שאילתות SQL

ב SQL קיימת שאילתה בסיסית אחת לאיחזור מידע מבסיס הנתונים: פקודת **select**. קיימות אופציות רבות לפקודת ה **select** ב SQL , שיתוארו בהדרגה. נשתמש בתבניות בסיס הנתונים של מערכת הבנק:

- Branch-scheme=(branch-name, assets, branch-city)
- Customer-scheme=(customer-name, street,customer-city)
- Deposit-scheme=(branch-name, account-number, customer-name, balance)
- Borrow-scheme=(branch-name, loan-number, customer-name, amount)

נציין כי יש הבדל חשוב בין שאילתת SQL לבין שאילתת אלגברת יחסים: ב SQL טבלה עשויה להכיל שתי nיות זהות (או יותר). כלומר טבלת SQL אינה, בדרך כלל, **קבוצה** של nיות, כיוון שבקבוצה אין משמעות לשני אברים זהים. טבלות SQL מהוות אוסף, **multiset** (או שק) של nיות. טבלאות SQL תהיינה קבוצות אם תקיימו עליהן אילוצים של מפתח או אם נעשה שימוש באופצית **distinct** .

1. שאילתות SQL בסיסיות

שאילתת SQL הבסיסית מורכבת משלושת פסוקים: **select**, **from**, **where**. המבנה של השאילתה הבסיסית הוא:

select <attribute list>
from <table list>
where <condition>

כאשר:

- <attribute list> היא רשימה של תכונות שאת ערכיהן מאחזרים.
- <table list> היא רשימה של שמות יחסים שעליהם פועלת השאילתה.
- <condition> הוא ביטוי בוליאני המזהה את הnיות המאוחזרות.

נציג כעת את השאילתה הבסיסית בעזרת כמה דוגמאות.

שאילתא 0

מצא את הרחוב ושם העיר של לקוח ששמו 'Morag'.

select street, customer-city
from customer
where customer-name='Morag'

שאילתה זו ניגשה רק ליחס customer המופיע בפסוק **from**. השאילתה מאחזרת את הnיות ביחס customer המקיימות את התנאי בפסוק ה **where** , ואז מטילה את התוצאה על התכונות street ו customer-city. שאילתה זו שקולה לביטוי הבא באלגברת יחסים:

$\Pi_{\text{street, customer-city}}\ (\sigma_{\text{customer-name='Morag'}}\ (\text{customer}))$

כלומר כאשר מופיע יחס אחד בפסוק ה **from** של שאילתת SQL, השאילתה שקולה (כמעט) לצירוף של פעולות בחירה והטלה של אלגברת יחסים. פסוק ה **select** מגדיר את התכונות שעליהן מבצעים הטלה, ופסוק ה **where** מגדיר את תנאי הבחירה.

ההבדל היחיד הוא שהתוצאה של שאילתת SQL עשויה לכלול nיות כפולות, כיוון שהשאילתה אינה כופה את האילוץ שיחס הוא למעשה קבוצה.

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

שאילתא 1

מצא את שמות הלקוחות שיש להם חשבון חיסכון בסניף 'Aviv' בסכום מעל 300 ש"ח

select customer-name
from deposit
where branch-name='Aviv' **and** balance>300

ניתן להרכיב פרדיקטים מורכבים מפרדיקטים פשוטים יותר ע"י המקשרים **and, or, not** בדומה לאלגברת יחסים.

2. מכפלה קרטזית

שאילתא 2

מצא את שמות וכתובות הלקוחות שיש להם חשבון חיסכון בסניף 'Aviv'.

select customer.customer-name,street, customer-city
from customer, deposit
where customer.customer-name = deposit.customer-name
and branch-name='Aviv'

בפסוק ה **where** התנאי 'branch-name='Aviv הוא תנאי-בחירה, והתנאי customer.customer-name = deposit.customer-name הוא תנאי הצירוף בין היחסים deposit ו customer. כפי שניתן לראות כאשר מופיעות שתי תכונות (או יותר) בעלות שם זהה, חייבים לשייך כל אחת מהתכונות ליחס שאליו היא שייכת. הדבר נעשה כמו באלגברת היחסים, ע"י הקדמת שם היחס לשם התכונה, והפרדתם בעזרת נקודה (למשל deposit.customer-name).

