גירסה 1.01 – 2.7.2005

שפת SQL

ניר אדר

Nir Adar

Email: underwar@hotmail.com

Home Page: http://www.underwar.co.il

אנא שלחו תיקונים והערות אל המחבר.

תוכן עניינים

1. מבוא – בסיסי נתונים ושפת SQL

מסמך זה מציג את שפת SQL, ומדגיש את הקשר שלה אל בניית אתרים. המסמך אינו מניח ידע מוקדם בשפה.

1.1. אתרי אינטרנט ובסיסי נתונים

בראשית ימי האינטרנט, רוב האתרים באינטרנט היו **אתרים סטטיים** – האתרים הורכבו ממספר דפי HTML שהיו מקושרים אחד לשני. הסטאטיות של האתרים התבטאה בכך שכל גולש שנכנס אל האתר, ראה בדיוק את אותו הדף. שפת HTML מאפשרת לבונה הדף לעצב טקסט כרצונו, אולם בכל פעם שמשתמש ייכנס אל אותו הדף, הוא יראה בדיוק אותו דבר.

אתרים סטאטיים מאפשרים לבעל האתר להציג מידע לגולש, אולם עם התפתחות האינטרנט וגדילת אתרים, מופיעים יותר ויותר **אתרים דינאמיים** – המציגים מידע אישי לכל גולש. הדוגמא הנפוצה ביותר היא כל אתרי דואר הרשת, כגון hotmail ואחרים. Hotmail הינו אתר המציג לכל גולש תוכן דינמי – את האימיילים השייכים לו.

כיצד יוצרים אתר דינמי? בעזרת שימוש בשפות הפועלות על השרת, השולחות מידע לכל משתמש.
השפות הפופולריות ביותר לצד שרת הן PHP, ASP ולאחרונה גם ASP.Net צוברת תאוצה.
שפות הפועלות בצד השרת מבצעות עיבוד על נתונים שונים בצד השרת, ואז שולחות את התוצאה הסופית אל המשתמש.

כאשר אנחנו בונים אתר דינמי, מתעורר צורך לשמור נתונים על השרת – הנתונים יכולים להיות רשימת המשתמשים הרשאים להכנס לאיזור מסויים בשרת, רשימת הודעות שנשלחו בפורום הנמצא באתר, רשימת קבצים הנמצאת באתר והדוגמאות עוד רבות.
ניתן לשמור מידע זה בקובץ בצד השרת, אולם כיום נהוג לשמור את המידע בתוך בסיס נתונים. בסיסי הנתונים הנפוצים ביותר הם בסיסי נתונים טבלאיים התומכים בגישה אליהם על ידי שפת SQL.
בסיס נתונים טבלאי הוא בסיס נתונים בו המידע נשמר כטבלאות. כל אלמנט מידע מיוצג כשורה בטבלה.
בבסיס נתונים אחד יכולים להיות מספר טבלאות.

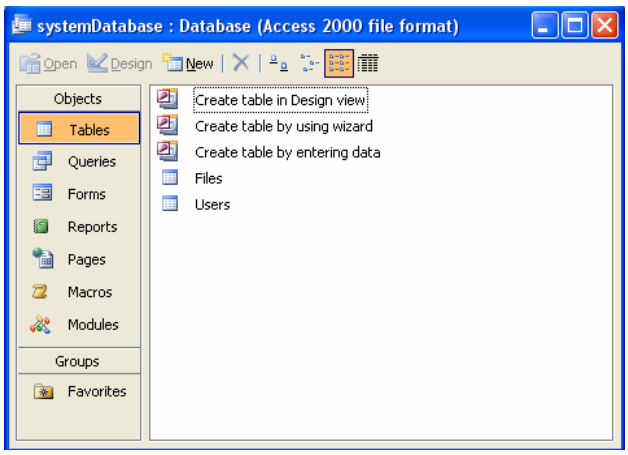
שפת השרת בה אנחנו משתמשים ניגשת אל הטבלאות השמורות על ידי פקודות בשפת SQL, ושולחת נתונים מהן אל המשתמש.

