
IF(expression ,expr_true, expr_false)
```

הדוגמה פשוטה ביותר :

```
SELECT IF(1=4,'true','false');
```

שאלה האם 1 שווה ל-4, ומחזירה true או false. כשתניריצו את התרגיל, תקבלו false

CASE

```
WHEN meet condition 1 THEN a
WHEN meet condition 2 THEN b
ELSE c
END
```

כך נראה שאלתה לדוגמה שמשתמשת ב-case:

```
SELECT `workers_name`, `workers_city`,
(CASE
WHEN `workers_city` = 'Jerusalem' THEN 'Merkaz'
WHEN `workers_city` = 'Kiryat Gat' THEN 'Darom'
WHEN `workers_city` = 'Nahariya' THEN 'Tzafon'
ELSE 'unknown'
END
) as Area
From workers
```

השאילתה מוסיפה עמודה נוספת Area. כאשר `workers_city` הוא 'Jerusalem' הוא 'Merkaz' כאשר עיר 'Kiryat Gat' מקבל את העיר 'Darom' ובכל מקרה אחר 'unknown'.

workers_name	workers_city	Area
Moshe	Nahariya	Tzafon
Yechezkel	Ramat Hasharon	unknown
Yirmiyahu	Bat Yam	unknown
Gershon	Beer-sheva	unknown
Asher	Jerusalem	Merkaz
Metushelah	Natanya	unknown
Yoshiahu	Hakrayot	unknown
Sheshet	Hakrayot	unknown
Eliezer	Bat Yam	unknown
Shlomo	Kiryat Gat	Darom
Zrubavel	Hakrayot	unknown
Koresh	Rosh haayin	unknown
Yehuda	Makabim	unknown

ביטוי רגולרי

ב MySQL - ניתן לבצע שאילתות שימושיים רגולרים באמצעות האופרטור REGEXP , אופרטור LIKEabil הרבה יותר מאופטור LIKEabil הרבה פחות מרכיבי של PHP לטפל בביטויים רגולרים , והוא ברහלט כל שחייב להיות בארגז הכלים של כל מפתח.

התו המוחך נקודה .

התו המוחך (מטה-תגית) מייצג כל תו שהוא. אך אם נרצה לחפש ביטוי שמתחלף ב-3 תוים כלשהם שאחריהם shos, נכתב זאת כך:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP '^...shon';
```

והתוצאה:

Gershon

דריך יותר אלגנטית לכתוב חזרות היא באמצעות ציון מספר החזרות בין סוגרים מסוללים. כך נראית אותה השאילהה באמצעותה שבסוגרים מסוללים:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP '.{3}shon';
```

והתוצאה:

Gershon

שאילתיה נוספת נוספת שבסוגרים מסוללים:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP '.{3}sh.{2}';
```

והתוצאה:

Gershon, Sheshet

שאילתות שמתחלות ומסתיימות בתו מסויים

לחפש ביטוי שמתחלף ב-u. שימו לב שב-MySQL אין משמעות לאותיות קטנות או גדולות.

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP '^y';
```

והתוצאה:

Yechezkel, Yirmiyahu, Yoshiahu, Yehuda

כדי לחפש ביטויים מסוימים ב-u:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP 'yahu$';
```

והתוצאה:

Yirmiyahu

ט ט של תווים

כדי לבחור מוט של תוים, מציינים את ט התווים בין סוגרים מרובעים. לדוגמה, charachim להתחילה את השאלה באחד התווים hik abcdefgjihk

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP "[abcdefgjihk]";
```

והתוצאה:

Gershon, Asher, Eliezer, Koresh

אפשר לכתוב את השאלה בדרך יותר אלגנטית אם מפרידים בין התו הראשון והאחרון בסוקן מפרד. לדוגמה, השאלה הבאה:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP "^[a-k]";
```

והתוצאה:

Gershon, Asher, Eliezer, Koresh

זו שאלהה שבודחת רק מחרוזות שכוללות אותיות עבריות:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP "[נ-ע]";
```

בחירה בין אפשרויות

כדי לבחור בין אפשרויות מציבים | בין האפשרויות. לדוגמה, gershon או yirmiyahu .ycibb כב' כב' .gershononly|yirmiyahu

השאילהה:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` REGEXP 'gershon|yirmiyahu';
```

תוצאות:

Yirmiyahu, Gershon

Like Wildcard

הדרך הפשוטה ביותר לשימוש ב WHERE - היא כדי למצוא התאמה מדויקת לביטוי, בנוסף ניתן למצוא התאמה פחות מדויקת אם משתמשים במילת המפתח LIKE ובאחד מה wildcards, % או _.

% מסוג wildcard

הWildcard % מתייחס לכל תו וולספּר של תוים.

לדוגמא, אם רציתם התאמה למילים שתחילות ב-Y בלבד, מציבים את % אחרי Y כדי לציין כל דבר:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` LIKE 'Y%'
```

והתוצאה היא כל השמות שתחילים ב-Y:

Yechezkel, Yirmiyahu, Yoshiahu, Yehuda

אם אונחנו מעוניינים רק במחוזות שתחילות ב-yahu, נקדים את ה-wildcard % ב-yahu. כך תראה השאילתה:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` LIKE "%yahu"
```

ו結果ו:

Yirmiyahu, Yoshiahu

אפשר גם למקם את ה-wildcard באמצע המחרוזת להסתrema:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` LIKE "%sh%"
```

מה שיחזיר כל מה שמכיל sh בתחילת המחרוזת, באמצע המחרוזת או בסופה.

Moshe, Gershon, Asher, Metushelah, Yoshiyahu

וגם:

Sheshet, Shlomo, Koresh

_ מסוג wildcard

הWildcard _ מצייןתו אחד בדיקון, זה יכול להיות כל תו.

לדוגמא, אם רציתם מילה שתחילתה ב-yו ואחריו ?, אפשר לנסה שאליתה זו:

```
SELECT `workers_name` FROM
`workers` WHERE `workers_name` LIKE 'yo_?'
```

ותוצאה:

Yosi, Yoni

אפשר להשתמש ב מספר פעמים במחוזת להסתrema, לדוגמא:

```
SELECT `workers_name`
FROM `workers`
WHERE `workers_name` LIKE '_o_i'
```

ותוצאה:

Yosi, Yoni

NOT LIKE

כפי שיש LIKE, התוצאה המשלימה מתקבלת באמצעות NOT. לדוגמה:

```
SELECT `workers_name` FROM `workers`
WHERE `workers_name` NOT LIKE "%sh%"
```

Re-View

אחרי שבמדריכים הקודמים למדנו כיצד לצרף 2 טבלאות ויתר לשאלתה אחת באמצעות JOIN כדי לדעת שהשימוש ב- JOIN הוא יקר מהבינה משאים וכן עלול להכביר על המערכת ולהאט את האפליקציה שעבדת עלייה כל כך קשה.

אחת האפשרויות לפטור את הבעיה הוא שימוש ב-view.

? מה זה view

View הוא שאלתה שאנו שומרים לשימוש בעתיד וגם טבלה וירטואלית. משתמשים בו-view כדי לפשט שאלות מורכבות (דוגמת JOIN על מספר טבלאות).

לשם הדוגמה, נעבד מוקדם נתונים הכלול 2 טבלאות, cars ו-workers:

```
CREATE TABLE IF NOT EXISTS `cars` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `model` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `cars` (`id`, `model`) VALUES
(1, 'Tesla'),
(2, 'Sussita');
```

```
CREATE TABLE IF NOT EXISTS `workers` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `name` varchar(25) DEFAULT NULL,
  `salary` int(6) DEFAULT '0',
  `cars_id` int(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `workers` (`id`, `name`, `salary`, `cars_id`)
VALUES
(1, 'Moshe', 50000, 2),
(2, 'Yechezkel', 18000, NULL),
(3, 'Yirmiyahu', 12500, 1),
(4, 'Gershon', 15900, NULL),
(5, 'Asher', 13500, NULL);
```

נשתמש ב-JOIN כדי לקבל מידע מטבלות cars ו-workers ביחד באמצעות שאלתה:

וכדי להפוך את השאלתה ל-view (לטבלה וירטואלית) נשתמש בתחבירו:

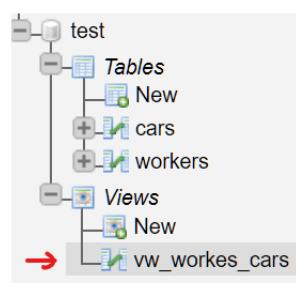
CREATE VIEW [view_name] AS [השאלתה]

```
SELECT w.id, w.name, w.salary, c.model
FROM workers AS w
JOIN cars AS c
ON c.id = w.cars_id
```

חזר לדוגמה שלם:

