13.10.2015

קורס SQL יום 1

RDBMS יתרונות לעומת קבצים, שפת SQL, מושגי יסוד: Schema, Database

שמירת נתונים במחשב (על הדיסק הקשיח בד"כ), נעשית בעיקרון במבנים שניקראים קבצים שנימצאים תחת תיקיות.

דוגמאות לקבצים: מסמך בוורד, אקסל או תוכנה אחרת. לצורך אישי זה בד"כ מספיק. קובץ שיצרנו, אפילו אם זו רשימה של 100 אנשים שלכ"א צריכים לתת מתנה לחג, קובץ אקסל עם השם, טלפון ועמודה שאומרת אם קיבל או קיבלה יספיק, ואפשר לנהלו ידנית. בקבצים כאלה יספיק מבט חטוף לסדר או לעדכן את הקובץ.

מה יקרה אם יתנו לנו לנהל **ספר טלפונים** של עיר בינונית עם 50,000 מנויים ויום אחד נצטרך לשלוף את כל אלה שעם קידומת 052 ו-050, או שנירצה רשימה של כל אלה שגרים ברחוב רוטשילד, כדי לשלוח להם מכתב אזהרה שבעוד חודש מתחילים לחפור אצלם ברחוב עבור הרכבת הקלה?

המבט החטוף שלנו לא יספיק, צריך להפעיל פונקציה של חיפוש על כל הקובץ, ועבור כל אחד שמצאנו לבצע איזה עיבוד (הכנסת שמו למכתב).

מכאן התפתח הצורך ליצור קבצים בעלי מבנה של רשומות. מה זה רשומה? אוסף שדות שיש ביניהם איזה קשר. **מהו שדה אם כך?** שדה היא פיסת האינפורמציה הקטנה ביותר שכדאי לשמור.

למשל האם לשמור שם כשדה אחד, או כשני שדות? זה כבר תלוי באיך נשתמש בשדה.

הקבצים הראשונים שנוצרו ניקראו : סידרתיים, זאת כיוון שניתן היה לעבדם במחשב כאוסף עוקב של רשומות מההתחלה לסוף. במקרים מסוימים, יכולנו גם למיין את הקובץ, למשל על פי שם משפחה וזה היה מקל על מציאת אדם בודד (כמו בספר טלפונים). כל פעם שנוסף מנוי, היינו צריכים לעשות רה-אירגון של הקובץ.

אז החליטו להוסיף משהו שניקרא **אינדקס**. האינדקס הכיל רק את השם, וכל כניסה באינדקס הצביעה פיזית על הרשומה מתאימה, כך בהוספת מנוי היינו צריכים רק לארגן את האינדקס שהיה קטן יותר.

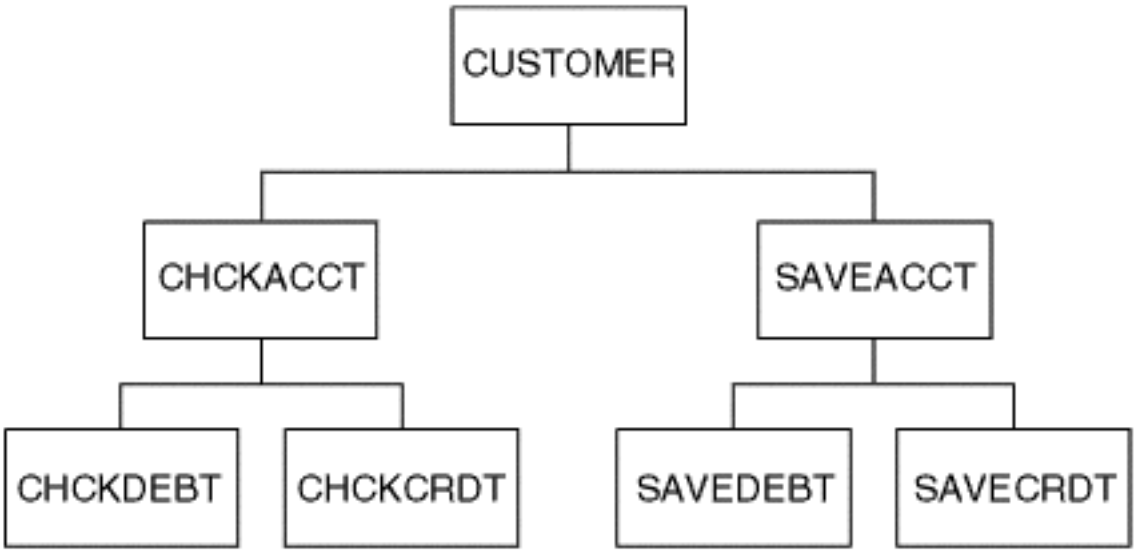
האם זה היה עוזר לשליפת כל אלה בעלי קידומת 052? לשם כך היינו צריכים ליצור **אינדקס נוסף**, על פי מספרי טלפון.

אם הקובץ היה עבור לקוחות של בנק. נניח שהייתה רשומה עבור חשבון עו"ש שבה נישמר המאזן בחשבון. מה אם לאותו לקוח היה גם חשבון חיסכון? האם זו תהיה רשומה נוספת באותו קובץ? האם זה יהיה קובץ ניפרד? מה אם רצינו לשמור כל טרנזקציה שהלקוח ביצע - היינו משכפלים כל רשומה כמספר הטרנזקציות? ביזבוז עצום של מקום. או שהיו לנו כבר 3 קבצים, אחד עבור מאזן הלקוח, שני עבור התנועות שביצע ושלישי עבור חשבון חיסכון.

בכל אחד מהקבצים הללו יהיה שם של לקוח , מספר חשבון, כתובת וכו'. מה אם מחר **הלקוח עובר דירה?** כעת ניצטרך **לעדכן כתובת בשלושה קבצים**. בקיצור, השימוש בקבצים מעין אלה דרש גם זמן וגם משאבי זיכרון עצומים, שלא לדבר על כתיבת תוכניות ושיכתובם עם כל צורך נוסף של הבנק (למשל הכנסת ניהול ניירות ערך לבנק, מיסוי במקור של רווחים וכו'). גרם לכפילויות מידע ולמידע לא עיקבי.

מה אם העברנו את שליחת המכתבים ללקוחות למיקור חוץ? אם ניתן להם את קובץ הלקוחות יש כאן חשיפת נתונים רגישים (כמו מאזן כספי בחשבונות) לחברה שאינה חלק מהבנק. כדי לפתור קשיים אלה ולנהל את הנתונים ביתר יעילות, חשבו על המושג של **בסיס נתונים**, שיהיה גמיש מספיק, וייעל את מלאכת אחזקת, שימור ועידכון הנתונים.