1.2. דוגמא לבסיס נתונים

על מנת להציג את הטבלאות נשתמש בתוכנת Access המאפשרת ליצור בסיסי נתונים.

נציג דוגמא לבסיס נתונים מוכן ב-Access. במסמך זה לא נציג כיצד יוצרים אחד כזה.
בסיס הנתונים שנציג מורכב משתי טבלאות:

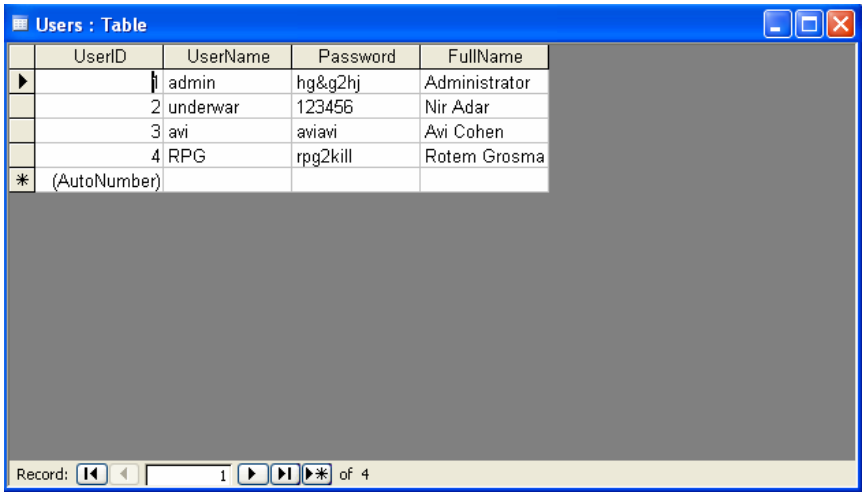
- Users – השומרת את נתוני המשתמשים שלנו.
- Files – השומרת מידע על הקבצים השייכים לכל משתמש.



בתמונה ניתן לראות את שמות הטבלאות: Files, Users, הנמצאות בתוך בסיס הנתונים.

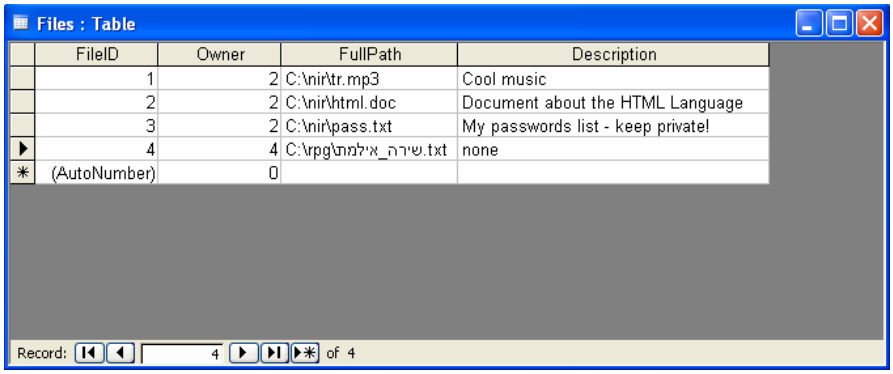
הערה: בונה בסיס הנתונים יכול לתת לטבלה שם כרצונו.