```
CREATE VIEW vw_workers_cars
AS
SELECT w.id, w.name, w.salary, c.model
FROM workers AS w
JOIN cars AS c
ON c.id = w.cars_id
```

אחרי שייצרו את ה-view, אנחנו יכולים לראות אותו ברשימה ה-views של מסד הנתונים:



ניתן לבצע שאליתה על ה-view כמו על טבלה רגילה:

```
SELECT * FROM vw_workes_cars
```

ניתן למחוק את כל ה-view:

```
DROP VIEW vw_workes_cars
```

המחלוקת לא תשפיע על הטבלאות המקוריות.

שני דברים שכדי לשים אליהם לב:

- המוצאה היא כמו JOIN אבל במחיר נמוך יותר מבחןת צריכה משאבי מעכנת
- ה-view מתאים עם הטבלאות המקוריות. במידה ונסנה את המידע בטבלאות המקוריות ישנהו המידע גם בטבלת ה-view.

כיצד לשנות view?

אחרי שיצרנו את ה-view אנחנו יכולים לשנות אותו.

לעדיין:

```
UPDATE vw_workes_cars
SET
    name = 'Moris'
WHERE
    id=1;
```

- זהירות, זה עדכן גם הסבלה המקורית.

מצפיר - תת שאלתה

תת-שאלתה (sub query) היא שאלתה בתוך שאלתה. משתמשים בתת-שאלתה כ恕ורי מידע נוסף לפני שוענים על השאלה העיקרית. במצב זה, קודם שואלים את השאלה שמספקת את המידע הנוסף, זו תת-השאלת, ואז עונאים על השאלה עצמה.

לדוגמה,

אם נרצה לדעת מיהם העובדים שלהם מספר מקסימלי של ילדים? לא נוכל לענות על השאלה זו לפני שנענה על השאלה מהו המספר המקסימלי של ילדים שיש לכל עובד. לכן, השאלה מהו המספר המקסימלי של ילדים תהיה תת-השאלת שאלתנו.

אפשר לכתוב זאת כך:

תת-שאלתה (בלי שנענה אליה לא נוכל לענות על השאלה העיקרית) : מהו המספר המקסימלי של ילדים?
השאלת העיקרית (אחרי שעננו על תת השאלת אנחנו יכולים לענות אליה) : מיהם העובדים שלהם מספר מקסימלי של ילדים?

תת-שאלתה הולכת למעשה

אבל מה שאנו צריכים זו תשובה אוטומטית, ולכן נשים את השאלה מהו המספר המקסימלי של ילדים בתרו תת-שאלתה. את תת השאלת נשים בין סוגרים. כך:

```
SELECT `workers_id` , `workers_name` , `num_childs`
FROM `workers`
WHERE `num_childs` = (
    SELECT MAX( `num_childs` )
    FROM `workers`
)
```

והתשובה:

num_childs	workers_name	workers_id
4	Yoshiahu	7
4	Koresh	12

לטיכם, כדי לבצע שאלת הcolałת תת-שאלת, נשים את תת-השאלת בתוך סוגרים עגולים בשאלת העיקרית.

```
SELECT MAX(`num_childs`)
FROM `workers`
```

והתשובה: 4

אחרי שקיבלנו תשובה שהמספר המקסימלי של ילדים הוא 4, אנחנו יכולים לשאול את השאלה "איזה מהעובד יש 4 ילדים?"

```
SELECT `workers_id` , `workers_name` , `num_childs`
FROM `workers`
WHERE `num_childs` = 4
```

והתשובה:

num_childs	workers_name	workers_id
4	Yoshiahu	7
4	Koresh	12

תזכיר - פונקציות מקבצות

פונקציות מקבצות (Aggregate functions) מוחזירות מידע אודות קבוצות של רשומות. לדוגמה, **מספר השורות** שמחזירה השאלתה, **ממוצע עמודה מסוימת**, **וערכים מינימליים ומקסימליים** בעמודה.

COUNT()

אם נרצה לקבל את מספר הרשומות בטבלה משתמש בפונקציה- **COUNT()**, עם שם עמודה . לדוגמה .

כשהשתמשים בפונקציות מקבצות, צריך לשים לבן-NULL הוא לא אפס, אלא העדר ערך. המשמעות של זה היא הרבה מפנוי שפונקציות מקבצות לא יתחשבו ברשומות שערךם NULL. לדוגמה, כשנريץ את הפונקציה **COUNT()** על העמודה **num_childs**, הפונקציה תחזיר רק את מספר השורות שאין NULL. נריץ את השאלתה:

```
SELECT COUNT( num_childs )
FROM `workers`
```

שתחזיר:

COUNT(num_childs)
11

11 מספרם של העמודות שאין NULL.

```
SELECT COUNT( * )
FROM `workers`
```

והתוצאה בהתאם:

COUNT(*)
13

כמו לכל עמודה רגילה, ניתן לתת גם לעמודה שהיא תוצאה של פונקציות מקבצות, שם דידוחי באמצעות AS:

```
SELECT COUNT( * ) AS count
FROM `workers`
```

והתוצאה בהתאם:

count
13

פונקציות מקבצות נוספת

פונקציית מקבצת מועלות נוספת ה**AVG**, שמחזיר את ממוצע העמודה שעליה הוא פועל.

MIN(`num_childs`)	MAX(`num_childs`)
0	4

והתוצאה בהתאם:

```
SELECT AVG( `num_childs` )
FROM `workers`
```

והתוצאה:

AVG(`num_childs`)
2.0

הפונקציה **SUM()** מוחזירה את סכום העמודה. לדוגמה:

```
SELECT SUM( `num_childs` )
FROM `workers`
```

והתוצאה:

SUM(`num_childs`)
22

22 הוא סך כל הילדים של כל עובדי החברה.

הפונקציות **MIN**-**MAX** מוחזירות את הערך המינימלי והמקסימלי בהתאם. לדוגמה:

```
SELECT MIN(`num_childs`), MAX(`num_childs`)
FROM `workers`
```

99

שימוש ב- HAVING

כדי להציג את הקבוצות המוחזרות לנו מפונקציות מקבצות, נשתמש ב-**HAVING**.
לדוגמא, כדי להציג את תוצאות השאלה שלנו לכולם שיש להם שלושה ילדים או פחות מכך, משתמש בשאלתה הבאה:

```
SELECT `num_childs` , COUNT( * ) AS count
FROM `workers`
GROUP BY `num_childs`
HAVING `num_childs` <= 3
```

התוצאה:

num_childs	count
3	2
2	3
1	2
0	2

```
SELECT `num_childs` , COUNT( * ) AS count
FROM `workers`
GROUP BY `num_childs`
```

התוצאה:

num_childs	count
4	2
3	2
2	3
1	2
0	2

מה מספר הקבוצות בשאלתה?

כדי לקבל את מספר הקבוצות, נשתמש ב-**DISTINCT**, מפני שהוא מחזיר רק תוצאות ייחודיות, ומועלם מהתוצאות כפולות.

```
SELECT COUNT( DISTINCT `num_childs` ) AS
num_childs
FROM `workers`
```

התוצאה:

num_childs
5

5 קבוצות.

שאילתה ראשונה, מוצאת את שמות שלושת העובדים הראשונים דפ' ה-5:

```
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` <= 3
```

שאילתה שנייה, מוצאת את שמות העובדים שה-id שלהם 8 עד 10:

```
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` > 7 AND `workers_id` < 11
```

כדי לאחד בין השאלות השתמש ב-UNION, כך נראית השאילתה המאוחדרת:

```
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` <= 3
UNION
SELECT `workers_id`, `workers_name`
FROM `workers`
WHERE `workers_id` > 7 AND `workers_id` < 11
```

Re - Union

הרבה פעמים נרצה להציג מספר שאילותות, ולקבל סט אחד של תוצאות. לדוגמה, **שמות 3 העובדים הראשונים** בראשימת העובדים, **וגם שמות העובדים 8 עד 10**. לצורך כך משתמש ב UNION - כדי **שייחבר בין השאלות**.

workers_name	workers_id
Moshe	1
Yekhezkel	2
Yirmiyahu	3
Sheshet	8
Eliezer	9
Shlomo	10

חשוב לציין כל רשותה (שורה בסיסד הנתונים) רק פעם אחת. לדוגמה, השאילתה הבאה שורצוה לשילוף את העובדים 1-3, וגם את אילו שגירים בקריות.