שאילתה זו שקולה לשאילתת אלגברת יחסים הכוללת מכפלה קרטזית, בחירה והטלה:

$\Pi_{\text{customer.customer-name street, customer-city}}(\sigma_{\text{customer.customer-name =deposit. customer-name}}$
(customer x deposit))

שאילתא 3

מצא את שמות וכתובות הלקוחות שיש להם חשבון חיסכון בסניף כלשהו הנמצא בעיר 'Jaffa'

select customer.customer-name,street, customer-city
from customer, deposit, branch
where customer.customer-name = deposit.customer-name
and deposit.branch-name= branch.branch-name **and** branch-city='Jaffa'

3. כינוי

שאילתא 4

מצא את שמות וכתובות הלקוחות שגרים בעיר שבה גר 'Morag'

select C.customer-name,C.street, C.customer-city
from customer M, customer C
where C.customer-city = M.customer-city **and** M. customer-name='Morag'

כאן הכרזנו על כינויים M ו C ליחס customer. הכינוי יבוא מיד אחרי שם היחס בפסוק ה **from**. ניתן לחשוב על הכינויים C ו M כעל שני עותקים שונים של היחס customer; ניתן להשתמש בכינוי בכל שאילתת SQL, גם אם אין בה גישה ליחס מסוים יותר מפעם אחת. למשל ניתן לכתוב את שאילתה 2 בצורה הבאה:

select C.customer-name,street, customer-city

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

from customer C, deposit D
where C.customer-name = D.customer-name
and branch-name='Aviv'

4. העדר פסוק where ושימוש ב *

כאשר שאילתת SQL אינה כוללת פסוק **where**, אין בשאילתה תנאי בחירה על nיות. כלומר כל הnיות של היחס בפסוק ה **from** ייבחרו בתוצאת השאילתה. זה שקול ל TRUE **where**. כאשר שאילתת SQL אינה כוללת פסוק **where** ופסוק ה **from** כולל יותר מיחס אחד, אזי כל הnיות של המכפלה הקרטזית של היחסים בפסוק ה **from** יבחרו.

שאילתא 5
מצא את שמות כל סניפי הבנק

select branch-name
from branch

שאילתא 6
מצא את כל הקומבינציות של שמות סניפים עם מספרי חשבונות חיסכון

select branch-name, account-number
from branch, deposit

כדי לאחזר את ערכי כל התכונות של הnיות שנבחרו, אין צורך לציין את שם כל התכונות בפסוק ה **select**. ציון של * (כוכבית) בפסוק ה **select** שקול ל"כל התכונות". שאילתה 7 תציג את כל ערכי התכונות של סניפים הנמצאים בעיר 'Jaffa' ; שאילתה 8 תציג את כל התכונות של deposit ואת כל התכונות של customer לכל חשבונות החיסכון בסניף 'Aviv' והלקוחות המהווים בעלי החשבונות הללו.

שאילתה 7

select *
from branch
where branch-city='Jaffa'

שאילתה 8

select *
from customer, deposit
where customer.customer-name = deposit.customer-name
and branch-name='Aviv'

5. טבלאות כקבוצות ב SQL

כפי שצוין, בדרך כלל SQL אינה מתייחסת ליחס (בסיסי או תוצאה של שאילתה) כאל קבוצה.
בתוצאה של שאילתה, עשויות להופיע nיות זהות אחדות. SQL אינה מסלקת כפילויות כאלה באופן אוטומטי מכמה סיבות:
סילוק כפילויות היא פעולה יקרה. אחת הדרכים היעילות לממש אותה היא ראשית למיין את הnיות ואז לסלק כפילויות.
המשתמש עשוי להיות מעוניין לראות nיות כפילות בתוצאה של שאילתה.
כאשר פונקצית הקבצה מופעלת על nיות, ברוב המקרים אין מעוניינים לסלק nיות כפילות.

אם בכל זאת מעוניינים לסלק nיות כפילות משתמשים במילת המפתח **distinct** בפסוק ה **select**, שפירושה היא שרק nיות שונות זו מזו תישארנה בתוצאה. זה הופך את תוצאת שאילתת SQL ליחס, קבוצה של nיות בהתאם להגדרה הפורמלית של יחס.
למשל, שאילתה 9 תאחזר את סכומי היתרות של חשבונות החיסכון בסניף 'Aviv'. אם סכום מסוים מופיע ביחס deposit פעמים אחדות (למשל אם חשבון מסוים שייך ללקוחות אחדים), הוא יופיע פעמים אחדות גם בתוצאת השאילתה:

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

שאילתה 9

מצא את יתרות חשבונות החיסכון בסניף 'Aviv'.

select balance
from deposit
where branch-name='Aviv'

אם מעוניינים רק ברשימת ערכים ללא כפילויות, כלומר שכל סכום יופיע פעם אחת בלבד, משתמשים ב **distinct**:

select distinct balance
from deposit
where branch-name='Aviv'

SQL ממשה באופן ישיר את פעולות-הקבוצה (set operations) של אלגברת יחסים: פעולת האיחוד (**union**), וכן בניבים שונים גם פעולות של החיסור (**minus** או **except**) וחיתוך (**intersect**). התוצאה של שאילתה המשתמש בפעולת-קבוצה היא תמיד קבוצה, כלומר nיות כפילות מסולקות מהתוצאה (אלא אם כן מופיעה מילת המפתח **all** מייד אחרי הפעולה, למשל **union all**). כמו באלגברת יחסים, פעולת-קבוצה מוגדרת רק אם היחסים שעליהם היא פועלת הם בעלי תבניות תואמות.