תוכן לדוגמא עבור טבלת המשתמשים:



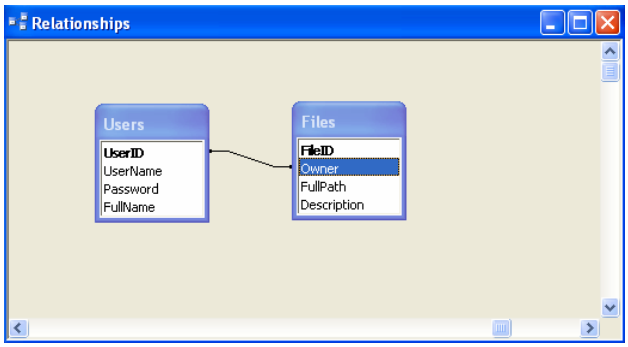
עבור כל משתמש נשמר מספר מזהה יחודי (UserID), שם המשתמש (UserName), סיסמת המשתמש (Password) ושמו המלא.

דוגמא לטבלת הקבצים:



עבור כל קובץ נשמר מזהה יחודי עבור הקובץ (FileID), המזהה של המשתמש שהוא הבעלים של הקובץ (Owner), הספריה בה נמצא הקובץ (FullPath) ותיאור של הקובץ (Description).

נשים לב כי Owner מתייחס למשתמש הנמצא בטבלה Users. צורת עבודה זו נפוצה ביותר בעבודה מול בסיסי נתונים.



השדה Owner בטבלת הקבצים מתאים לשדה מזהה המשתמש בטבלת ה-Users.

1.3. שפת SQL

כדי שבונה האתר יוכל לתקשר עם בסיס הנתונים שלו, הוא צריך שפה, ממשק, שיאפשר לו לגשת אל בסיס הנתונים, ולבצע עליו פעולות. שפה זו הינה לרוב שפת SQL.
קיימים מערכות בסיסי נתונים רבים התומכים בשפת SQL, כגון MySQL, MS-SQL ועוד.
נשים לב להבדל בין השפה למערכות: השפה מוגדרת וקיימת בלי קשר אל מערכות בסיסי הנתונים.
מערכות בסיסי הנתונים הן תוכנות, שרתים, המסוגלים להבין את שפת SQL.

הפקודות העיקריות של השפה הן Select, Insert, Update, שמשמעותן: בחירת (שליפת) נתונים מתוך בסיס הנתונים, הכנסת נתונים חדשים אל בסיס הנתונים ועדכון נתונים קיימים. נציג כעת פקודות אלו ואחרות.

הפקודות והמילים השמורות של שפת SQL יכתבו במסמך זה באותיות גדולות. אם זאת, אין חובה לכתוב אותם באותיות גדולות. שפת SQL הינה case-insensitive, כלומר הפקודות יכולות להיות באותיות גדולות או קטנות כרצוננו. הסיבה שנכתוב את המילים השייכות לשפה באותיות גדולות היא על מנת שיהיה קל להבחין מהן המילים השייכות לשפה, ומה ספציפיות לדוגמא.

SELECT .1.3.1

פקודת ה-SELECT משמשת לבחירת רשומות מסויימות מתוך טבלה הנמצאת מבסיס הנתונים.
על מנת לבחור נתונים, עלינו לציין את שם הטבלה ואת העמודות שאנו רוצים לשלוף.

לדוגמא:

```
SELECT column1, column2, ...
```

על מנת להגיד שאנחנו רצים לבחור את כל העמודות של טבלה מסויימת, ניתן להשתמש באופרטור *:

```
SELECT *
```

המילה from באה לציין את הטבלה ממנה אנחנו רוצים לשלוף את הנתונים. לדוגמא, הפקודה הבאה תחזיר לנו את כל העמודות השייכות לטבלה בשם tableName:

```
SELECT * FROM tableName
```

where .1.3.1.1

הוספת תנאי הקובע אילו מהשורות בטבלה יכללו בתוצאה שתוחזר אלינו.
התנאי הינו תנאי בוליאני – כל שורה שהתנאי מתקיים עבורה בתוצאה אותה תחזיר השאילתה.

ניתן לעשות תנאי המורכב ממספר תנאים, ולחברם על ידי OR, AND ו-NOT.
ניתן בכל תנאי להשתמש באופרטורים הבאים:

שוויון	=
אי שוויון	<>
גדול מ-	>
קטן מ-	<
גדול שווה	>=
קטן שווה	<=
דומה ל-	like
בין לבין	between ... and ...