- איזכור של בסיסי נתונים הירארכים (קשר פיזי בין חלקיו הקרויים סגמנטים של נתונים) שקדמו למודל היחסי של טבלאות, למשל **IMS**



כל מלבן ניקרא סגמנט, שימו לב שרשומה לוגית מורכבת מאוסף של כל הסגמנטים. מלבד הסגמנט של השורש, כל סגמנט אחר יכול להכיל מספר מופעים (או כלום). אם 'נשטח' את המבנה, תיווצר רשומה בודדת עבור לקוח. שמירה של רשומה 'שטוחה' - תגרום לביזבוז מקום וגם תאלץ אותנו לתת מגבלות, למשל, לאדם לא יכולים להיות יותר מ- 4 חשבונות חיסכון, אחרת זה בלתי נישלט.

עוד יתרון, למשל, יש לנו בבנק לקוח חשוב שיש לו חשבון חיסכון של מעל 100 מליון שקל, אבל לפקידי הבנק של העו"ש הוא לא רוצה לחשוף מידע זה. ניתן לתת לפקידים הרשאה לטפל רק בעו"ש ואפילו לא לראות את חשבונות החיסכון. (ההרשאה ברמת הסגמנט)

בסיס נתונים זה עובד בצורה מהירה ביותר בגלל הקשר הפיזי (מצביעים) בין הסגמנטים.

חסרונות של IMS ודומיו

הניהול הוא ברמת הסגמנטים, כך שאם למשל נירצה להוסיף שדה לסגמנט, ניצטרך לשכתב את כל התוכניות שעיבדו סגמנט זה.

השליפה של נתונים דורשת תוכנית מיוחדת עבור כל סוג של נתון. זה אומר שצריך לתכנן מראש את כל סוגי השליפות הרצויות, וקשה להוסיף סוגים אחרים. זה כמעט בלתי אפשרי לצפות מראש את כל השימושים העתידיים בבסיס הנתונים.

13.10.2015

המודל של בסיס נתונים יחסי התפתח כצורך לפתור את הבעיות שיצר המודל ההירארכי.
דוגמאות ליתרונות של מסד נתונים יחסי לעומת קבצים כמו שליפת עמודות מסוימות, הסתרת חלק מהמידע במידת הצורך, ניהול פנימי של הנתונים ע״י המנגנון (מנוע) של מסד הנתונים עצמו ללא התערבות התוכניתן.

עוד אספקט של מתכנני בסיס הנתונים היחסיים היה ליצור שפה פשוטה יחסית, כדי לשלוף נתונים שונים ומשונים מהבסיס, ללא צורך להיות מדעני מחשבים.

יתרונות של מסד נתונים לעומת קובץ סידרתי

(מסד נתונים זה כמו אוסף לא רק של נתונים, אלא גם של סידורם ואירגונם)

1. החלוקה לשדות (עמודות) נותנת אפשרות להרחבה בעתיד ללא פגיעה או שינוי של תוכניות קיימות. בקובץ סידרתי הקריאה לרשומה כוללת את כל השדות. יום בהיר אחד החליטו להוסיף עמודת ותק למשל, כי רוצים לכתוב תוכנית שתחלק פרסים לאלה שעובדים מעל 5 שנים.

   יש כרגע תוכנית שמחשבת את ממוצע השכר. היא לא צריכה להשתנות כתוצאה מתוספת העמודה.

2. הכנסת רשומה נוספת לקובץ ממוין דורשת מיון מחדש. אם מסד הנתונים (הטבלה) מאורגן במיון על פי שדה מסוים (עמודה), ברגע הכנסת הרשומה, היא מסתדרת במקומה הטבעי.

3. נתונים רגישים, אם למשל קובץ העובדים מכיל משכורות, או קובץ של חברים בפייסבוק מכיל סיסמאות, איננו רוצים שכל תוכניתן יוכל לראות את כל השדות - אפשר לפצל רשומה על פני טבלאות שונות ולתת הרשאות שונות לאנשים שונים.

4. ניתן להגדיר אינדקסים על שדות שונים בקלות ובצורה סטנדרטית בעת בניית מסד הנתונים.
5. שפה יחסית קלה ונגישה לעיבוד ושליפת נתונים, ללא צורך לכתוב תוכניות בשפת תכנות מורכבת.

חסרונות

זמן רב לתכנן מסד נתונים יעיל ומאורגן היטב. אנשי תחזוקה מיוחדים רק עבור המסד DBA.

מושגי יסוד של המודל היחסי
לפני שניגע במשהו מעשי ונתרגל על המחשב, אציין שבסיסי הנתונים מסוג SQL למעשה

מתייחסים להרבה טעמים שונים, כלומר ווריאציות שונות שלכולן יש מכנה משותף ולכן מה שנילמד יהיה רלוונטי לכולם. חלק מבסיסי הנתונים מבוססי SQL ניקראים:

אוראקל, DB2, MySql, Postgres, Informix, SQL Server, MS Access ועוד.

אנחנו נתמקד בשלבה מעשי, בבסיס הנתונים SQL Server של חברת מיקרוסופט.

קצת היסטוריה של בסיסי נתונים יחסיים

ב-1970 הציג לראשונה את המודל, דוקטור E.F TED. אחד הספרים הראשונים שמתאר את המודל של: CODD ו- DATE. בשנת 1974 הוצגה שפת SQL. בשנת 1978 הוצגה מערכת בשם System/R של CODD. ובשנת 1986 הוצג סטנדרד אמריקאי (אנסי) עבור בסיס הנתונים DB2 של IBM. הסטוריה אישית שלי, בחיל האוויר הוצג DB2 בשנת 1984. (ב י.ב.מ הוכרז ב- 1983). בשנת 1986 גם הוצגה מערכת נוספת של חברת Relational Software, שיותר מאוחר הפכה לחברת Oracle.

מושגים בסיסיים

עמודה - זה גם ניקרא שדה לעיתים קרובות - ממש כמו בקבצים, יחידת המידע הקטנה ביותר אליה ניתן להתיחס בקלות. לעיתים הן ניקראות מאפיינים (attributes)
שורה - אוסף של עמודות שמאוגדות על פי הגיון כלשהו (כמו רשומות בקובץ)
טבלה - אוסף בלתי סדור של שורות. זהו המבנה הבסיסי ביותר שניתן לשמור (כמו קובץ)

בסיס הנתונים זהו אוסף של טבלאות שמייצג גם את הנתונים שברצוננו לשמור וגם את הקשר בין כל סוגי הנתונים שקיימים.