```
SELECT `workers_id`, `workers_name`, `workers_city` FROM
`workers`
WHERE `workers_id` <= 3
UNION
SELECT `workers_id`, `workers_name`, `workers_city` FROM
`workers`
WHERE `workers_city`='hakrayot'
```

והתוצאה:

workers_city	workers_name	workers_id
Hakrayot	Moshe	1
Ramat Hasharon	Yekhezkel	2
Bat Yam	Yirmiyahu	3
Hakrayot	Sheshet	8
Hakrayot	Eliezer	9
Hakrayot	Shlomo	10

שים לב, ש-**Moshe** אינו מוצג פעמיים למרות שהוא אחד משלושת העובדים הראשונים ווגם תונשב הקרים.

Re - Join

אחת **הבעיות המרכזיות** של מסדי נתונים קלאסיים נובעת מהבעיה **שאי אפשר לצפות מראש** את כל הطالאות שיידרשו.

לדוגמה, חברת בתחלת דרכה **תחליל** עם ניהול **טבלת עובדים**, ועם **התבססות** החברה תצטרך להויסיפ **טבלת רכבים** ו**טבלת ספקים** **ולקוחות** וכו'ב.

אבל **יכן ניתן לקשר בין הטבלאות**? לדוגמה, **יכן ניתן לקשר בין העובדים לרכבים** שבהם הם **מהගים?** התשובה היא שמדובר **בשימוש בצירוף טבלאות** או **JOIN**.

מבנה ה JOIN – כולל את המרכיבים הבאים:

SELECT על **שמות העמודות**
FROM שהוא **שם הטבלה הראשונה**
INNER JOIN או **LEFT JOIN** **או** **ON** **שם הטבלה השנייה**
ON הוא **התנאי לצירוף הטבלאות.**

כדי לקבל רק את השורות שקיימות בשתי הטבלאות המקוריות משתמש ב-**INNER JOIN**. לדוגמה:

```
SELECT `workers`.`workers_name` , `workers`.`cars_id`
, `cars`.`cars_model`
FROM `workers`
INNER JOIN `cars`
ON `workers`.`cars_id` = `cars`.`cars_id`
```

כדי לקבל את כל השורות של הטבלה הראשונה משתמש ב-**LEFT JOIN**. לדוגמה:

```
SELECT `workers`.`workers_name` , `workers`.`cars_id`
, `cars`.`cars_model`
FROM `workers`
LEFT JOIN `cars`
ON `workers`.`cars_id` = `cars`.`cars_id`
```

הכל לעשות JOIN בין טבלאות שיש בינהן יותר משדה משותף אחד

שאנו עושים צירוף של טבלאות באמצעות **JOIN**, ויש לנו יותר משדה אחד משותף, אנחנו צריכים לציין את כל השדות המשותפים כדי שהצירוף יתבצע כהלכה.

בפועל, עושים את זה על ידי צירוף השדות באמצעות **AND**, ראה את ההדגשה בשאלתה שלנו:

```
SELECT sales.area,sales.state,name,sales
FROM sales
JOIN stores
ON
stores.state = sales.state
AND
stores.area = sales.area
```

השאילה השגיה

השאילה השגיה עשו **JOIN** על עמודה אחת בלבד.

```
SELECT sales.area,sales.state,name,sales
FROM sales
JOIN stores
ON stores.state = sales.state
```

נותנת לנו הרבה יותר רשומות מחייבות בגל שניות לעשות **JOIN** על עמודה אחת בלבד, אך למעשה יש שתי מעודדות משותפות בין הטבלאות אותן מעוניינים לצירוף.

השאילה הנכונה

השאילה הנכונה עשו **JOIN** על כל העמודות המשותפות, שתיים במקרה זה, כפי שניתן לראות להלן:

```
SELECT sales.area,sales.state,name,sales
FROM sales
JOIN stores
ON
stores.state = sales.state
AND
stores.area = sales.area
```

והתוצאה היא התוצאה הנכונה. שורה אחת לכל חנות.

עקרונות וטיפים לՏיכום

איך לא ליפול בפח כנתוניים מסד נתונים MySQL ?

במציאות טבלאות נתונים יכולות להכיל כמה עזומה של מידע וקל לאבד את הראש כמו גם לגרום למסד נתונים איטי. המפתחה למניעת הבלגן הוא **תכנון של מסד הנתונים ע"פ מספר עקרונות מנחים**:

- ❖ כל טבלה צריכה **لتאר נושא אחד בלבד**. לדוגמה, טבלה של עובדים צריכה להכיל את **השם, תאריך הלידה, מספר הטלפון**. אבל היא **לא צריכה להכיל את פרטי הרכב שבו נוהג העובד את הרכבים נקבע בטבלה אחרת**.
- ❖ אין טעם **לפרק יותר מדי את הנתוניים**. לדוגמה, **לפתח טבלה לעובדים, טבלה למספר טלפונים של העובדים וטבלה לתאריכי לידיה**.
- ❖ כל רשומה של טבלה צריכה שייה לה **מספר סידורי primary key ייחודי שיזהה אותה מיתר הרשומות באותה טבלה**. בדומה, **למספר זההות שלך שהוא ייחודי רק לך**.
- ❖ כל תא בטבלה צריך **لتאר מידע אוטומי (שאינו רישימה)**. אם אתה רואה שאחד התאים מכיל רישימה של **ערכים** זהرمز לך שציריך **לפתח טבלה אחרת**. לדוגמה, אם **בטבלת העובדים אחת העמודות היא בונוסים** שקיבל העובד אז אתה עלול להתਪנות ולשים באותו תא את **רשימת הבונוסים אותו קיבל** העובד מדי חדש. עדיף לך **ליצור טבלה חדשה של בונוסים** עם **הפניות באמצעות מפתחות זרים לעובדים**.
- ❖ אם אתה צריך **להקליד שוב ושוב את אותו שם ברשומות שונות** זה **סימן שאתה צריך לפתח טבלה אחרת**. לדוגמה, אם **בטבלת העובדים** אתה צריך שוב ושוב **להקליד** עבור עובדים שונים את **אותו מודל מכונית** אז כדאי לך **לפתח טבלת מכוניות ולקשר אליה באמצעות מפתחות זרים**.

השאיפה = היא ליצור מסד נתונים מנוירמל. מה שאומר **שכל טבלה מתארת נושא אחד ולא נפילה** בפח של **פירוק יתר, לכל רשומה יש מספר ייחודי** וכל אחד מהתאים **בטבלאות מכילים מידע אוטומי**. **מן פנוי מידע מנוירמל הוא קל יותר להבין** וגם **מסד הנתוניים רץ מהר יותר**.

יחסים בין טבלאות במסד נתונים

מוד נתונים SQL הוא יחס relational מה שאומר שיש **יחסים** בין הטבלאות. המידע בטבלה אחת מתקשר באופןם השונים שלם **ל מידע בטבלה אחרת.**

ישנם שלושה סוגי נפוצים של יחסים:

- + **יחס של אחד לאחד (1:1)** - כל רשומה בטבלה א' מתייחסת לרשותה אחת בלבד בטבלה ב', וכל רשומה בטבלה ב' מתייחסת לרשותה אחת בלבד בטבלה א'.

לדוגמה, **כל** משרה באתר AllJobs **שייכת** לסדרת שירותי מוצר **אחד בלבד בלבד.**

לדוגמה, **המשרה** **Full Stuck - AllJob** **שדרנית התקבל אליה** **שייכת** **לשירות מוצר** **FullStuck**.

בהתאם, **בטבלת העבודות (jobs)** במסד הנתונים **נמצאים** **שמות סדרות השירותים ומספר מוצר** **ובטבלת השירותים** **בטעות** **מזהה הייחודי בטבלת עבודות CurrentJobs** **FullStuck** **השרות אליה המשרה שייכת.**

Jobs	
ID	NAME
1	FullStuck
2	Gaming
3	Software

Current Jobs		
ID	NAME	JobID
1	Full Stuck	1
2	AllJobs	
2	Gaming LS	2
3	Software SV	3

- + **יחס של אחד לרבים - (1:N)** כל רשומה בטבלה א' יכולה להתיחס לאפס רשומות, אחת או מספר רשומות בטבלה ב' אבל לא להיפר.

לדוגמה, **יחס** **בין קונים להזמנות שהם עושים.** אותו קונה יכול לעשות מספר הזמנות.

בדוגמה הבאה, שתי טבלאות **buyers** (**קונים**) ו- **orders** (**הזמנות**). הקונים יכולים להזמין יותר מפעם אחת. בכל פעם שקונה עונה הזמנה נוספת רשומה לטבלה **orders** היכולת את המזהה הייחודי של הקונה.

buyers	
id	name
1	Moshe
2	David
3	Joe

orders		
id	date	buyer_id
1	2020-01-01	1
2	2020-01-03	3
3	2020-01-12	3
4	2020-01-12	3

שימוש בטבלת ציר ביחס של רבים לרבים

אותו עובד יכול להשתמש ביותר ממכונית אחת, וגם אותה מכונית יכולה לשמש מספר עובדים.

זו אינדיקציה ברורה לחיסום של **רבים לרבים** מסגד הנתונים.

טבלת צירpivot מאפשרת לנו **לקשר בין עובדים לרכבים**.

בטבלה **עמודה אחת תוקדש למזהה הייחודי של העובד ועמודה שנייה למזהה הייחודי של המכונית**.

בדוגמה הבאה, הטבלה **cars_workers** היא **טבלת הציר הקשורת בין הרכיבים בטבלה cars ו-worker**

cars_workers

+-----+	+-----+	
id	cars_id	workers_id
1	2	1
2	2	5
3	1	3
4	3	3

cars

+-----+	+-----+
id	model
1	Talbot
2	Sussita
3	Mustang

workers

+-----+	+-----+
id	name
1	Moshe
2	Yechezkel
3	Yirmiyahu
4	Gershon
5	Asher

אולי נראה לך בעיתוי להבין באיזה מכונית מושע כל עובד. נשתמש ב-**JOIN** (לקריאה מדרין **JOIN**) כדי לצרף את הטבלאות:

```
SELECT cw.id AS cw_id, name, w.id AS w_id,
model, c.id AS c_id
FROM cars_workers AS cw
JOIN workers AS w
ON w.id = cw.workers_id
JOIN cars AS c
ON c.id = cw.cars_id;
```

התוצאות:

+-----+	+-----+	+-----+	+-----+	+-----+
cw_id	name	w_id	model	c_id
1	Moshe	1	Sussita	2
2	Asher	5	Sussita	2
3	Yirmiyahu	3	Talbot	1
4	Yirmiyahu	3	Mustang	3

איתור רשומות ללא מיפוי

105

```
SELECT w.id AS w_id, name, cw.id AS cw_id
FROM workers AS w
LEFT JOIN cars_workers AS cw
ON w.id = cw.workers_id;
```

אחרי שנחנו יודעים אילו עובדים קיבלו רכבים אולי השאלה הבאה היא אילו עובדים לא קיבלו רכבים כי אנחנו לא רוצים לקפץ אף אחד.

כתבו שאלתה שתאפשר לנו **לראות את רשימת כל העובדים בין אם קיבלו רכבים או שלא:**

w_id	name	cw_id
1	Moshe	1
5	Asher	2
3	Yirmiyahu	3
3	Yirmiyahu	4
2	Yechezkel	NULL
4	Gershon	NULL

אפשר לשפר את השאלתה עוד יותר אם נתמקד באוטם העובדים שלא מופיעים בטבלת הלקוח באמצעות הוספת הפקה הבאה לשאלתה:

```
WHERE cw.id IS NULL
```

כתבו את השאלתה במלואה:

```
SELECT w.id AS w_id, name
FROM workers AS w
LEFT JOIN cars_workers AS cw
ON w.id = cw.workers_id
WHERE cw.id IS NULL;
```

התוצאה:

w_id	name
2	Yechezkel
4	Gershon

מדריך תכנון בסיס נתונים – קשרים בין היחסיות

סימנים מוסכמים

בכדי לעשוט את מה שאני עומד להסביר מיד מספיק ניר, אך במקרה **لتאר תרשימים באופן מקצועית** יותר מומלץ להשתמש בתוכנות כמו **Visio** או **Smart Draw**.

ישות – מסומנת **במלבן**.

밀ת קשר בין היחסיות – מסומנת **במעויין**.

קשר יחיד – מסומן **בחץ**.

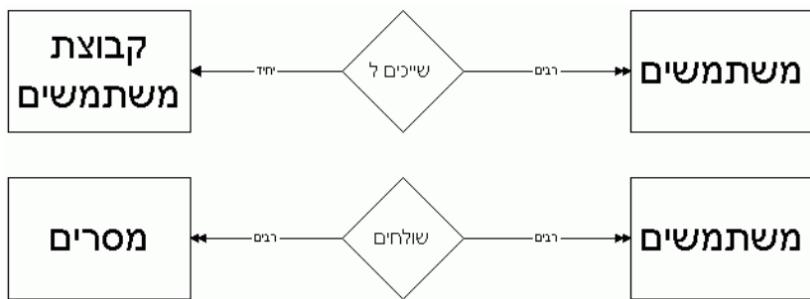
קשר רבים – מסומן **בחץ עם ראש כפול**.

איך מרים קשרים בין היחסיות? ישנו 3 סוגי קשרים:

קשר יחיד ליחיד – למשל **מפעל** שבו יש **מנהל אחד ועובד אחד**. **הקשר** הוא **סוג יחיד ליחיד**: רק **מנהל אחד** ו**עובד אחד**.

קשר יחיד לרבים – ניקח את אותו מפעול. אבל הפעם נניח שיש **מספר בלתי מוגבל של עובדים**. הפעם הקשר **בין המנהל לשות עובד** יהיה **יחיד לרבים**: רק **מנהל אחד אבל כמה עובדים**.

קשר רבים לרבים – זהו **הקשר המסובך ביותר**. נחזור שוב למפעול. נניח שבמפעול יש **מחלקות**, וכל **מחלקה יש מנהל מחלקה**. הפעם הקשר **בין מנהל מחלקה לבין שות עובד יהיה רבים**: **כמה מנהלי מחלקות שבהן כמה עובדים**.



דוגמאות ב揆וטות ישות

��וצות משתמשים ומשתמשים: כאן הקשר הוא **יחיד** ל**��וצת משתמשים**. כמה **��וצת משתמשים** אחד. **משתמש** אחד לא יכול להיות רשום לכמה **��וצות משתמשים** או **��וצת משתמשים אחד מכילה כמה משתמשים**.

משתמשים ומסרים: כאן הקשר הוא **רבים לרבים**. יכולם להיות **כמה מסרים למשתמש אחד**, ומ**משתמש אחד יכול לשלוח מסר לכמה משתמשים**.

��וצת משתמשים ומסרים: זהה דוגמא ל**שתיות שאין בניהן קשר**.

דרכי פעולה

דרך הפעולה עם קשר יחיד לרבים:

כשיש **קשר יחיד לרבים** (נדcir: **אחד מישות מסויימת וכמה מישות אחרות**) אנו שותלים שדה מפתח זה בטבלת הרבים. למשל: בבסיס נתונים בית ספר, יש **טבלת כיתות** (מספר כיתה, שם) ו**טבלת תלמידים** (מספר, שם). כדי **שנדע איזה תלמיד שיש לאיזו כיתה** נוסיף **עמודת מספר כיתה בטבלת תלמיד**. כך נוכל **לדעת לאיזו כיתה שיר כל תלמיד**.

דרך הפעולה עם קשר רבים לרבים:

דרך הפעולה עם קשר רבים לרבים מסובכת מעט יותר –علינו **לפצל טבלאות**. נניח כי **כל תלמיד יכול ללמידה בכמה כיתות**. ניצור **טבלה חדשה הכוללת מספר, מספר תלמיד, מספר כיתה**. **טבלת הלקוחה הכלול** גם היא מספר ושם. ככה, למשל, **ונכל לשולף את כל הכתובות שבנה לומד התלמיד**.
כלומר: בהתמודדות עם קשר **"רבים לרבים"** אנו יוצרים **טבלה חדשה וגורמים להיווצרות 2 קשרים** של **"חיד לרבים"** שאיתם, כמובן, **אנו יודעים להתמודד**.

משתמשים (Users)					
כתובת	שם משפחה	שם	מספר קבוצה	מספר אינדקס ללא כפיליות	

קבוצת משתמשים (User Group)					
		מקום מרכז'	עיר	מספר אינדקס ללא כפיליות	

מסרים(Messages)					
	תוקן	נושא	מ. שולח	מספר אינדקס ללא כפיליות	

מסרים_תפוצה(Tfo_Messages)					
		מס. נמען	מס. מסר	מספר אינדקס ללא כפיליות	

יישום קשרים בטבלאות

שיםו לב שקיימת קשר הרבים לרבים, נוספה טבלת "מסרים_תפוצה" והדבר מאפשר לנו להכניס שורה לכל נמען. ככה כשנרצה לדעת איזה מסר נשלח לאילו נמענים, נשלף טבלת "מסרים_תפוצה" את כל הרשומות בעלות אותו "מס. מסר".
חסוב מכך להגדיר "אינדקס ללא כפיליות" – זה מה שמורה למסד הנתונים להבדיל בין מפתח ראשי של אותה טבלה למפתח זה.

חוקי נרמול טבלאות

מהו נרמול טבלאות ולמה זה טוב?

על רגל אחת, הנרמול הוא **תהליך שיפור יעילות בסיס הנתונים שלו**. וعصין, כמובן, עבור היחסים מפורטים. חוקי הנרמול הם ארבעה חוקים הגיוניים ולוגיים שמאפשרים לנו **لتacen את בסיס הנתונים שלו בצורה הגיונית, למנוע בעיות בעתייד, לצמצם את שטח הדיסק שבסיס הנתונים תופס ואולי הסיבה החשובה מכלום: להביא את בסיס הנתונים במצב שהוא קל לעובוד אותו בעדרת SQL.**

סה"כ, קיימים 5 חוקי נרמול ועוד חוק אחד, **קצת פחות رسمي: שלושת הראשונים בסיסיים, הרביעי מורכב** קצת יותר והשימוש בחמישי נדיר מאוד.

החוק הראשון מטפל בקשר של יחיד-ליך(1:1)

החוקים 2 ו-3 מטפלים בקשר של יחיד לרבים(N:1)

החוקים 4 ו-5 מטפלים בקשר של רבים לרבים(N:N)

מושגים בסיסיים:

- + **שדה :** נתון בסיסי המאפיין **ישות מסוימת**. לדוגמה: **הנתון הבסיסי מס' טלפון** יופיע את הישות **"תלמיד"**.
- + **מפתח ראשי (FK) :** שדה המזהה באופן חד ערכי רשומה מסוימת. למשל: **מספר תעודה זהה**.
- + **מפתח מורכב :** אוסף של שדות המהווים ביחד מפתח ראשי. למשל: **בטבת גולים**, **מספר המשחק והזמן המציג את>bבקעת הגול** יהו יחד **מפתח מורכב**.
- + **מפתח זר(FK) :** שדה המצביע על מפתח ראשי בטבלה אחרת. למשל: **בטבת תלמיד**, **מס' הклассה** הוא מפתח זר.
- + **רשומה :** אוסף של שדות, שיש בהם **קשר**. למשל: **רשומות תלמיד** כולל **מספר ת.ז, שם, כתובה ועוד**.

חוק הנרמול הראשון (1NF)

First Normal Form (1NF):

A table is in first normal form (1NF) if and only if every **attribute** in the table is **atomic**.

טבלה **מנורמלת** בرمמה הראשונה אם ורק אם **כל רכיב** בה הוא **אטומי**.

למשל, **טבלת תלמידים**, העמודה "שם התלמיד" תהיה מפורקת לשתי עמודות: **שם פרטי** ו**שם משפחה**.

חוק זה מאפשר לנו **לעשות מניפולציות בקלות** על הנתונים בעזרת SQL **לא צורך בפועלות מכוערות** על מה שנתקבל לאחר השאלתה.