דוגמא 1:

בחר את כל השורות בטבלה tableUsers בהן הערך של השדה username הוא UnderWarrior והסיסמא היא abcde:

```
SELECT * FROM tableUsers WHERE username='UnderWarrior' AND
password='abcde'
```

דוגמא 2:

בחר את כל השורות בטבלה tblSalaries בהן ערך השדה Salary גדול ממש מ-1000:

```
SELECT * FROM tblSalaries WHERE Salary>1000
```

אם נניח כי הטבלה tblSalaries נראית כך:

UserName	Salary
Moshe	1000
David	1200
Eynat	900
Nir	9999

אז תוצאת השאילתה תהיה:

UserName	Salary
David	1200
Nir	9999

דוגמאות נוספות לשאילתות מורכבות יותר יוצגו מיד, לאחר שנכיר את שאר הפקודות של שפת SQL.

order by .1.3.1.2

המילים השמורות ORDER BY משמשות לסידור התוצאות לפי ערכן בעמודה מסויימת.

השאילתה הבאה למשל תחזיר את כל הנתונים מהטבלה tableUsers ממויינים לפי השדה username:

```
SELECT * FROM tableUsers ORDER BY username
```

1.3.1.3. שליפת נתונים ממספר טבלאות

ניתן לשלוף בשאילתה אחת נתונים ממספר טבלאות. במקרה זה נציין ב-from את הטבלאות מופרדות על ידי פסיקים, ובציון השמות נציין את שם הטבלה, נקודה, ואז את שם השדה שאנחנו רוצים לקחת מאותה טבלה, לדוגמא:

```
SELECT * FROM table1.columnName1, table1.columnName2,
table2.columnName3 FROM table1, table2
```

1.3.1.4. ערכים חסרים

NULL – ערך מיוחד המציין ערך חסר בטבלה - "משבצת ריקה". לדוגמא: תאריך החזרה של ספר שעדיין לא הוחזר.

השוואה ל-NULL:

- expr IS NULL – מחזיר true אם expr הוא ערך NULL
- expr IS NOT NULL

דוגמא: שליפת כל הספרים שעוד לא הוחזרו.

```
SELECT Book_Id FROM Borrowed WHERE To_Date IS NULL
```

בעיה: ערך של ביטוי אריתמטי או ביטוי לוגי המכיל NULL תמיד יהיה NULL.

הפתרון: קביעת "ברירת מחדל" עבור ערך שיכול להיות NULL: COALESCE(value, default)

דוגמא: הצגת משך ההשאלה בשבועות של כל הספרים (עבור ספרים שעדיין לא הוחזרו יש להציג 0):

```
SELECT Book_Id , COALESCE(To_Date – From_Date, 0) / 7 FROM Borrowed;
```

בצורה כללית: COALESCE(expr1,expr2,…) מחזירה את הערך הראשון השונה מ-NULL.

INSERT .1.3.2

הפקודה insert משמשת אותנו לצורך הכנסת נתונים חדשים אל הטבלה.

תחביר:

```
INSERT INTO table_name VALUES (value1, value2,....)
```

הנתונים יוכנסו אל הטבלה לפי סדר העמודות שבה. value1 יוכנס אל התא הראשון שבשורה בטבלה. value2 יוכנס אל התא השני, וכו'.

ניתן גם לציין במפורש לאילו עמודות אנו רוצים להכניס נתונים (ובאיזה סדר) על ידי ציון שם העמודה בצמוד לשם הטבלה, לפי התחביר הבא:

```
INSERT INTO table_name (column1, column2,...) VALUES (value1,
value2,....)
```

נשים לב שניתן על ידי שימוש בתחביר זה לא להכניס נתונים אל כל השדות, אלא רק לחלק (על ידי ציון שמות רק חלק מהעמודות והכנסת ערכים רק לעמודות אלו). במידה ומבנה הטבלה מאפשר זאת, השדות האחרים יישארו ריקים.