אוסף הנתונים שמצוי בכל רגע ורגע בבסיס הנתונים (דבר שמשתנה כמובן), ניקרא: מופע (instance) , המבנה של בסיס הנתונים ניקרא : **סכמה (schema)**. וזה דבר שלרוב אינו משתנה. אלה הם הקשרים הלוגיים והיחסים בין הטבלאות של בסיס הנתונים.

ER - Entity Relationship (לעיתים תראו גם ERM , ה- M עבור מודל, או ERD, ה- D עבור דיאגרמה) - זהו בעצם אוסף של עצמים (אובייקטים) עם היחסים ביניהם. בעברית זה, אוסף הטבלאות והקשרים ביניהן.

לדוגמה: בסיס נתונים מכיל 2 טבלאות, אחת להזמנות כלליות והשניה לפירוט הזמנות. עבור כל שורת הזמנה כללית יהיו אחת או יותר שורות של פירוט בטבלה שניה - קשר של אחד לרבים. יתכנו קשרים של אחד לאחד, רבים לרבים וכו׳.

עבור ידע כללי, אזכיר שני מודלים נוספים של שמירת נתונים שקיימים היום:
1) Object Relational Data Model - זה בגלל ההתפתחות של תכנות מונחה עצמים ב-25 שנה האחרונות ושילוב בין מה שניקרא: Object oriented data model ו- Relational data model.

2) Semi-structured data model. אם מישהו מכיר XML - שפה שמשמשת לשמור נתונים מאותו סוג, בעלי תכונות שונות.

13.10.2015

מה זה SQL - Structured Query Language. זהו איחוד של שתי שפות: שפה להגדרת בסיס נתונים, ושפה לעיבוד נתונים מתוך בסיס נתונים קיים.

אנחנו בעיקרון אמורים ללמוד את השפה של עיבוד הנתונים, כי לרוב, גם מתכנתים ובוודאי משתמשי קצה אינם מגדירים בסיס נתונים. זהו תפקיד מאד מיוחד של קבוצה שניקראת DBA.

אבל, בכל זאת נצטרך גם לגעת בשפה שמגדירה את בסיס הנתונים, אחרת לא יהיה לנו עם מה לעבוד. אולם את רוב הזמן נקדיש לעיבוד הנתונים ולא לבניית מסד הנתונים.

- מעבר על מילות המפתח העיקריות בשפת SQL, CREATE, SELECT, DELETE UPDATE, DROP, GRANT, REVOKE, ALTER
- דוגמאות לטבלאות וקביעת **מפתח ראשי ייחודי** (Unique Primary Key), מהם שדות (עמודות) המועמדות להיות מפתח ראשי?

```
+----+----------+-----+-------------+----------+
| ID | NAME     | AGE | ADDRESS     | SALARY   |
+----+----------+-----+-------------+----------+
| 1  | Ramesh   | 32  | Ahmedabad   | 2000.00  |
| 2  | Khilan   | 25  | Delhi       | 1500.00  |
| 3  | kaushik  | 23  | Kota        | 2000.00  |
+----+----------+-----+-------------+----------+
```

יכול להיות מצב שמפתח ראשי יהיה מורכב ממספר עמודות. למשל, אם יש טבלת הזמנות מפורט, ובכל הזמנה יכולים להיות מספר פריטים מסוגים שונים (שורה עבור כל סוג פריט). כדי להגיע להזמנת הברגים בתוך הזמנה מספר: 7777, נצטרך גם את מס׳ השורה הספציפית. זה ניקרא Composite key.

Order_number	Line_number	Item_name	Quantity
7777	001	widget	12
7777	002	screws	15

עבור טבלה זו נשתמש ב- order_number וב- Line_number כמפתח ראשי

- מהו מפתח זר (Foreign key) ? הראנו שתי טבלאות, אחת של לקוחות והשניה של הזמנות והמפתח הראשי בטבלת הלקוחות: Customer_ID הופיע כשדה בטבלת ההזמנות ונחשב שם (בהזמנות) כמפתח זר. זהו הקשר הלוגי בין הטבלאות.

לקוחות והזמנות בטבלאות שקשורות לוגית

- באופן כללי מסד נתונים יחסי (Relational Database), הוא אוסף של טבלאות שהקשר ביניהן הוא קשר לוגי (אין הצבעה פיזית בין שורות של טבלה אחת לאחרת). הקשר, אם קיים, מתבצע בעזרת השוואת ערכים. בדוגמה הנ״ל, אם ישנה הזמנה בעלת זיהוי 00234 ובתוכה השדה של זיהוי לקוח הוא: 001 , ע״מ ולשלוף את נתוניו האישיים של הלקוח, לוקחים את המפתח הזר (002) ולפיו מחפשים בטבלת הלקוחות (שם הוא מפתח ראשי).

Order table

order_id	item_name	quantity	customer_id
00234	Mac laptop	3	002
06555	Flashlight	10	002
09811	Pencil	17	001

Customer table

customer_id	credit_card
001	2368871717178
002	1872993655449
003	1872993655440

<u>יום מספר 2</u>

כאמור, החלק של **בנית בסיס נתונים** יהיה שולי עבורנו ולא נתעכב יותר מידי על כל האפשרויות.

(מכאן נפנה להוראות השימוש ב- SQL Server)

ניצור בסיס נתונים חדש וניקרא לו: workplace1 (גראפית ועל ידי פקודה ניצור workplace2 ונימחקו)

```
create database workplace2
```

דוגמה **ליצירת טבלה** בבסיס נתונים שהגדרנו קודם, בשם workplace1 :

```
USE workplace1;

CREATE TABLE  Person
(
personID int NOT NULL,
lastName varchar(30) NOT NULL,
firstName varchar(25) NOT NULL

)
```

ברירת המחדל של קידומת הטבלה היא dbo, ולכן תיווצר טבלה בשם: dbo.person
כעת ניתבונן בעזרת design על מבנה הטבלה.

דוגמה **לשינוי הטבלה** הנ"ל והוספת עמודה (שינוי הסכמה):

```
USE workplace1;
ALTER TABLE dbo.Person
ADD Gender char(1) NOT NULL;
```

דוגמה **להוספת עמודה** מחושבת:

```
USE workplace1;
ALTER TABLE dbo.Person
ADD fullName AS lastName+', '+firstName;
```

לפני שנעשה שינויים יותר משמעותיים בטבלה, ניראה כיצד להכניס נתונים:

<u>דוגמה להכנסת נתונים לטבלה:</u>

```
INSERT INTO Person
(
personID, lastName, firstName
)
VALUES
(274, 'Kidman', 'Nicole'),
(275, 'Einstein', 'Albert'),
(276, 'Cohen', 'Yossi')
```

כעת ניפתח ב- edit (של 200 שורות) את הטבלה ונתבונן. שיו לב לעמודה המחושבת של שם פרטי ומשפחה - לא היינו צריכים להכניס אותה.