שאילתה 10

מצא את שמות כל הלקוחות של סניף 'Aviv' (בעלי חשבונות חיסכון ו/או חשבונות הלוואה)

(**select** customer-name
from deposit
where branch-name='Aviv')
union
(**select** customer-name
from borrow
where branch-name='Aviv')

6. חישובים בשאילתות SQL

ניתן לכלול חישובים אריתמטיים בשאילתת SQL בפסוק ה **select** או כחלק מפסוק ה **where**. ניתן להפעיל את האופרטורים האריתמטיים '+' , '-', '*' ו '/' (חיבור, חיסור, כפל, חילוק) על ערכים נומריים בשאילתה.

שאילתה 11

הראה את מספרי החשבון ואת יתרות כל חשבונות החיסכון בסניף 'Aviv' אם כל חשבון חיסכון בסניף זה יזכה בתוספת של 10% מיתרתו.

select account-number, 1.1*balance
from deposit
where branch-name='Aviv'

שאילתה 12

נניח כי תבנית חשבונות החיסכון כוללת גם את התכונה "ריבית שנתית" - interest , המתאר את הריבית השנתית באחוזים. מצא את שם הבעלים, מספרי חשבונות החיסכון ואת היתרות של חשבונות החיסכון שהריבית השנתית שלהם גדולה מ500 ש"ח.

select customer-name, account-number, balance
from deposit
where balance*interest/100 > 500

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

7. שאילתות מקוננות והשוואת קבוצות

ביצוע שאילתות מסוימות מחייב כי ערכים מבסיס-הנתונים יאוחזרו ראשית ואז יעשה בהם שימוש בתנאי של השוואה. שאילתות מסוג זה ניתן לנסח על ידי שימוש בשאילתות מקוננות, שמהוות שאילתות **select** שלמות בתוך פסוק **where** של שאילתה אחרת. השאילתה ה"אחרת" נקראת "שאילתה חיצונית" (outer query).שאילתה 9 למשל, בוצעה ללא שאילתה מקוננת, אך ניתן לבצעה גם במבנה מקונן:

```
select customer-name
from customer
where customer-name in      (select customer-name
                             from deposit
                             where branch-name='Aviv')
      or
      customer-name in      (select customer-name
                             from borrow
                             where branch-name='Aviv')
```

ראוי לשים לב כי בשאילתה הראשית אין צורך לציין **distinct** היות והשאילתה הראשית שולפת את הערך של customer-name מהיחס customer שבו היא מהווה מפתח קביל, כך שחיות כפילות לא תיתכנה בתוצאה. אילו בפסוק ה **from** של השאילתה הראשית היינו מציינים את היחס depositבמקום זאת, היינו צריכים לציין **distinct** בפסוק ה **select**.
האופרטור **in** יכול גם להשוות חית ערכים בסוגריים עם קבוצה של חיות בעלות אותו מבנה. למשל השאילתה

שאילתה 13

```
select customer-name
from customer
where (street, customer-city) in      (select street, customer-city
                                       from customer
                                       where customer-name='Morag')
```

תבחר את שמותיהם של כל הלקוחות שגרים באותו רחוב ועיר שבה גר הלקוח 'Morag'.

בנוסף לאופרטור **in** יש מספר אופרטורי השוואה נוספים שבאמצעותם ניתן להשוות שערך יחיד v (בדרך כלל ערך של תכונה מסוימת) לקבוצת ערכים V (בדרך כלל שאילתה מקוננת).
האופרטור **some=** מחזיר ערך true אם הערך v שווה לאחד הערכים בקבוצה V, ולכן שקול לאופרטור **in**. אופרטורים נוספים שניתן לשלב עם **some** כוללים: **<, <=, >, >=, =** .
ניתן לצרף את מילת המפתח all לכל אחד מהאופרטורים הנ"ל (**<, <=, >, >=, =**). למשל, התנאי (v **> all** V) יחזיר true אם הערך v גדול מכל הערכים בקבוצה V. כדוגמה:

שאילתה 14

מצא את שמות החוסכים שיש להם חשבון חסכון בעל יתרה הגדולה מכל היתרות בסניף 'Aviv'

```
select distinct customer-name
from deposit
where balance > all    (select balance
                        from deposit
                        where branch-name='Aviv')
```

במקרה הכללי עשויות להיות מספר רמות של שאילתות מקוננות. שוב קיימת אפשרות של דו-משמעות של שמות התכונות אם קיימות תכונות בעלות שמות זהים בשאילתה החיצונית

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

ובשאילתה הפנימית. הכלל הוא שאם מצוינת תכונה ללא שם יחס, היא מתייחסת ליחס המוגדר בשאילתה המקוננת הפנימית. כלל זה דומה מאד לצורה שבה מוגדרים משתנים לוקליים בשפות עיליות רבות כמו פסקל. לדוגמה:

שאילתה 15

מצא את הסכום ואת מספר החשבון של כל חשבון הלוואה שיש חשבון חיסכון בסכום זהה לסכום של חשבון ההלוואה באותו סניף.

```
select distinct loan-number, amount
from borrow B
where amount in      (select balance
                      from deposit
                      where branch-name=B.branch-name)
```

בשאילתה המקוננת חייבים לציין B.branch-name כיוון שתכונה זו מתייחסת ל תכונה branch-name של borrow בשאילתה החיצונית. התכונה branch-name ללא ציון שם יחס מתייחסת ליחס deposit בשאילתה הפנימית.