נזכיר עם הדוגמה שלנו: אם הטבלה **לא הייתה מנורמלת**, היינו צריכים **알גוריתם מסורבל ומיותר שידע להבחין** בין השם הפרטי לשם המשפחה:

1. **לצמצם רוחחים משני הצדדים**: "שלום שמחון" => "שלום שמחון"

2. **לצמצם כל יותר מרוחח אחד לרווח אחד**: "שלום שמחון" => "שלום שמחון"

3. **איתור המיקום של הרווח**. מתחילה המחרוזת עד המיקום של הרווח זה השם הפרטי. ממיקום הרווח עד סוף המחרוזת זה שם המשפחה.

ובודאי שמתם לב **שזה מיותר**.

במקרים רבים, הטבלה **לא תהיה מנורמלת** בرمמה הראשונה, (1NF) למשל **במצב שמצריך שמירת תאሪיך ועתה לידיה**. לפי חוק הנרמול, אנו **צריכים עמודה נפרדת לתאריך** הלידה ועמוד **נפרד לשעת הלידה**. ברוב המקרים, **התאריך והשעה נשמרים בשדה אחד**.

חוק הנרמול השני (2NF)

Second Normal Form (2NF):

A table is in second normal form (2NF) if and only if **every non-key attribute** in the table
is **functionally dependent** on the **entire primary key**.

טבלה **מנורמלת** בرمמה השנייה אם ורק אם היא **מנורמלת** בرمמה הראשונה **ולכל מאפיין שהוא לא חלק מהמפתח הראשי, נשען על המפתח הראשי – על כל חלקיו**.

חוק זה נוגע רק למצבים של **מפתח מורכב**.

דוגמא:

בטבלה **ניסויים במעבדה**, מס' המדун והתאריך והשעה בהם נערכ הניסוי מהווים מפתח. מצב של **שדה המכיל את שם המדун** יגרום

לטבלה להיות **לא מנורמלת** בرمמה **השניה**, כיון שם המדун לא נשען על כל חלקו המפתח הראשי, אלא רק על מס' המדун.

במצב זה, **ניתן** טבלה מדענים ובטבלה הניסויים יהיה שדה מפתח זר (FK) **שיכלול את מספר המדען**.

מס' המדען ניתן לנוי גישה לשם המדען, באמצעות **שאילתת JOIN**.

הערה:

כפי שהסבירנו קודם, מצב שבו **טבלה אינה מנורמלת** בرمמה הראשונה **נפוץ מאד**, ולכן לטעון כי **טבלה מנורמלת** בرمמה **השניה גם אם אינה מנורמלת** בرمמה **הראשונה**.

חוק הנרמול השלישי (3NF)

Third Normal Form (3NF):

A table is in third normal form (3NF) if and only if no **non-key attribute** in the table is **functionally dependent** on **any other non-key attribute**.

טבלה **מנורמלת ברמה השלישית** אם ורק אם **היא מנורמלת ברמה השנייה** וכל רכיב בה שאינו המפתח הראשי לא תלוי ברכיב אחר שאינו המפתח הראשי.

חוק זה נותן תשובה **למקרים קלאסיים של תכנון לקיי**. למשל: **ברשותת תלמיד מס' היכיתה** מהוות מפתח זר. מצב בו **יתווסף** שדה של שם היכיתה אינו רצוי, כיוון שאז נוצר מצב של תלות בין שדות שאינם המפתח הראשי: שדה שם היכיתה **תלויה בשדה מס' היכיתה**. במצב כה **נוריד** את שדה שם היכיתה, וניגש אליו **דרך מס' היכיתה** באמצעות **שאילתת JOIN**

כשטבלה מנורמלת ברמה השלישית, היא מנורמלת לפוי BCNF.

חוק הנרמול הרביעי (4NF)

Fourth Normal Form (4NF):

A table is in Fourth Normal Form (4NF) if and only if it does **not represent two or more independent many-to-many relationships**.

טבלה **מנורמלת ברמה הרביעית** אם ורק אם **היא מנורמלת ברמה השלישית (BCNF)** והוא **לא מייצגת יותר מקשר אחד של רבים-לربים**.

טבלת קשר תייצג קשר אחד של רבים-לרבים ולא יותר, אחרת **היא לא תהיה מנורמלת ברמה הרביעית**. החוק בא בכך **להסוך** **במקומות בסיסי הנטונים** (מיד יבוא הסבר...).

דוגמאות:

Teachers (**TeacherID**, **TeacherName**)

Classes (**ClassID**, **ClassName**)

Professions (**ProfessionID**, **ProfessionName**)

נניח **שלכל מורה יש כמה כיתות שהוא מלמד**, וכמה מקצועות בהם הוא מתמחה. יצרת טבלה אחת שתציג את **קשרי הרבים-לרבים** **לרבים** **תהייה שגיה**:

TeacherClassProfession (**TeacherID**, **ClassID**, **ProfessionID**)

הטבלה שגיה כיוון **שיהיה בחבוץ מקום כי יהיושורות בהן מצוין רק מקצוע ושורות בהן מצוינת רק כיתה**.
לפי חוק הנרמול הרביעי **יש ליצור שתי טבלאות שונות שיציגו את קשרי הרבים-לרבים**.

חוק הנרמול החמישי (5NF)

5

Fifth Normal Form (5NF):

A table is in Fifth Normal Form (5NF) if it is **4NF** and its **information content cannot be reconstructed from several tables containing fewer attributes.**

טבלה מנורמלת בرمה החמישית אם ורק אם היא **מנורמלת בرمה הרביעית** ותוכנה לא יכולה מושג מכמה טבלאות המכילות מספר נמור של מאפיינים.

במילים אחרות: טבלה מנורמלת בرمה החמישית אם אין מידע הקשור **ישירות לטבלה** אשר **יכול להיות מושג באמצעות יותר משאלת איחוד (Join) אחת** הקשור **לטבלה רבים-רבבים**. חוק הנרמול החמישי **מבטיח כי בהם מידע הקשור לטבלה מסוימת לא יהיה משוחזר "בחלקים"** מטבלאות שונות.

Teachers (TeacherID, TeacherName)

Classes (ClassID, ClassName, TeacherID)

Professions (ProfessionID, ProfessionName, ClassID)

בדוגמה הבאה, כדי **לראות באילו מקצועות מלמד המורה**, נצטרך **לעשות שתי שאלות איחוד**. זה, כמובן, מצב **לא רצוי מבחינה לוגית ומבחןת ביצועים**. כדי שטבלה **Teachers** תהיה מנורמלת בرمה החמישית, נצטרך **لتאר את המבנה כך:**

Teachers (TeacherID, TeacherName)

Classes (ClassID, ClassName)

Professions (ProfessionID, ProfessionName)

ClassTeacher (ClassID, TeacherID)

ProfessionTeacher (ProfessionID, TeacherID)

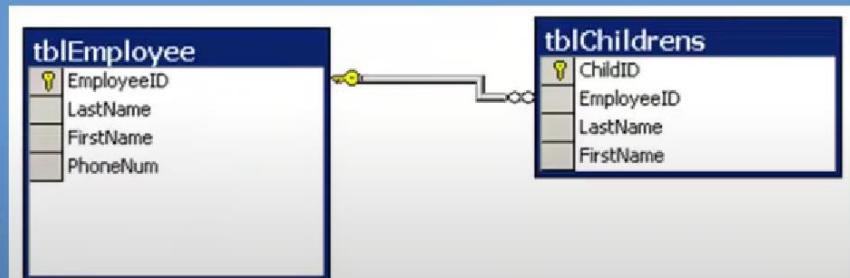
במצב זהה **כדי לדעת מידע הקשור ישירות לטבלה Teachers** כמו למשל **מקצועות הלימוד** נצטרך **לעבור דרך טבלת הרביבים-rabim ProfesseionTeacher** ולא דרך טבלה אחרת שלא קשורה לנושא שאלת האיחוד.