פעולה של מחיקת שורות:

```
use workplace1
delete from person
```

מקווה ששמרתם את השאילתה שמכניסה נתונים - כעת נבצעה שנית, כדי להחזיר השורות. ונבצע מחיקה סלקטיבית:

```
use workplace1
delete from person
where personID = 274
```

בידקו את התוצאה, וכעת נשתמש בשאילתת ה- insert כדי להחזיר את השורה האבודה.

לפני השינוי הבא, הציגו את הטבלה ב- design view והתבוננו בשורת ה- personID (במיוחד בצידה השמאלי), זיכרו איך היא ניראת.

הוספת אילוץ כדוגמת מפתח ראשי על עמודה מסוימת (יכולנו, אגב, להגדיר את המפתח בבניה המקורית עם ה- CREATE):

```
USE workplace1;
ALTER TABLE Person
ADD CONSTRAINT PK_PersonID
PRIMARY KEY (personID);
```

שוב פיתחו את הטבלה ב- design view וניראה מי מבחין בשינוי.

על מנת ליצור ממש בסיס נתונים, נגדיר טבלה נוספת בשם: department שתהיה מורכבת מ- department_code ו- department_name. כלהלן:

```
CREATE TABLE departmenr  - - שגיאה מכוונת בשם הטבלה
(
department_code char(2) not null,
deparment_name varchar(40),
primary key(deparment_code)      - - הפעם עם מפתח ראשי מלכתחילה
)
```

וננסה להכניס את הנתונים הבאים:

```
INSERT INTO dbo.department
(
department_name, department_code
)
VALUES
('human resources', '01'),
('marketing','02')
```

זה יכשל, כי טעינו בשם הטבלה שהגדרנו, ולכן נימחק את הטבלה השגויה על ידי:

```
drop table dbo.departmenr
```

נתקן את שאילתת היצירה של department ונריצה שנית, ואח"כ את ה- insert

בנוסף, נעשה עוד שינוי בטבלת אנשים (person), ונוסיף קוד מחלקה עבור כל עובד:

```
USE workplace1;
ALTER TABLE person
ADD deparment_code char(2);
```

זה יאפשר לנו גם לתרגל את **פקודת ה- update** של sql:

```
use workplace1
update dbo.person
set deparment_code='01'
where personID=274
```

ניתן לעדכן מספר עמודות ביחד על ידי ציון העמודות וערכן החדש (שמוחק את הישן), מופרדות בפסיקים כגון:

```
use workplace1
update dbo.person
set deparment_code='01',
    firstName = 'Joseph'
where personID=276
```

ועכשיו כבר ניתן ליצור קשר לוגי ,'מחייב' בין הטבלאות על ידי תוספת של עוד אילוץ של מפתח זר.

```
USE workplace1;

ALTER TABLE person
ADD CONSTRAINT FK_department_To_person_On_department_code
FOREIGN KEY (deparment_code)
REFERENCES department(department_code);
```

כעת נסו לבצע עריכה של הטבלה person (edit top 200) ונסו למשל לשנות את המחלקה של ניקול קידמן ל- 03. מה קרה? גם משפט update מסוג זה יכשל כאן.
יצרנו קשר גורדי בין הטבלאות, וזה מתחיל להיראות ממש כבסיס נתונים אמיתי ואפקטבי.

באופן כללי יש לזכור **שמערכת אילוצים** של מפתחות זרים, שאנו כופים על בסיס הנתונים, מיועדת ע"מ לשמור על **אמינותם של הנתונים**, או מה שנקרא באופן כללי: Data Integrity

היינו יכולים עדיין לעבוד עם טבלאות, ללא יצירת מפתחות זרים, רק עם ההסכמה על מה מייצגת כל עמודה בכל טבלה. במילים אחרות - מפתחות זרים אינם דרושם כדי לקרוא ולאחזר נתונים מקושרים לוגית, למשל מחלקה עבור אדם או כלהאנשים עבור חקה מסוימת.

נחזור עוד מעט לנושא של אילוצים, אבל קודם נבצע שאילתה מסוג select (מה שיהיה הלחם והחמאה שלנו בתירגולים). כל מה שצריך לדעת בעצם על create, alter, drop, insert, update ו- delete אתם כבר יודעים, אולם אתם אינכם יודעים כמעט כלום עדיין ב- SQL.

שאילתות שליפה לדוגמה:

```
select firstName
from  dbo.person
where personID = 276
```

על פי אותה לוגיקה, אני מבקש שתכתבו שאילתת שליפה מטבלת המחלקות של שם המחלקה עבור מחלקה: 02

ננסה עוד: כל העובדים שמספר העובד שלהם קטן מ-276 (כמה כאלה יש?)

```
select count(*)
```

```
from dbo.person
where personID <276
```

מספר העובדים שמספר העובד שלהם גדול מ - 274.

שימו לב שניתן לבחור את השאילתה לביצוע בעזרת סימונה עם העכבר (מתוך קבוצה). מה שמוחזר מהשאילתה ניקרא: Result Set. אלה יכולים להיות 0 או יותר שורות.

<u>טיפוסי נתונים</u>

אנחנו נדון בעיקר בטיפוסי כמו: int (שלמים), char (תוים בגודל קבוע), varchar (תוים באורך משתנה)

ישנם עוד הרבה מאד טיפוסי נתונים ויתכן שנזכיר עוד בהמשך ואולי גם נשתמש באחרים, אולם הסוגים המצויינים הנ"ל יספיקו בעיקרון לרוב הדיונים והתרגילים שעליהם נעבוד.
ישנו ערך מיוחד שמסמל **"אין ערך" וניקרא: null**. עבור שדות מסוימים אם נירצה מצב שבו הם יכולים לא להכיל שום ערך, אפשר לקבוע בזמן יצירת הטבלה שנאפשר עבורם null.
למשל, אם אנחנו יצרני מכוניות, ויש לנו בסיס נתונים עם טבלה שבה שומרים את נתוני המכונית שיצרנו, ורוצים לעדכנה לפני שצבעו אותה, העמודה של הצבע יכולה להכיל null.
ערך ה- null שונה משדה ריק או מאפס, מבחינת בסיס הנתונים, ובהמשך ניראה מה השוני.