באופן כללי, שאילתה הנכתבת במבנה מקונן של בלוקים של **select**… **from** … **where** תוך שימוש ב **in** או אופרטור השוויון יכולה תמיד להיכתב בשאילתה של בלוק אחד.
למשל שאילתה מס' 15 יכולה להיכתב גם בצורה:

```
select distinct loan-number, amount
from borrow B, deposit D
where B.branch-name=D.branch-name and amount = balance
```

SQL כוללת גם אופרטור השוואה **contains** בין קבוצות. אופרטור זה אינו ממומש בחלק מן הניבים המסחריים של SQL. אופרטור זה משווה שתי קבוצות ומחזיר ערך true אם אחת הקבוצות מכילה את הקבוצה האחרת.

שאילתה 16

מצא את שמות הסניפים שלכל האנשים הגרים ב 'jaffa' יש בהם חשבונות חיסכון.

```
select branch-name
from branch B
where        (select customer-name
              from deposit
              where branch-name=B.branch-name)
              contains
              (select customer-name
              from customer
              where customer-city='jaffa')
```

בשאילתה זו, השאילתה המקוננת השנייה (שהיא קבועה ואינה מקושרת לשאילתה החיצונית) מוצאת את שמות כל הלקוחות ב 'jaffa'. לכל היה של סניף (B) השאילתה המקוננת הראשונה מוצאת את שמות הלקוחות שביש להם חשבון חיסכון באותו סניף.
שים לב כי פעולה זו שקולה לפעולת החילוק באלגברת היחסים.

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

8. פונקצית exists

פונקצית **exists** משמשת ב SQL לבדוק אם התוצאה של שאילתה מקוננת אינה ריקה. הפונקציה מחזירה ערך true אם תת-השאילתה מכילה n יות . לדוגמא:

שאילתה 17

מצא את שמות האנשים שיש להם חשבון חיסכון וגם חשבון הלוואה בסניף 'Aviv'

```
select distinct customer-name
from deposit D
where  branch-name='Aviv' and
        exists  (select *
        from borrow B
        where branch-name='Aviv' and D.customer-name=B. customer-name )
```

שאילתה זו ניתן כמובן לבצע גם בדרכים אחרות, למשל בעזרת intersect במבנה של שאילתה 9.
ניתן באופן דומה להשתמש ב **not exists** המחזירה ערך true אם תת-השאילתה ריקה.

שאילתה 18

מצא את שמות האנשים שאין להם אף חשבון חיסכון בסכום מעל 400 ש"ח.

```
select customer-name
from customer C
where not  exists      (select *
                       from deposit D
                       where D.customer-name=C. customer-name
                             and  balance>400)
```

ניתן להבין שאילתה זו כדלקמן: לכל n ית לקוח (C) מחשבים את השאילתה המקוננת, שמאחזרת את קבוצת כל n יות חשבונות ההפקדה של הלקוח בסכום מעל 400 ש"ח. אם קבוצה זו ריקה, ללקוח אין אף חשבון בסכום מעל 400 ש"ח.

ניתן לבצע באמצעות **exists** ו **not exists** גם בדיקת הכלה שבוצעה בעזרת **contains**. למשל, שאילתה מס' 16 יכולה להתבצע גם בדרך הבאה:

שאילתה א16

מצא את שמות הסניפים שלכל האנשים הגרים ב 'jaffa' יש בהם חשבונות חיסכון.

```
select branch-name
from branch B
where  not exists
                (select *
                from customer
                where customer-city='jaffa' and
                        customer-name not in (select customer-name
                                              from deposit
                                              where branch-name=B.branch-name)
```

המבנה של שאילתה זו תואם את הניסוח מחדש של שאילתה 16: מצא שמות סניפים, כך שלא קיים אדם הגר ב 'jaffa' שאין לו חשבון חיסכון בסניף.
זוהי צורה נוספת לבצע את פעולת החילוק של אלגברת היחסים.

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

9. קבוצות מפורשות וערכי null

ניתן להשתמש בפסוק ה **where** גם בקבוצות מפורשות של ערכים. קבוצה כזו מוקפת בסוגריים.

שאילתה 19

מצא את שמות האנשים שיש להם חשבון חיסכון בסניף 'Darom', 'Merkaz' או 'Tsafon'.

select distinct customer-name
from deposit
where branch-name **in** (Darom', 'Merkaz', 'Tsafon')

SQL מאפשרת גם לבדוק אם ערך מסוים הוא ערך דמה (null), כלומר חסר או לא מוגדר. אבל לא ניתן להשתמש ב = או <> לצורתך השוואה של תכונה ל null. השוואה כזו תיתן תמיד false, מפני ש SQL מתייחסת לכל ערך null כאל ערך ייחודי, שונה מערכי null אחרים!
לכן משתמשים ב:SQL במקום זאת ב **is null** או ב **is not null**.