דוגמא: נניח כי הטבלה tblPersons הינה הטבלה הבאה:

LastName	FirstName	City
Adar	Nir	Haifa
Cohen	Moshe	Eilat

נפעיל את הפקודה הבאה:

```
INSERT INTO tblPersons VALUES ('Grossman', 'Rotem', 'Ofakim')
```

ונקבל בטבלה את הנתונים הבאים:

LastName	FirstName	City
Adar	Nir	Haifa
Cohen	Moshe	Eilat
Grossman	Rotem	Ofakim

אם נריץ כעת את הפקודה:

```
INSERT INTO tblPersons (LastName, City) VALUES ('Zion', 'Jerusalem')
```

נקבל את הטבלה:

LastName	FirstName	City
Adar	Nir	Haifa
Cohen	Moshe	Eilat
Grossman	Rotem	Ofakim
Zion		Jerusalem

UPDATE .1.3.3

הפקודה UPDATE משמשת אותנו על מנת לעדכן נתונים הקיימים כבר בטבלה. תחביר הפקודה:

```
UPDATE table_name SET column1 = [new value] WHERE condition
```

נעדכן כעת את הטבלה שהצגנו בדוגמא הקודמת.

```
UPDATE tblPersons SET City = 'Tel-Aviv' WHERE LastName='Adar'
```

ונקבל את הטבלה החדשה:

LastName	FirstName	City
Adar	Nir	Tel-Aviv
Cohen	Moshe	Eilat
Grossman	Rotem	Ofakim
Zion		Jerusalem

נשים לב כי בדוגמא זו רק שורה אחת קיימה את התנאי, ולכן רק שורה אחת עודכנה. במידה והיו מספר שורות המקיימות את התנאי, כולן היו מתעדכנות.

אם נרצה נוכל גם לעדכן מספר שדות בו זמנית על ידי פקודת UPDATE אחת. התחביר במקרה זה הינו:

```
UPDATE TABLE table_name SET (column_1, column_2) = ([new value 1],
[new value 2]) WHERE condition
```

CREATE TABLE .1.3.4

הפקודה CREATE TABLE משמשת אותנו ליצירת טבלאות חדשות.
נזכיר כי כפי שהראנו בסיס נתונים אחד יכול להיות מורכב ממספר טבלאות. פקודה זו משמשת להוספת טבלה חדשה אל בסיס הנתונים.

כדי להוסיף טבלה חדשה נבין קודם טיפה יותר את רעיון הטבלה. הטבלה מורכבת מעמודות ושורות.
כל שורה מכילה את נתונים של איבר בודד – למשל בטבלה השומרת נתונים על בני אדם, שורה אחת תכיל נתונים על בן אדם יחיד.
כל עמודה בטבלה מכילה נתונים מאותו סוג – למשל עמודה אחת מכילה את שמות המשפחה של האנשים שנתוניהם שמורים בטבלה, עמודה אחרת את גילם, וכו'. בכל עמודה יש נתונים **מסוג** אחד בלבד. איזה סוגים של נתונים קיימים? מספרים שלמים (לדוגמא 41), מספרים ממשיים (3.12), מחרוזות (לדוגמא המחרוזת 'hello') ועוד.
כאשר אנחנו יוצרים טבלה חדשה, עלינו לציין: שמות העמודות שאנו רוצים שיהיו בטבלה, ואת הסוג של כל אחת מן העמודות.

תחביר הפקודה CREATE:

```
CREATE TABLE table_name (column1 data_type_for_column_1, column2
data_type_for_column_2, ...)
```

דוגמא:

```
CREATE TABLE tblPersons (FirstName char(50), LastName char(50), City
char(50), Birth_Date date)
```

char(50) הינה מחרוזת שאורכה המקסימאלי הינו 50.