<u>אילוצים נוספים בשימוש</u>

UNIQUE - אילוץ שיכול קצת לבלבל, ניתן להוסיף אילוץ זה על כל עמודה שרוצים (כדאי לשקול בהגיון), כך שבסיס הנתונים לא ירשה הכנסת שורות עם אותו ערך בעמודה זו. זה מבלבל כי העמודה אינה חייבת להיות מפתח ראשי. מפתח ראשי גם הוא ייחודי, אבל לא צריך לציין לגביו: UNIQUE
Not null - מאלץ את מי שמכניס נתונים לטבלה (שורות), לתת ערך לשדות שמוגדרים עם אילוץ זה. למשל, טבלה של מכוניות במישרד הרישוי, שדה של לוחית רישוי תמיד חייב להכיל ערך.
Default- במידה ונירצה ששדה יכיל ערך, גם אם לא טרחו להכניס אליו. למשל, בטבלת עובד יש שדה בשם: מספר ילדים. אם לא ניתן ערך, היינו רוצים שהשדה יכיל 0.
גם קביעת שדה כמפתח ראשי, או מפתח זר, הם אילוצים.
Check - זה מיועד כדי לבדוק בעת הכנסת ערך, שהוא תואם לדרישות שקבענו, למשל אם יש שדה של גיל עבור חברים בפייסבוק, אפשר לאלץ שדה זה להכיל ערכים גדולים או שוים ל 13.

לדוגמה:

```
CREATE TABLE   Facebook_users(
        ID            INT NOT NULL,
        NAME        VARCHAR (20) NOT NULL,
        AGE          INT NOT NULL CHECK (AGE >= 13),
        ADDRESS  CHAR (25) ,
        PRIMARY KEY (ID)
);
```

נושא האילוצים הוא יותר **מתפקידו של מתכנן בסיס הנתונים**, שעובד על פי הדרישות הפונקציונליות מהמערכת שאותה מקימים. מפתחי אפליקציות ומשתמשי קצה, מקבלים את מבנה הבסיס כעובדה מוגמרת.

עבודה על הטבלאות של ספריית הסטודנטים (SQLQuery_create_library)

זוהי מערכת טבלאות (בסיס נתונים) שבעזרתה ניתן לנהל ספרית סטודנטים ונשתמש בה כדי לתרגל כל מיני שאילתות (**בעיקר פעולות של SELECT**)

תאור הנתונים וחלוקתם לטבלאות:

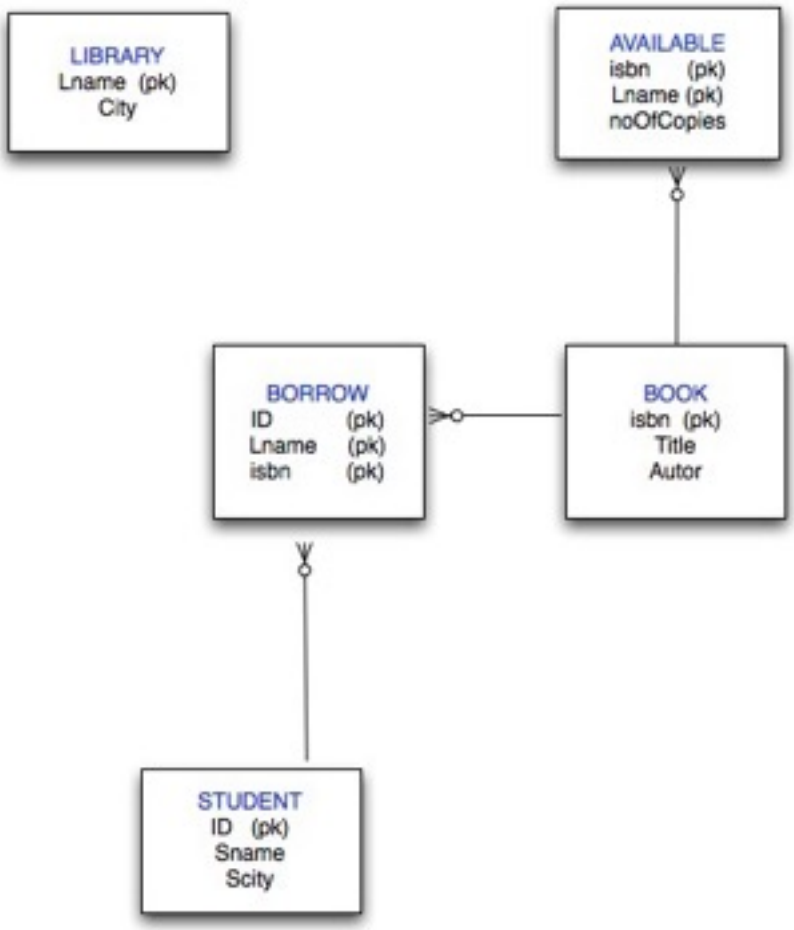
1. Library - טבלה זו מכילה את שם הספריה ואת העיר בה היא ממוקמת (Lname, City) - כל הספריות.

2. Book - טבלה זו מכילה נתונים עבור ספרים, מספר זיהוי ספר (ISBN), שם (Title) ושם מחבר (Author) אלה כל הספרים הקיימים בין אם נימצאים במלאי של ספריות או לא.

3. Available - טבלה שמכילה את הספרים הזמינים בכל רגע נתון בכל ספריה (0 או יותר). העמודות הן זיהוי ספר (ISBN), שם ספריה ומספר עותקים של הספר בספריה שבשורה.

4. Borrow - עבור כל ספר בהשאלה, טבלה זו מכילה שורה של הסטודנט ששאל/ה אותו ומאיזו ספריה הושאל. שימו לב שמספר ספרים עם אותו מספר זיהוי יכולים להיות מושאלים מספריה אחת (לספריה יתכנו מספר עותקים של אותו ספר).

5. Student - טבלה שמכילה את כל הסטודנטים, מספר זיהוי סטודנט/ית, שם ועיר שבה הסטודנט/ית גר/ה. שימו לב שהעמודה לעיר ניקראת אחרת מאשר ב-Library



זה יראה בערך כמו בדיאגרמה מסוג ERD הנ"ל. אין שורות כפולות בטבלאות

סטודנט אחד יכול לשאול כמה ספרים (או כלום), ספר יכול להיות זמין בהרבה ספריות, בכמויות שונות או לא זמין באף אחת, או אפילו לא מוחזק באף אחת. סטודנט יכול לגור בעיר שיש בה ספריות או שאין. עותקים מאותו ספר יכולים להיות מושאלים לסטודנטים שונים, שימו לב שבטבלת ההשאלות כל שלושת העמודות יחד מהוות מפתח יחודי, בטבלת הזמינות מס׳ זיהוי ספר + שם ספריה (אותו ספר יכול להיות בכמה ספריות).