הערה: במקרים רבים **معدיפים מתכנני בסיס הנתונים שהטבלה תישאר מנורמלת בرمה הרביעית בלבד** ולא בرمה החמישית. למעשה, חוק הנרמול החמישי הוא **תיאורטי בעיקר**.

צורה נורמלית ראשונה - 1NF – המשך

הפתרונות:

הפרדת כל תכונה שעשויה לקבל קבוצת ערכים לקבוצת ישות נפרדת .



צורה נורמלית ראשונה – 1NF
כל שדה שאינו חלק מהמפתח חייב להיות תלוי באופן
מלא בכל שדה המפתח.

כאשר יש מפתח פשוט בעל תכונה אחת , אין קושי לקיים דרישת זו . כאשר המפתח מורכב , יש לבדוק אם כל תכונה אינה תלולה בחלק מהמפתח בלבד .



דוגמה:
המפתח במקרה זה מורכב משני שדות : מספר סטודנט ומספר קורס .
השדה שם הסטודנט תלוי רק במספר הסטודנט , השדה שם הקורס תלוי רק במספר הקורס , השדה ציון תלוי באופן מלא בכל שדות המפתח (ציון לסטודנט בקורס).
ולכן : שם סטודנט ושם קורס לא תלויים במלוא המפתח ולכן אינם צריכים להיות טרלבּל.

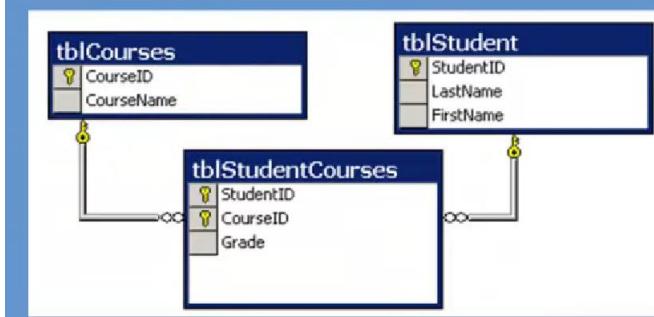
תלוי רק ב CourseID (חלק מפתח) (CourseName) לכן шагו!

צורה נורמלית ראשונה – 1NF

פתרונות:

לפצל לשולשה טבלאות : סטודנטים , קורסים וסטודנטים בקורס .

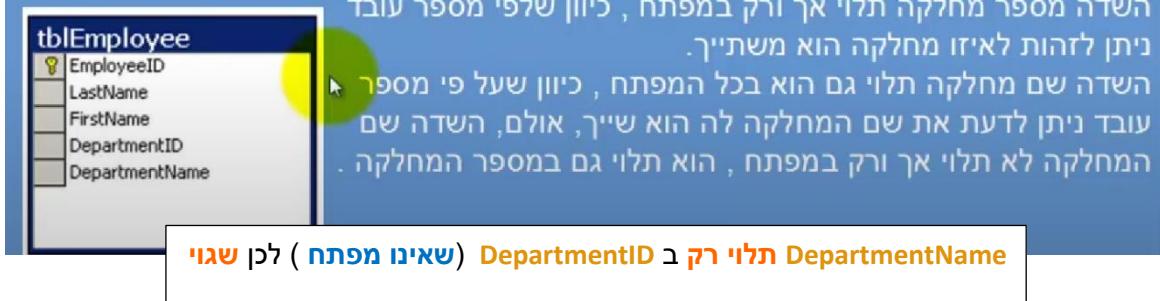
אם לסטודנטים יש מספר ציונים בקורס יתווסף למפתח מספר ציון .



צורה נורמלית ראשונה – 3NF

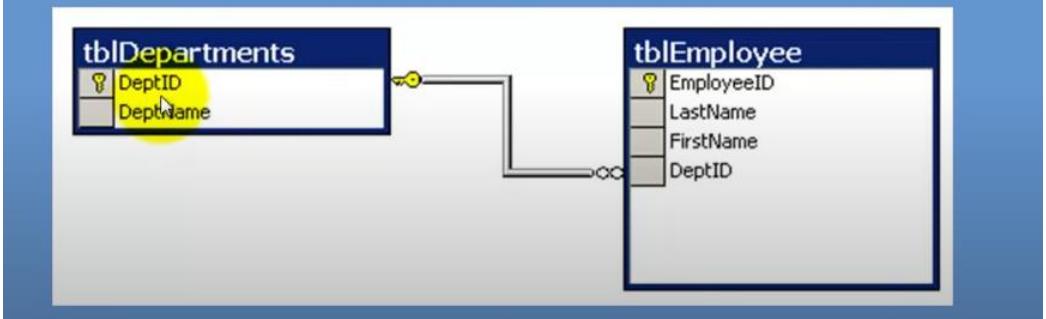
כל שדה שאינו שדה מפתח (או חלק מהמפתח),
אסור שייה תליי בשדה אחר שאינו שדה מפתח.

דוגמה:



פתרונות:

יש לפצל את הטבלה לשתי טבלאות:
 טבלה עובדים: מספר עובד, שם פרטי, שם משפחה, מספר מחלקת.
 טבלה מחלקות: מספר מחלקת, שם מחלקת.



נירמול: תרגיל מסכם

1. ת"ז חוקר←שם פרטי, שם משפחה

טעות נפוצה:
חוקר←ת"ז חוקר, שם פרטי, שם משפחה

הסבר הטעות: אין בطالאות עמודה בשם "חוקר" (אין לנו שום דבר בשם זה). יש רק עמודות בשם "ת"ז חוקר", "שם פרטי" ו"שם משפחה", כאשר מה שמצויה חוקר זה הת"ז שלו.

במכון המחבר "צפת למחקרים הגלילי" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושל משפחה. במכון מתנהלים פרויקטים של מחקר. לכל פרויקט מחקר יש קוד מצחה, תיאור, מנהל שהוא אחד החוקרם, ותקציב שנתי שՏסכומו משתנה מיד' שנה. חוץ מהמנהל, לכל פרויקט משיכים חוקרם. חוקר יכול להשתียר ליותר מפרויקט אחד. החוקרם מפרסמים ביחיד מאמרם. בנוסף לרישימת חוקרם שהם מחברי המאמר, לכל מאמר יש קוד מאמר מצחה, כותרת, וכתב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר מאמר אחד.



1. ת"ז חוקר←שם פרטי, שם משפחה

2. קוד פרויקט←תיאור, ת"ז מנהל

טעות נפוצה:
פרויקט←קוד פרויקט

הסבר הטעות: כמו קודם: אין בطالאות עמודה בשם "פרויקט" (אין לנו שום דבר בשם זה). פרויקט מצوها בطالאות לפי הקוד שלו.

במכון המחבר "צפת למחקרים הגלילי" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושל משפחה. במכון מתנהלים פרויקטים של מחקר. לכל פרויקט מחקר יש קוד מצחה, תיאור, מנהל שהוא אחד החוקרם, ותקציב שנתי שՏסכומו משתנה מיד' שנה. חוץ מהמנהל, לכל פרויקט משיכים חוקרם. חוקר יכול להשתียר ליותר מפרויקט אחד. החוקרם מפרסמים ביחיד מאמרם. בנוסף לרישימת חוקרם שהם מחברי המאמר, לכל מאמר יש קוד מאמר מצחה, כותרת, וכתב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר מאמר אחד.

1. ת"ז חוקר←שם פרטי, שם משפחה

2. קוד פרויקט←תיאור, ת"ז מנהל

טעות נפוצה:
פרויקט←מנהל

הסבר הטעות: המנהל הוא אחד החוקרם. חוקר מצואה על ידי הת"ז שלו (כמו שתכתבו בתlös 1). אין עמודה בשם "מנהל", כי אנחנו מזמינים חוקר (מנהל) לפי הת"ז שלו.

במכון המחבר "צפת למחקרים הגלילי" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושל משפחה. במכון מתנהלים פרויקטים של מחקר. לכל פרויקט מחקר יש קוד מצחה, תיאור, מנהל שהוא אחד החוקרם, ותקציב שנתי שՏסכומו משתנה מיד' שנה. חוץ מהמנהל, לכל פרויקט משיכים חוקרם. חוקר יכול להשתียר ליותר מפרויקט אחד. החוקרם מפרסמים ביחיד מאמרם. בנוסף לרישימת חוקרם שהם מחברי המאמר, לכל מאמר יש קוד מאמר מצחה, כותרת, וכתב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר מאמר אחד.

1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל

טעות נפוצה:
 פרויקט ← שם פרטי מנהל, שם משפחה
מנהלה

הסבר הטעות: אנחנו מזהים חוקר (מנהל)
לפי הת"ז שלו. נניח שיש שני חוקרים,
שקרואים להם דוד כהן, עם שני ת"ז שונים,
בצורה הזאת לא יוכל לדעת מי מהם הוא
המנהל של הפרויקט.

במבחן המבחן "צפת למחקרים בגליל"
מוסעים קרים. לכל חוקר יש מספר
ת"ז, שם פרטי, ושם משפחה. במכון
מתנהלים פרויקטים של מחקר. לכל
פרויקט מחקר יש קוד מזהה, תיאור,
ותקציב שנתי שסכומו משתנה מדי
שנה. חוץ מהמנהל, לכל פרויקט
משמעותיים חוקרים. חוקר יכול להשתיר
ליוטר מפרויקט אחד. החוקרים
郿וסדים ביחיד מאמרם. בנוסף
לרישימת חוקרים שהם מחברים המאמר,
לכל מאמר יש קוד מאמר מזהה, כותרת,
וכتب עת שבו הוא התפרסם. חוקר יכול
לפרסם יותר מאמר אחד.

1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב



טעות נפוצה:
 פרויקט ← תקציב שנתי

הסבר הטעות: התקציב משתנה משנה
לשנה. לכן צריך טבלה שבה שלוש עמודות:
קוד פרויקט, שנה, תקציב.

במבחן המבחן "צפת למחקרים בגליל"
מוסעים קרים. לכל חוקר יש מספר
ת"ז, שם פרטי, ושם משפחה. במכון
מתנהלים פרויקטים של מחקר. לכל
פרויקט מחקר יש קוד מזהה, תיאור,
מנהל שהוא אחד החוקרים,
ותקציב שנתי שסכומו משתנה מדי
שנה. חוץ מהמנהל, לכל פרויקט
משמעותיים חוקרים. חוקר יכול להשתיר