שאילתה 20

מצא את מספרי חשבונות החיסכון שאינם משוייכים לסניף כלשהו. (מצב בלתי רצוי כלל שבמערכת ריאלית לא היה קורה)

select distinct account-number
from deposit
where branch-name **is null**

10. פונקציות הקבצה והקבצה

לעתים שכיחות יש צורך להפעיל פונקציות על קבוצות של חיות. פונקציות אלה נקראות פונקציות הקבצה (aggregate functions). הארגומנט של פונקצית הקבצה הוא אוסף של ערכים (לאו דווקא קבוצה) . פונקצית הקבצה מחזירה תמיד ערך יחיד.
ב SQL יש חמש פונקציות הקבצה מובנות: **count, sum, max, min, avg**. הפונקציה **count** מחזירה את מספר החיות או הערכים הנבחרים בשאילתה. הפונקציות **max, min, sum** ו **avg** מחזירות בהתאמה את סכום הערכים, את הערך המינימלי, המקסימלי, ואת הערך הממוצע של ערכים אלה. ניתן להשתמש בפונקציות אלה בפסוק ה **select** או בפסוק **having** (שיתואר בהמשך).
דוגמאות:

שאילתה 21

מצא את הסכום, ואת הערך המירבי והמינימלי של ערכי הנכסים של הבנקים.

select sum(assets), **max**(assets), **min**(assets)
from branch

אם ברצוננו לבצע שאילתה כזו על חלק מהחיות אזי יש להשתמש בפסוק **where** מתאים.

שאילתה 22

מצא את הערך המירבי והמינימלי של חשבונות החיסכון של לקוחות שגרים בעיר 'Jaffa'.

select sum(balance), **max**(balance s), **min**(balance)
from deposit, customer
where customer-city='Jaffa' **and** customer.customer-name=deposit.customer-name

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

שאילתה 23
מצא את מספר סניפי הבנקים

select count(*)
from branch

שאילתה 24
מצא את מספר חשבונות החיסכון של לקוחות שגרים בעיר 'Jaffa' .

select count(*)
from deposit, customer
where customer-city='Jaffa' **and** customer.customer-name=deposit.customer-name

בשאילתות 23 ו 24 הכוכבית (*) מתייחסת לשורות (חיות), כלומר **count**(*) מחזירה את מספר השורות בתוצאת השאילתה. בשאילתה 24, אם יש חשבון מסוים משותף לשני לקוחות הגרים ב 'jaffa' הוא ייספר פעמיים, כיוון שיהיו שתי חיות שיענו על קריטריון הבחירה בפסוק ה **where**.
ניתן להשתמש ב **count** לספור ערכים בעמודה מסוימת. למשל בדוגמה הבאה:

שאילתה 25
מצא את מספר הערים השונות שבהם יש לקוחות של הבנק

select count(**distinct** customer-city)
from customer

יש לשים לב כי בשאילתה זו, **count**(customer-city) היה נותן ערך כמו של **count**(*), כיוון שסופרים גם ערכים חוזרים.

שאילתה 26
מצא את שמות הלקוחות שיש להם חשבונות חיסכון בסניפי בנק בשתי ערים שונות לפחות.

select distinct D.customer-name
from deposit D
where (**select count**(**distinct** branch-city)
  from deposit P, branch B
  where B.customer-name=D.customer-name **and**
    P.branch-name= B.branch-name) >=2

השאילתה הפנימית סופרת את מספר הסניפים השונים שלכל לקוח יש; אם ערך זה גדול או שווה לשתים, החיה של חשבון הלקוח (D) תיבחר.

במקרים רבים מעוניינים להפעיל פונקצית ההקבצה על תת-קבוצות של חיות ביחס, בהתאם לערכים של תכונה מסוימת. למשל, אם אנו מעוניינים למצוא את היתרה הממוצעת בכל סניף, או מספר חשבונות החיסכון של כל לקוח. במקרה זה עלינו לקבץ את החיות שיש להם ערך משותף בתכונה המתאימה, ולהפעיל את פונקצית ההקבצה על כל קבוצה כזו בנפרד. את זה עושה פסוק **group by** של SQL. פסוק **group by** מציין את התכונה (תכונות) שערכיה מגדירים את הקבוצות. רק תכונות המופיעות בפסוק ה **group by** עשויות להופיע בפסוק ה **select**.

שאילתה 27
לכל לקוח שיש לו חשבון חיסכון אחד או יותר, הדפס את שם הלקוח, את מספר חשבונות החיסכון שלו ואת הסכום הכולל של היתרות.

select customer-name, **count**(*), **sum**(balance)
from deposit
group by customer-name

בשאילתה זו חיות היחס deposit חולקו לקבוצות. בכל קבוצה חיות עם אותו ערך של "התכונה המקבצת" customer-name. כלומר בכל קבוצה יופיעו רק חשבונות חיסכון של לקוח אחד.
הפונקציות **count** ו **sum** מופעלות על כל אחת מהקבוצות הללו.