נתחיל במספר שאילתות קלות, אבל בעלות משמעות.
מצא את כל מספרי הזיהוי של הספרים שזמינים בספריה כלשהי בכמות של 2 ומעלה והצג רשימה שלהם וליד כל ספר את הכמות.

```
select ISBN, noOfCopies
from   Available
where noOfCopies >= 2
```

מצא כמה ספרים מושאלים לסטודנט בעל מספר זהות 099 (שמוש ב- as לשם עמודה בתוצאה)

```
select count(*) as “NUmber of Books” - - or count(isbn)
from  Borrow
where ID = 099
```

מה ההבדל בין count(*) ו- count(column) ? אם יש ערכי nullבעמודה הם לא יספרו בדרך השניה.

<u>פונקציות צבירת נתונים</u>

count, sum, max, min , avg

אלה ניקראות גם פונקציות סקלאר, שעבורן בעצם מקבלים שורה אחת, למשל:

```
use library
select avg(noOfCopies) as 'Average no of copies'
from dbo.Available

use library
select max(noOfCopies) as 'Maximum no of copies'
from dbo.Available

use library
select min(noOfCopies) as 'Minimum no of copies'
from dbo.Available

use library
select sum(noOfCopies) as 'Total no of copies available'
from dbo.Available
```

אפשר גם להוסיף תנאים בעזרת where כדי לגביל את התחום של פונקציות הערך הבודד הללו. למשל, נסו להוסיף לשאילתה האחרונה: where isbn > '3333333333'

JOIN - שליפה מיותר מטבלה אחת, על בסיס ערך משותף

שאילתה שכוללת שילוב של שתי טבלאות. ע"מ למצוא את מספרי הזיהוי של הספרים שמצד אחד הושאלו על ידי מישהו, כלומר מופיעים בטבלה: Borrow, ומצד שני עדיין קיימים עותקים שלהם זמינים באחת או יותר מהספריות.

```
select a.ISBN
from Available a, Borrow b
where a.ISBN = b.ISBN
and a.noOfCopies > 0
```

זה בעצם אומר תן לי את הספרים (מספר זיהוי של) שמופיעים בשתי הטבלאות ועבור כל צירוף כזה בדוק אם מספר העותקים שנותר עבור הספר בטבלה Available הוא גדול מאפס (כלומר יש עותק פנוי).

זוהי פעולה שניקראת Join, של שתי טבלאות על השדה ISBN שבמקרה זה הוא חלק מהמפתח בשתי הטבלאות (אבל לא חייב להיות).

ראינו כאן עוד אספקט שבו כינינו לשם קיצור את הטבלאות a ו- b , זה ניקרא Alias (או שם נירדף), כדי לחסוך בכתיבת שם הטבלה על מנת לזהות שדות במדוייק. יש לציין שאם השדה מופיע רק בטבלה אחת (כמו במקרה של noOfCopies) אפשר להשמיט את שם או כינוי הטבלה ורק לכתוב:

```
and  noOfCopies > 0
```

על מנת להבין טוב יותר את תהליך ה- join ניראה דוגמה של מה קורה בפועל. ניקח ספר בעל isbn של: '0824774566' וננסה לבדוק אם הוא נימצא גם ב- borrow וגם ב- available (כלומר זמין ומושאל). נעשה זאת בעזרת join:

```
select av.isbn
from  dbo.available av, dbo.borrow bo
where av.isbn = bo.isbn
and av.isbn = '0824774566'
```

מה קיבלנו? (אם בסיס הנתונים נכון תהינה 12 שורות חזרה).
איך זה קרה ומהאפשר לעשות כדי להתגבר על הבעיה?
זה קרה בגלל התכונה של join שהוא מבצע מה שניקרא "מכפלה קרטזית' של טבלה בטבלה. כלומר על כל מופע בטבלה אחת, הוא מאחזר את כל המופעים התואמים בטבלה השניה. אם תריצו את השאילתות הבאות, הכל יתבהר:

```
select * from dbo.available where isbn = '0824774566'
select * from dbo.borrow where isbn = '0824774566'
```

שימרו בבקשה את השאילה הנ"ל. נילמד בהמשך כיצד להתגבר על תופעה כזאת.

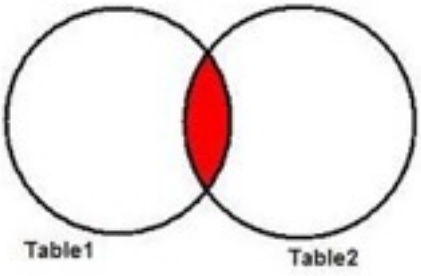
התחביר של שאילתת ה- Join הנ"ל אינו היחיד האפשרי, ניתן ליכתוב:

```
select a.isbn
```

```
from Available a inner join Borrow b
on a.isbn=b.isbn
```

מילת המפתח: inner, מציינת שזוהי שליפה לנתונים שנימצאים בשתי הטבלאות גם יחד. בהמשך ניראה גם מה הכוונה בביצוע outer join.

בצורה וויזואלית כדי להבין את פעולת ה-join הפנימי, הנה איור:



פעולת ה- Join היא פעולה מאד בסיסית ב-SQL ולמעשה מדגימה את העוצמה שיש בבסיס נתונים יחסי להרכיב צירוף נתונים רצוי, על ידי שילוב של מספר טבלאות (אין הגבלה מעשית למספר הטבלאות ב- join)

תחשבו איך לשנות את השאילתה הנ"ל ע"מ להציג את שם הספר במקום את מספר הזיהוי שלו.

כעת ניראה דוגמה של join על 3 טבלאות, מתי צריך אותו ואיך לבצע אותו:
נניח שנירצה עבור כל שם ספר בטבלה book לשלוף את שם הספריה שבה הוא זמין ב-2 עותקים לפחות ואת מספרי הזהות של מי מהסטודנטים ששאלו אותו. יש לנו בקשה לעמודות מ-3 טבלאות:

השם זהו title (book), הספריה בה הוא זמין מעל 2 noOfCopies >2 (available), ומספר זהות הסטודנטים ששאלו אותו - ID (borrow).

ניראה זאת בשתי הצורות (תחילה הישנה שבאופן אישי חביבה עלי):

```
select bk.title, av.Lname, bo.ID
from dbo.available av,
      dbo.borrow    bo,
      dbo.book      bk
where
      av.isbn = bo.isbn
and  a.isbn  =  bk.isbn
and  noOfCopies > 2
```

ובעזרת מילות המפתח: inner join

```
select bk.title, av.Lname, bo.ID
from dbo.available av  inner join
      dbo.borrow    bo
on    av.isbn = bo.isbn inner join
      dbo.book      bk
on    av.isbn = bk.isbn
```

כעת נחזור לשאילתה שניקשתי מכם לשמור (עם מספר הספר שהופיע 12 פעמים) ונוסיף מילת מפתח שתגרום למספרלהופיע פעם אחת בלבד. מילת המפתח: DISTINCT

13.10.2015

לפני שנפנה לתרגילים, אם נירצה למיין את התוצאה של שאילתה ניכתוב:

```
order by col1, col2, …DESC  or ASC
```

- תרגילי כיתה

1. לכתוב שאילתה (Query) שתתן לנו את כל מספרי הזיהוי של הספרים שנימצאים במלאי של ספריה אחת לפחות ממוינים מהקטן לגדול.