ליוטר מפרויקט אחד. החוקרים
郿וסדים ביחיד מאמרם. בנוסף
לרישימת חוקרים שהם מחברים המאמר,
לכל מאמר יש קוד מאמר מזהה, כותרת,
וכتب עת שבו הוא התפרסם. חוקר יכול
לפרסם יותר מאמר אחד.

1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב

טעות נפוצה:
 פרויקט ← שנה, תקציב

הסבר הטעות: השנה לא תליה בפרויקט.

יכול להיות כמה שנים
לפרויקט

במבחן המבחן "צפת למחקרים בגליל"
מוסעים קרים. לכל חוקר יש מספר
ת"ז, שם פרטי, ושם משפחה. במכון
מתנהלים פרויקטים של מחקר. לכל
פרויקט מחקר יש קוד מזהה, תיאור,
מנהל שהוא אחד החוקרים,
ותקציב שנתי שסכומו משתנה מדי
שנה. חוץ מהמנהל, לכל פרויקט
משמעותיים חוקרים. חוקר יכול להשתיר
ליוטר מפרויקט אחד. החוקרים
郿וסדים ביחיד מאמרם. בנוסף
לרישימת חוקרים שהם מחברים המאמר,
לכל מאמר יש קוד מאמר מזהה, כותרת,
וכتب עת שבו הוא התפרסם. חוקר יכול
לפרסם יותר מאמר אחד.

1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב
4. קוד פרויקט → ת"ז חוקר

הסביר:

זו תלות רב ערכית: לכל פרויקט יש יותר מוחוק אחד וכל חוקר יכול להשתיר ליותר פרויקט אחד.

במבחן המבחן "צפת למחקרים הגליל" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושם משפחה. במכון מתנהלים פרויקטים של מחקר. לכל פרויקט מחקר יש קוד מזהה, תיאור, מנהל שהוא אחד החחוקרים, ותקציב שנתי שסכומו משתנה מיידי. חוץ מהמנהל, לכל פרויקט משיכים חוקרם. חוקר יכול להשתיר ליותר פרויקט אחד. החחוקרים מפרסמים ביחד המאמר. בנוסף לרשותם חוקרם שהם מחברי המאמר, לכל מאמר יש קוד מאמר זהה, כתורת, וכותב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר ממאמר אחד.

נירמול: תרגיל מסכם

1. ת"ז חוקר ← שם פרטי, שם משפחה
 2. קוד פרויקט ← תיאור, ת"ז מנהל
 3. קוד פרויקט, שנה ← תקציב
 4. קוד פרויקט → ת"ז חוקר
 5. קוד מאמר ← כתורת, כתוב עת
- במבחן המבחן "צפת למחקרים הגליל" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושם משפחה. במכון מתנהלים פרויקטים של מחקר. לכל פרויקט מחקר יש קוד מזהה, תיאור, מנהל שהוא אחד החחוקרים, ותקציב שנתי שסכומו משתנה מיידי. חוץ מהמנהל, לכל פרויקט משיכים חוקרם. חוקר יכול להשתיר ליותר פרויקט אחד. החחוקרים מפרסמים ביחד המאמר. בנוסף לרשותם חוקרם שהם מחברי המאמר, לכל מאמר יש קוד מאמר זהה, כתורת, וכותב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר ממאמר אחד.

1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב
4. קוד פרויקט → ת"ז חוקר
5. קוד מאמר ← כתורת, כתוב עת
6. קוד מאמר → ת"ז חוקר

טעות נפוצה:
קוד מאמר, ת"ז חוקר ← כתורת, כתוב עת



הסביר הטעות: הכותרת וכותב העת הם של המאמר בלבד. אם אני יודעת את קוד המאמר, אני יודעת את כתוב העת והכותרת. אם אני יודעת את קוד המאמר, אני לא צריך לדעת גם את ת"ז החוקר כדי לדעת את שם המאמר ואת כתוב העת שלו.

במבחן המבחן "צפת למחקרים הגליל" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושם משפחה. במכון מתנהלים פרויקטים של מחקר. לכל פרויקט מחקר יש קוד מזהה, תיאור, מנהל שהוא אחד החחוקרים, ותקציב שנתי שסכומו משתנה מיידי. חוץ מהמנהל, לכל פרויקט משיכים חוקרם. חוקר יכול להשתיר ליותר פרויקט אחד. החחוקרים מפרסמים ביחד המאמר. בנוסף לרשותם חוקרם שהם מחברי המאמר, לכל מאמר יש קוד מאמר זהה, כתורת, וכותב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר ממאמר אחד.

1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב
4. קוד פרויקט → ת"ז חוקר
5. קוד מאמר ← כותרת, כתוב עת
6. קוד מאמר → ת"ז חוקר

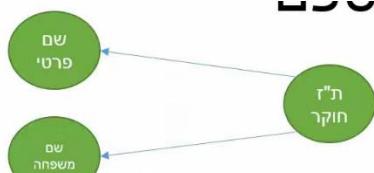
טעות נפוצה:

קוד מאמר, קוד פרויקט ← ת"ז חוקר

הסבר הטעות: (א) שם מקום בשאלת לא היה מצוין קשור בין מאמר לפרויקט; (ב) גם אם היה, שתי תליות רב ערכיות צרוכות להיות מופרדות זו מזו, למשל: אסור לשים את תלוות 4 ואת תלוות 6 באותה תלוות (יהו שורות כפולות בטבלה מאוחדת).

במבחן המבחן "צפת למחקרים הגליל" מועסקים חוקרם. לכל חוקר יש מספר ת"ז, שם פרטי, ושם משפחה. בכך מתנהלים פרויקטים של מחקר. לכל פרויקט מוחקן יש קוד מזיהה, תיאור, מנהל שהוא אחד החחוקרים, ותקציב שנתי שוכמו משתנה מיידי שנה. חוץ מהמנהל, לכל פרויקט מסוימים חוקרם. חוקר יכול להשתיר ליותר פרויקט אחד. החחוקרים מפורטים ביחס מאמרם. בנוסף לרישימת חחוקרים שהם מחברי המאמר לכל מאמר יש קוד מאמר מזיהה, כותרת, וכתוב עת שבו הוא התפרסם. חוקר יכול לפרסם יותר מאשר אחד.

השאלה:



1. ת"ז חוקר ← שם פרטי, שם משפחה

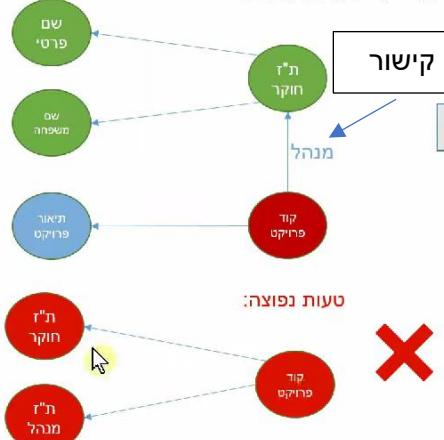
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב
4. קוד פרויקט → ת"ז חוקר
5. קוד מאמר ← כותרת, כתוב עת
6. קוד מאמר → ת"ז חוקר

טעות נפוצה:



הסבר הטעות: אין עמודה בשם "חוקר". כשבונים את התרשים משתמשים רק בשמות העמודות שופיעו ברשימה.

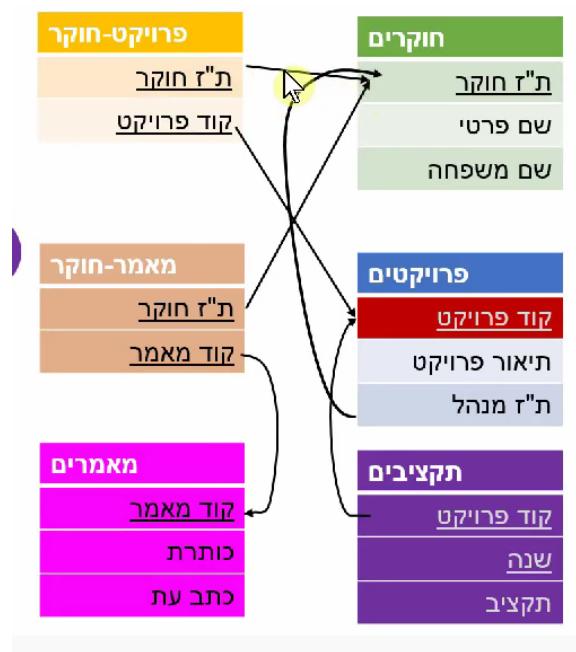
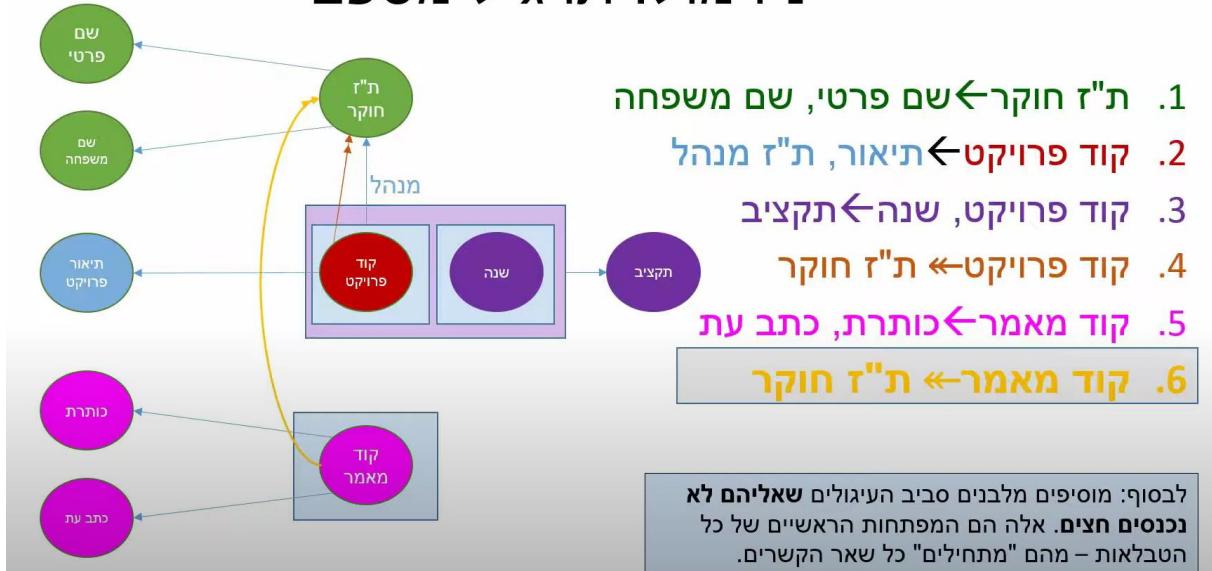
נירמול: תרגיל מסכם



1. ת"ז חוקר ← שם פרטי, שם משפחה
2. קוד פרויקט ← תיאור, ת"ז מנהל
3. קוד פרויקט, שנה ← תקציב
4. קוד פרויקט → ת"ז חוקר
5. קוד מאמר ← כותרת, כתוב עת
6. קוד מאמר → ת"ז חוקר

המנהל הוא אחד החחוקרים. לכל עמודה ברשימה של התליות הפונקציונליות יש בדיק עיגול אחד בתרשים!

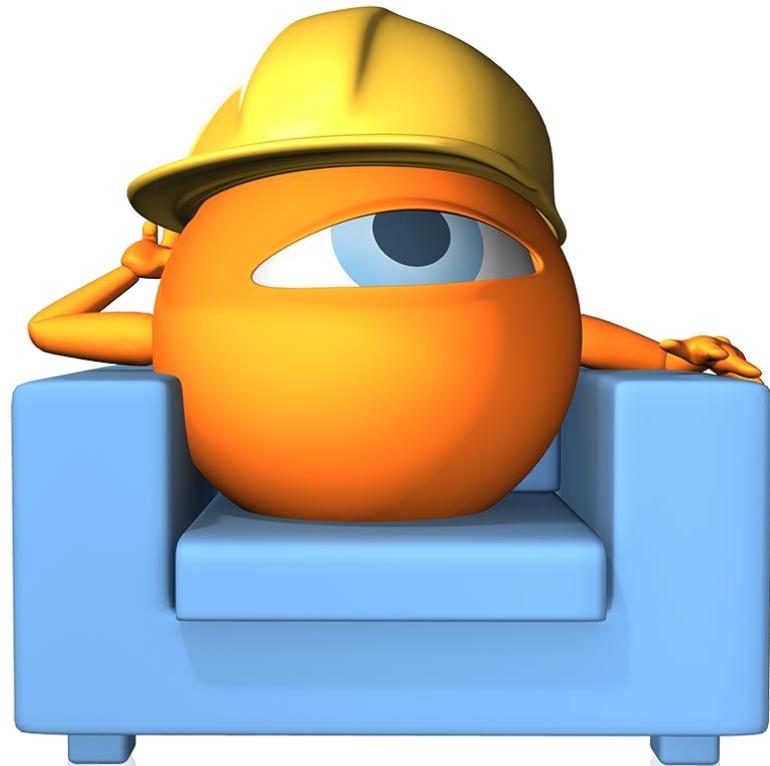
פְּרִוִּיקֶטֶת וְעַדְעָנָה נֵזֶקֶב



בצלחה!!!!

= ת + אלף 14

wowwww



designed by oren obstbaum

All **Jobs & Relax**