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

התוצאה של שאילתה כזו עשויה להיראות כדלקמן:

customer-name	count(*)	Sum(balance)
Even	1	300
Morag	2	550
Tal	4	1500
Levi	3	1200

שאילתה 28

עבור לקוחות שיש להם חשבונות חיסכון: מצא כמה חשבונות חיסכון יש ללקוחות הגרים בכל רחוב בכל עיר.

select customer.customer-city, street, **count**(**distinct** account-number)
from customer C, deposit D
where C. customer-name=D. customer-name
group by street, customer-city

בשאילתה זו מופיעה תנאי בחירה (בפסוק ה **where**) וכן הקבצה (**group by**). במקרה זה, ההקבצה וכן פונקציות ההקבצה מופעלות לאחר בחירת הnיות על פי תנאי הבחירה בפסוק ה **where**. דוגמא לתוצאה אפשרית (בהנחה שלקוחות הבנק גרים בארבעה רחובות בלבד):

customer-city	Street	**count**(**distinct** account-number)
Jaffa	Pinkas	3
Jaffa	Allenby	5
Rishon	Etzel	4
Rishon	Lehi	6

לעתים מעוניינים באיחזור של ערכי פונקציות הקבצה רק לגבי קבוצות המקיימות תנאי כלשהו.. לצורך זה מספקת SQL את הפסוק **having**, שעשוי להופיע יחד עם פסוק ה **group by**. **having** מהווה תנאי על קבוצות של nיות, כפי שקובצו על ידי פסוק ה **group by**, רק קבוצות המקיימות את תנאי ה **having** תופענה בתוצאת השאילתה.

שאילתה 29

לכל לקוח שיש לו חשבון לפחות שלושה חשבונות חיסכון, הדפס את שם הלקוח, את מספר חשבונות החיסכון שלו ואת הסכום הכולל של היתרות.

select customer-name, count(*), sum(balance)
from deposit
group by customer-name
having count(*)>=3

חשוב לשים לב כי תנאי הבחירה בפסוק ה **where** מגביל nיות שעליהן יפעלו הפונקציות, פסוק ה **having** מגביל קבוצות שלמות של nיות.

במקרה שיש שני תנאים שונים (האחד על הפונקציה בפסוק ה **select** והשני על הפונקציה בפסוק ה **having**). למשל, נניח כי אנו מעוניינים לספור בכל סניף את מספר הלקוחות שיש להם חשבון חיסכון עם יתרה גדולה מ 4000 ש״ח, אבל רק לסניפים שיש בהם לפחות 5 חשבונות חיסכון.

select branch-name, **count**(**distinct** customer-name)
from deposit
where balance>4000
group by branch-name
having count(**distinct** account-number)>=5

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

שאילתה זו שגויה, מכיוון שהיא תציג רק סניפים שיש בהם לפחות 5 חשבונות חיסכון עם יתרה גדולה מ4000. זאת מפני שהכלל הוא: פסוק ה **where** מבוצע ראשון לבחירת ה nיות הבודדות. אז מתבצעת ההקבצה ורק אז מתבצע פסוק ה **having** לבחירת קבוצות של nיות. הדרך הנכונה לבצע את השאילתה היא:

שאילתה 30

עבור סניפים שיש בהם לפחות 5 חשבונות חיסכון, הצג את מספר הלקוחות שיש להם חשבון חיסכון עם יתרה גדולה מ 4000 ש״ח.

```
select branch-name, count(distinct customer-name)
from deposit
where balance>4000 and
        branch-name in      (select branch-name
                             from deposit
                             group by branch-name
                             having count(distinct account-number)>=5 )
group by branch-name
```

11. השוואת תת מחרוזות

SQL מאפשרת לבצע השוואה על תתי-מחרוזת של תווים באמצעות אופרטור ההשוואה **like**. מחרוזות חלקיות ניתנות להגדרה באמצעות שני תווים שמורים: '%' מייצג מספר כלשהו של תווים (כולל מחרוזת ריקה), ו '_' מייצג תו שרירותי אחד.

שאילתה 31

מצא את פרטי הלקוחות ששם העיר בה הם גרים מתחילה ב 'R'.

```
select *
from customer
where customer-city like 'R%'
```

שאילתה 32

מצא את פרטי הסניפים שהאות השנייה של שמם היא 't' .

```
select *
from branch
where branch like '_t%'
```

12. מיון התצוגה

SQL מאפשרת למשתמש לסדר את הnיות בתוצאה של שאילתה על פי הערכים של תכונה אחת או יותר, באמצעות פסוק ה **order by**.