2. לכתוב שאילתה (Query) שתתן את רשימת מספרי זיהוי כל הסטודנטים ששאלו ספר של הסופר: Bernard Cushing או של הסופר .Sherman Chow (ללא חזרות) בסדר יורד.

מהו VIEW?

נניח שיש לנו בסיס נתונים שנוצר מזמן, אבל ברוות הימים, גילינו שיש לנו צורך תכוף להשתמש בחלק מטבלה, או במספר טבלאות ביחד, בצורה סלקטיבית - ורק על המבנה הזה אנו רוצים לכתוב שאילתות. האם יש איזו אפשרות ליצור מבנה חדש בצורה אבסטרקטית שיענה על דרישותינו מבלי לבקש מצוות ה- DBA לשנות את בסיס הנתונים הקיים? התשובה היא : כן.

משפט של SQL שנישמר בבסיס הנתונים ומשוייך לשם כלשהו (שמו של ה- view). זהו "מראה" של נתונים. זה מאפשר יתר גמישות עבור משתמשים לעבוד על מה שהם צריכים. זה יכול להיות חלק מטבלה (חלק מהשורות או העמודות או גם וגם), כמו גם צירוף של נתונים מטבלאות שונות.

התחביר ליצירה הוא כדלקמן:

```
CREATE VIEW view_name AS
SELECT col1, col2
FROM tbl1
```

לאחר שגדרנו את ה "מראה" - פשר להתיחס אליו כמו לטבלה רגילה בבסיס הנתונים, לשאול שאילתות ולעדכן. כל שינוי ב "מראה", ישנה גם את הטבלה/אות, עליהן הוא מבוסס.

ניתן לבנות את ה "מראה" עם אופציה שלא תאפשר לעדכנו עם נתונים שאינם יכולים להיכלל בו. לדוגמה, אם משפט ה- SQL שיצר אותו הגביל שדה בשם age שיהיה מעל 18, וה- view ניבנה עם האופציה: **with check option** כל נסיון להכניס רשומה עם גיל קטן מ-18 או לעדכן רשומה קיימת עם אותו דבר - יכשל.

יום מספר 3

הראנו שפעולת ה- join שביצענו על טבלאות Borrow ו- Available ניקראת גם **INNER join**. בהמשך ניראה סוגים שונים של join.
עוד דרך למשל לכתוב את השאילתה שכתבנו בשבוע שעבר תהיה:

```
SELECT a.ISBN
FROM Available a INNER JOIN Borrow b
ON a.ISBN = b.ISBN
WHERE a.noOfCopies > 0
```

ניתן בשאילתה הנ"ל להשמיט את הקידומת של noOfCopies (ה .a) כי השדה נימצא רק באחת מהטבלאות, לעומת זאת את ה- a.ISBN לא ניתן להשמיט.

SELECT COUNT(column_name) FROM table_name

ספירת שורות שעבורן column_name אינו null (לעומת count(*) שסופר את כל השורות)

הערה בתוך שאילתה ניתנת לכתיבה אם משמאלה מופיעים שני מקפים, לדוגמה:

```
select *
from book  - - This is a comment
```

עוד מילת מפתח : **between** שמאפשרת לבחור שורות בטווח מסויים (שני הצדדים כלולים בטווח), למשל:

select ISBN from book where isbn between '6666666666' and '9999999999'

שימוש ב- IN

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

שימוש במילת המפתח - **LIKE** - שני תוים בשימוש _ (קו תחתון) ו- %. קו תחתון מייצג תו בודד כלשהו ו-% מייצג בין 0 לאינספור. למשל, מצא את כל הסטודנטים ששמם מתחיל באותיות: Ma, או מצא את כל הספריות שנימצאות בערים שיש בשם העיר האות y. (LIKE 'Ma%', LIKE '%y%')

יש גם את האפשרות לבחירת תו ממחרוזת: LIKE '[abcd]%' אחד מ- a, b או c ואח"כ כל דבר.
אפשרי גם להוציא תוים מסויימים על ידי: LIKE '[^abc]_'

קומבינציה של AND ו- OR

```
select * from CUSTOMER
where LastName='Smith' and (FirstName='John' or FirstName='Smith')
```

עבור שיפור ביצועים

SELECT TOP number/percent

```
SELECT TOP 2 * FROM Customers
```

```
SELECT TOP 50 PERCENT * FROM Customers
```

נחזור על שימוש במילת המפתח - **DISTINCT**

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

כך נימנע משליפת שורות כפולות. דוגמה לשימוש:
שלוף רשימה של כל הערים שבהן גרים סטודנטים, ללא חזרה על עיר פעמיים או יותר.

```
select distinct scity
from dbo.student
```

פקודת UNION מאפשרת לנו לאחד יותר משאילתה אחת. שתיהן צריכות להיות זהות מבנית (אותו מספר עמודות ורצוי אותו סוג ערכים) כדי שזה יעבוד. הן יכולות להיות מטבלאות שונות. זהלא כ"כ נפוץ ולכן ניראה רק דוגמה טריביאלית שעבורה UNION לא ממש נחוץ:

קודם ניצור את בסיס הנתונים workplace1 ע"י הרצת: (workplace1_person_create)

```
use workplace1
create table person
(
  personID int not null,
  lastName varchar(30),
  firstName varchar(20)
)
```

ונכניס לו נתונים (אם עדיין לא קיים) בעזרת:
SQLQuery_workplace1_insert_person.sql

```
use workplace1
insert into person
(personID, lastName, firstName, gender)
values
(274,'Kidman', 'Nicole','f'),
(275,'Einstein', 'Albert','m'),
(276,'Cohen', 'Yossi','m')
```

וכעת לפעולת ה- UNION

```
select fullname as NAME
from dbo.person
where personID = 274
union
select fullname as NAME
from dbo.person
where personID = 276
```

איך הייתם משיגים אותה תוצאה בדרך אחרת? (רמז: אתם כבר יודעים כמה דרכים עבור זה).

כעת נמשיך לדבר על join

מה ההבדלים בין: INNER JOIN ו- FULL JOIN RIGHT OUTER, LEFT OUTER ?