DROP TABLE .1.3.5

הפקודה DROP משמשת על מנת למחוק טבלה מבסיס הנתונים. הגדרת הטבלה (העמודות שבה) וכל הנתונים שבה ימחקו ללא אפשרות לשחזרם.

תחביר הפקודה:

```
DROP TABLE table_name
```

לדוגמא: מחיקת הטבלה tblPersons מבסיס הנתונים:

```
DROP TABLE tblPersons
```

TRUNCATE TABLE .1.3.6

הפקודה TRUNCATE TABLE משמשת כדי למחוק את כל הנתונים של טבלה, מבלי למחוק את הטבלה עצמה כמו שהפקודה DROP עושה.

תחביר הפקודה:

```
TRUNCATE TABLE table_name
```

לדוגמא: מחיקת כל נתוני הטבלה tblPersons:

```
TRUNCATE TABLE tblPersons
```

LIKE .1.3.7

את LIKE הצגנו בקצרה קודם כאשר הראנו תנאים שאפשר להפעיל על שאילתה, אבל לא הדגמנו את פעולתה. LIKE מאפשרת לנו להגדיר תבנית לחיפוש, במקום להגדיר ערך מדויק שאנחנו רוצים.

תחביר:

```
SELECT "column_name" FROM "table_name" WHERE "column_name" LIKE
{PATTERN}
```

כאשר {PATTERN} היא תבנית המכילה wildcards.

wildcards אפשריים הם % שמשמעותו "0 או יותר תווים כלשהם" והסימן _ שמשמעותו "תו יחיד כלשהו".

דוגמאות:

- 'A_Z' – כל המחרוזות המתחילות באות A, לאחריה תו כלשהו ואז המחרוזת נגמרת בתו Z. לדוגמא: 'ABZ', 'A9Z' אבל לא 'ATTZ'.
- 'ABC%' – כל המחרוזות המתחילות ב-ABC, לדוגמא: 'ABCD' או 'ABCDE'.
- '%XYZ' – כל המחרוזות הנגמרות ב-'XYZ'.
- '%AN%' כל המחרוזות בהן הצירוף AN קיים בחלק כלשהו של המחרוזת.

נדגים שימוש ב-LIKE. נביט בטבלה הבאה (ששמה Store_Information):

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

נריץ את השאילתה:

```
SELECT * FROM Store_Information WHERE store_name LIKE '%AN%'
```

ונקבל:

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999

1.3.8. פונקציות בשפת SQL

SQL תומכת בכמה פונקציות שניתן לבצע על הטבלאות. הפונקציות כוללות חישוב ממוצע על טור, מציאת מקסימום, מינימום וכו'. הפונקציות הקיימות הן:

- AVG
- COUNT
- MAX
- MIN
- SUM

התחביר של השימוש בהן הוא:

```
SELECT <function type>(column_name) FROM table_name
```

נביט למשל בטבלה הבאה (ששמה Store_Information):

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

אם נפעיל את השאילתה:

```
SELECT SUM(Sales) FROM Store_Information
```

נקבל טבלה עם תא אחד, בו הערך $2750. (סכום כל ה-Sales: $1500+250+300+700=2750$).

הפונקציה COUNT מחזירה לנו את מספר הכניסות הקיימות עבור עמודה מסויימת.

```
SELECT COUNT(store_name) FROM Store_Information
```

הערך המוחזר משאילתה זו הינו 4.

ניתן לשלב גם את המילה DISTINCT בהגדרה כדי לבטל תוצאות כפולות:

```
SELECT COUNT(DISTINCT store_name) FROM Store_Information
```

הערך שיוחזר במקרה זה הינו 3.

כל הפונקציות הסטטיסטיות מתעלמות מערכי NULL. היוצא מן הכלל: COUNT(*).