שאילתה 33

מצא את פרטי חשבונות חיסכון בסניף 'Aviv'. מיין את החשבונות על פי שמות הלקוחות, ואת כל החשבונות השייכים ללקוח אחד מיין על פי היתרה.

```
select *
from deposit
where branch-name='Aviv'
order by customer-name, balance
```

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

ברירת המחדל היא מיון בסדר עולה. לצורך מיון בסדר יורד יש לציין את מילת המפתח **desc**.
ניתן גם להשתמש במילת המפתח **asc** כדי לציין מפורשות מיון בסדר עולה. למשל אם בשאילתה הקודמת היינו רוצים שהמיון המשני, על פי היתרה יהיה בסדר יורד (כלומר בין כל החשבונות של לקוח מסוים יופיע ראשון החשבון עם יתרה מירבית וכו') אזי פסוק ה **order by** של השאילתה היה

order by customer-name **asc**, balance **desc**

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

פעולות עדכון

1. פקודת insert

בצורתה הפשוטה ביותר **insert** משמשת להוספת n יה ליחס. יש לציים את שם היחס ואת רשימת הערכים של ה n יה. יש לציין את הערכים על פי הסדר שבו מופיעות התכונות בהגדרת התבנית (פקודת create table).

הוספה 1

הוסף n ית לקוח ליחס customer

insert into customer
values (‘Aharoni’, ‘Dizengof’, ‘Tel Aviv’)

צורה שנייה של פקודת **insert** מאפשרת למשתמש לציין במפורש שמות של תכונות המתאימות לערכים בפקודת ה **insert**. במקרה זה, ניתן להשמיט תכונות עם ערכי דמה (null) או ברירת מחדל. למשל:

הוספה 2

insert into deposit(branch-name, customer-name, account-number)
values (‘Aviv’**,‘**Davidi’, 123)

התכונה balance שהושמטה, תקבל ערך null או ערך ברירת מחדל (אם הוגדרה ברירת מחדל). הערכים בפקודה זו מצוינים בסדר שבו מצוינות התכונות בפקודת ה **insert** עצמה.

וריאציה של פקודת **insert** המאפשרת להוסיף n יות אחדות ליחס משתמשת בתוצאה של שאילתה. למשל:

הוספה 3

לכל לקוח שיש לו חשבון הלוואה בסניף ‘Aviv’ הוסף n יה של חשבון הפקדה באותו סניף עם יתרה שהיא עשירית מסכום ההלוואה שלו, ומספר חשבון הפקדה זהה למספר חשבון ההלוואה.

insert into deposit
select branch-name,loan-number,customer-name,amount*0.1
from borrow
where branch-name=’Aviv’

2. פקודת delete

פקודת **delete** מוחקת n יות מיחס. הפקודה כוללת פסוק **where**, בדומה לזה שבשאילתת SQL. n יות נמחקות באופן מפורש מיחס אחד בכל פעם. (יחד עם זאת, בגלל אילוצים של שלמות קשר, המחיקות עלולות לעבור ליחסים אחרים). מספר ה n יות הנמחקות בפקודת **delete**אחת הוא כמספר ה n יות הנבחרות בפסוק ה **where**. בהיעדר פסוק **where** כל n יות היחס יימחקו; יחד עם זאת, היחס עצמו לא יימחק, אלא יישאר בבסיס הנתונים כיחס ריק.

מחיקה 1

מחק את n ית הלקוח ‘Morag’

delete customer
where customer-name=’Morag’

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

מחיקה 2
מחק את כל חשבונות החיסכון של סניפים שערך הנכסים שלהם שלילי

```
delete  deposit
where branch-name in        (select branch-name
                             from branch
                             where assets <0)
```

מחיקה 3
מחק את כל חשבונות ההלוואה.

```
delete  borrow
```

3. פקודות update

פקודת **update** משמשת לעדכון ערכים של תכונות ב nיה אחת או יותר.כמו בפקודת ה **delete** פסוק ה **where** בפקודת ה **update** בוחרת את האניות שיש לעדכן מתוך יחס אחד. (יחד עם זאת, עדכון של מפתח ראשי עלול לגרום לעדכונים של מפתחות זרים ביחסים אחרים, בהתאם לאילוצי שלמות קשר שהוגדרו). פסוק set מגדיר את התכונות שמעדכנים ואת ערכיהן החדשים. למשל:

עדכון 1
שנה את שם הרחוב שבו גר 'Morag' ל 'Dizengof' .

```
update  customer
set  street= 'Dizengof'
where customer-name='Morag'
```

ניתן לשנות nיות אחדות בפקודת **update** אחת. למשל:
עדכון 2
הוסף 5% לכל היתרות של חשבונות חיסכון של לקוחות הגרים ב 'jaffa' .

```
update  deposit
set  balance= balance*1.05
where customer-name in (select customer-name
                        from customer
                        where customer-city='Jaffa')
```