בעיקרון משתמשים בסוגים אחרים של JOIN (אחרים מ- INNER) כאשר יש מקרים שלא בטוחים באמינות הנתונים או שהנתונים באים ממקורות שונים ורוצים לתאם ביניהם.

דוגמה לשימוש:

נתונות הטבלאות הבאות:

Employee

EmpID	EmpName
13	Jason
8	Alex
3	Ram
17	Babu
25	Johnson

Location

EmpID	EmpLoc
13	San Jose
8	Los Angeles
3	Pune, India
17	Chennai, India
39	Bangalore, India

שימו לב שבטבלה Employee ישנו עובד בשם Johnson שאיננו נמצא בטבלת המקומות וכן שישנו עובד בעל מספר זיהוי 39 בטבלת המקומות שאינו נמצא בטבלת העובדים.

אם נכתוב Join פנימי (INNER JOIN) על מספר זיהוי עובד, יושמטו עובדים 25 ו-39. נסו זאת.

אבל אנחנו מעוניינים להתאים בין שמות העובדים ומיקומם, ורוצים לדעת גם אם לא ידוע מיקומו של עובד. הטכניקה תהיה להשתמש במשהו שניקרא: LEFT JOIN או LEFT OUTER JOIN

לדוגמה:

```
select * from employee e
left outer join location l
on e.empID = l.empID;
```

אפשר גם להשמיט את מילת המפתח: OUTER ולכתוב:

```
select * from employee e
left join location l
on e.empID = l.empID;
```

התוצאה תהיה:

e.EmpID	e.EmpName	l.EmpID	l.EmpLoc
13	Jason	13	San Jose
8	Alex	8	Los Angeles
3	Ram	3	Pune, India
17	Babu	17	Chennai, India
25	Johnson	NULL	NULL

המילה LEFT מרמזת לו שמהטבלה שנימצאת בצד שמאל של ה- join ילקחו גם ערכים שאינם תואמים לטבלה מימין.
דבר דומה יקרה אם ניכתוב:

```
select * from employee e
right outer join location l
on e.empID = l.empID;
```

e.EmpID	e.EmpName	e.EmpID	l.EmpLoc
13	Jason	13	San Jose
8	Alex	8	Los Angeles
3	Ram	3	Pune, India
17	Babu	17	Chennai, India
NULL	NULL	39	Bangalore, India

תנחשו מה יקרה אם ניכתוב:

```
select * from employee e
full outer join location l
on e.empID = l.empID;
```

<u>פונקציות נוספות</u>

```
SELECT x = SUBSTRING('abcdef', 2, 3);
```

התוצאה של x תהיה: bcd

```
USE pubs;
SELECT pub_id, SUBSTRING(logo, 1, 10) AS logo,
       SUBSTRING(pr_info, 1, 10) AS pr_info
FROM   pub_info
WHERE  pub_id = '1756';
```

בחירה של חלק מעמודה. שימושי כאשר העמודה היא מחרוזת. הפרמטר הראשון הוא העמודה,השני מיקום - מתחיל לספור משמאל (מבוסס על 1 כתו השמאלי ביותר) והפרמטר השלישי הוא האורך הרצוי.

החזר את 5 התוים השמאליים של העמודה:

```
SELECT LEFT(Name, 5)
FROM Production.Product
ORDER BY ProductID;
```

ועבור תוים מימין יש כמובן את: RIGHT

- LTRIM/RTRIM - מיועדות לחתוך רווחים מימים או משמאל של תוכן העמודה.

דוגמאות להסבת סוגי נתונים מסוג אחד לאחר:

```
-- Use CAST
USE AdventureWorks2012;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CAST(ListPrice AS int) LIKE '3%';
GO


-- Use CONVERT.
USE AdventureWorks2012;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CONVERT(int, ListPrice) LIKE '3%';
GO
```

<u>GROUP BY</u>

נניח שרצינו לדעת מהטבלה של הספריות שמשיכת סיפריה לעיר, כמה ספריות נימצאות בכל עיר בטבלה. על ידי הסתכלות, אנחנו יכולים להבחין שניו יורק די שולטת, אבל אנחנו רוצים מספר מדוייק של כולן.

ישנו מבנה מיוחד ב- SQL שמאפשר **צבירה של שורות על פי שוויון בשדה מסוים**, ולמעשה הצגה של סיכום. במקרה שלנו תחשבו על זה כעל מצב שבו אנחנו ׳מכווצים את הטבלה׳ לשורות סכום עבור ערים, כלומר ליד כל עיר, ניתן רק את מספר הספריות שבה, ללא פרטים נוספים.

הטכניקה היא על ידי כתיבת שאילתה מהסוג הבא:

```
select city, count(*) as 'Number of libraries'
from dbo.library
group by city
```

יש לזכור שבשאילתה מסוג זה יכולים להישלף אך ורק השדות עליהם מסכמים (במקרה הזה שדה אחד ושמו **city**) ובנוסף שדה הסיכום עצמו.

תחשבו אם יש הגיון בנסיון לשלוף באותה שאילתה גם את שם הספריה.

נוסיף עוד טוויסט לעלילה. כעת רוצם אותו דבר כמו קודם, אולם נירצה להציג רק את הערים בהן יש לפחות 2 ספריות.

```
select city, count(*) as 'Number of libraries'
from dbo.library
group by city
having count(*) >=2
```

טוויסט נוסף בעלילה: בנוסף למה שהיה עד כה, נירצה גם את הספריות רק מערים ששמן מתחיל ב- S זה משהו שכבר למדנו, אבל כעת אתם צריכים לשלב גם את ה- group by וגם את ה- where. תנסו.

```
select city, count(*) as 'Number of libraries'
from dbo.library
where city like 'S%'
group by city
having count(*) >=2
```

למי זה הצליח? אם כן היכן מיקמתם את משפט ה where?

תזכרו שה- having עבור ה- group by הוא למעשה כמו ה- where עבור ה- select. פשוט מגביל את הבחירה. ולכן הם גם צריכים להיות צמודים בהתאמה, כ״א אחד לאן שהוא שייך.

ישנו עוד סוג של group by מלבד ספירת מספר השורות שעונות על שדה הסיכום. זה קורה במצב של עמודות שניתנות לסיכום. לדוגמה בטבלה Available ישנו שדה שניקרא: noOfCopies. אם נירצה לדעת עבור כל ספר (מספר זיהוי) כמה עותקים ישנם ממנו , בכל הספריות הקיימות - איך לדעתכם תיראה השאילתה?

ובאותה נשימה , את כל הספרים עבורם יש יותר מ-4 עותקים כ״א.