ניתן לעדכן ערכים בטבלה אחת בהסתמך על ערכים בטבלה אחרת. למשל:
עדכון 3
נניח שללקוח מותר שיהיה רק חשבון חיסכון אחד לכל היותר וכן חשבון הלוואה אחד לכל היותר. עדכן את מס' חשבונות ההלוואה של לקוחות שיש להם גם חשבון חיסכון להיות מס' חשבון החיסכון שלהם. (לצורך שאילתא זו נניח כי ללקוח לא יותר מחשבון חיסכון אחד וחשבון הלוואה אחד)

```
update  borrow
set  loan_number =     (select account_number
                       from deposit
                       where deposit.customer-name = borrow.customer-name)
where exists           (select *
                       from deposit
                       where deposit.customer-name = borrow.customer-name)
```

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

תצפיות ב SQL

1. מושג התצפית ב SQL

תצפית ב SQL היא טבלה המופקת מטבלאות אחרות. הטבלאות האחרות עשויות להיות טבלאות בסיסיות או תצפיות אחרות שהוגדרו קודם לכן. תצפית אינה בהכרח ממומשת בצורה פיזית; אלא ניתן לראותה כטבלה וירטואלית, בניגוד לטבלאות (היחסים) הבסיסיים,שהnיות שלהם מאוחסנות פיסית בבסיס הנתונים. עובדה זו מגבילה את אפשרות העדכון של תצפית, אך אינה מטילה מגבלה כלשהי על שאילתות על התצפית.
ניתן לראות תצפית כדרך להגדיר טבלה שניגשים אליה לעתים תכופות, למרות שאינה קיימת פיסית. למשל, אם שאילתה רבות מאחזרות פרטי הלקוח יחד עם פרטי חשבונות החיסכון שלו, ניתן להגדיר תצפית שהיא תוצאה של הצירוף הטבעי של היחסים customer ו deposit, ועליה לבצע כל פעם את השאילתה.

2. הגדרת תצפית

את התצפית מגדירים בעזרת הפקודה create view. התצפית מקבלת שם טבלה (וירטואלי), רשימה של תכונות, וכן שאילתה המגדירה את תוכן התצפית. אם אף אחת מהתכונות בתצפית אינה תוצאה של הפעלת פונקציות או אופרטורים אריתמטיים, אין צורך להגדיר את שמות התכונות בתצפית, ושמות התכונות יהיו זהים לשמות התכונות בטבלאות המקוריות המגדירות את התצפית. למשל:

תצפית 1

```
create view customer-deposits
as      select C.customer-name, street, customer-city, account-number, balance
        from customer C, deposit D
        where C.customer-name= D.customer-name
```

בתצפית זו לא הגדרנו את שמות התכונות, כך שהם יהיו כשמות ביחסים המקוריים.

תצפית 2

```
create view total-branch-deposits  (branch-name, total-balance)
as      select branch-name, sum(balance)
        from deposit
        group by branch-name
```

בתצפית זו הגדרנו את שמות התכונות. זאת כדי לתת שם לעמודה המחושבת על ידי פונקצית ההקבצה **sum**.

משהוגדרה תצפית, ניתן לבצע עליה שאילתות למשל:

שאילתת תצפית 1

הצג את כל מספרי חשבונות החיסכון שיש ללקוחות הגרים ב 'Jaffa'

```
select distinct account-number
from customer-deposits
where customer-city='Jaffa'
```

שאילתת תצפית 2

הצג את סכום ההפקדות בסניף 'Aviv'

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008

```
select *
from total-branch-deposits
where branch-name='Aviv'
```

ניתן לסלק הגדרה של תצפית שאין בה צורך יותר בעזרת הפקודה drop view. למשל:

```
drop view total-branch-deposits
```

3. עדכון התצפיות

עדכון תצפיות הוא מורכב ופעמים רבות אינו חד-משמעי. בנושא זה מתנהל מחקר נמרץ. באופן כללי, עדכון של תצפית, המבוססת על טבלה בסיסית אחת, ושאינה כוללת פונקציה הקבצה, ניתן לתרגום לעדכון של הטבלה הבסיסית. בכל מקרה אחר יש דרכים אחדות למפות את עדכון התצפית לעדכון של הטבלאות הבסיסיות.

למשל, אם מתבצעת פעולת העדכון הבאה על התצפית total-branch-deposits

```
update total-branch-deposits
set  total-balance=42000
where branch-name ='Aviv'
```

לא ברור אילו nיות של היחס deposit יש לעדכן, ובאיזה אופן!!!

לסיכום:

- תצפית הבנויה מטבלה בסיסית אחת ניתנת לעדכון אם תכונות התצפית כוללות את המפתח הראשי של הטבלה הבסיסית או מפתח קביל אחר שלה. כמו כן, כל תכונה ביחס הבסיסי שאינה יכולה לקבל ערך דמה (null) חייבת להיכלל בתצפית.
- תצפיות המוגדרות על טבלאות אחדות באמצעות צירוף אינן ניתנות לעדכון בדרך כלל.
- תצפיות המוגדרות בעזרת הקבצה או פונקציות הקבצה אינן ניתנות לעדכון.

(c) כל הזכויות שמורות לפרופ' גדעון דרור, ביה"ס למדעי המחשב, המכללה האקדמית של ת"א יפו. 2008