1.3.9. GROUP BY – פעולה על קבוצות של רשומות

לאחר שראינו את הפונקציות של שפת SQL, נרצה לעשות איתן דברים שימושיים. בדוגמא של SUM השתמשנו בפונקציה כדי לסכום את המכירות של כל החנויות שבבסיס הנתונים. אבל מה אם נרצה לסכום את המכירות עבור כל חנות בנפרד?

כדי לעשות זאת, עלינו לעשות שני דברים: 1. עלינו לבחור, בנוסף לסכום המכירות, גם את שם החנות. כמו כן נרצה לדאוג לכך שהפונקציה SUM בכל פעם תפעל רק על קבוצת השורות בהן יש את אותו שם חנות. כדי לעשות זאת אנחנו משתמשים ב-GROUP BY.

GROUP BY מאפשרת לנו להפעיל את הפונקציות של שפת SQL על קבוצות של רשומות.

תחביר לדוגמא:

```
SELECT column_name1, SUM(column_name2) FROM table_name
GROUP BY column_name1
```

לדוגמא, עבור טבלת המכירות שראינו קודם:

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

נפעיל את השאילתה הבאה:

```
SELECT store_name, SUM(Sales) FROM Store_Information GROUP BY
store_name
```

ונקבל את טבלת התוצאה הבאה:

store_name	SUM(Sales)
Los Angeles	$1800
San Diego	$250
Boston	$700

כלל: כאשר אנחנו משתמשים ב-GROUP BY, בנוסף לפעולות סטטיסטיות (הפונקציות השונות), מותר לשלוף רק שדות לפיהם מתבצע הקיבוץ (וביטויים שמערבים אותם).

HAVING .1.3.10

אופציה נוספת שנרצה היא להגביל את העמודות שמוחזרות כתוצאה משימוש ב-SUM או בכל אחת מן הפונקציות האחרות. למשל נרצה את כל החנויות שלהן יותר מהכנסה מסוימת. ניתן לבצע זאת על ידי המילה HAVING.

המילה HAVING מחליפה למעשה את המילה WHERE המופיעה בשאילתות שאינן כוללות פונקציות של SQL. המשמעות של HAVING הינה בחירת חלק מהקבוצות המתקבלות מ-GROUP BY.

תחביר:

```
SELECT column_name1, SUM(column_name2) FROM table_name
GROUP BY column_name1 HAVING (arithmetic function condition)
```

לדוגמא, עבור טבלת המכירות:

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

נפעיל את השאילתה הבאה:

```
SELECT store_name, SUM(sales) FROM Store_Information GROUP BY
store_name HAVING SUM(sales) > 1500
```

ונקבל את טבלת התוצאה הבאה:

store_name	SUM(Sales)
Los Angeles	$1800

1.3.11. WHERE לעומת HAVING

- **WHERE**: בחירת רשומות לפני הקיבוץ.
- **HAVING**: בחירת קבוצות אחרי הקיבוץ.

דוגמא: (לקוחה מהקורס בטכניון "מבוא למסדי נתונים").

המשימה: בין כל הספרים עם יותר מ-200 עמודים, חשב בכל שנה את מס' הספרים שיצאו לאור, בתנאי שבממוצע מספר העמודים באותה שנה גדול מ- 400.

השאילתה המתאימה נראית כך:

```
SELECT Year, COUNT(Book_Id) FROM Books WHERE Pages > 200
GROUP BY Year HAVING AVG(Pages) > 400
```

הסבר (שלבים):

- בחירת הרשומות של הספרים שמכילים יותר מ- 200 עמודים.
- קיבוץ הרשומות לקבוצות, כך שבכל קבוצה יש לכל הרשומות (הספרים) אותו Year.
- חישוב מספר העמודים הממוצע בכל קבוצה, ובחירת הקבוצות בהן הממוצע הוא לפחות 400.
- חישוב מס' הספרים בכל קבוצה.

INNER JOIN .1.3.12

נציג כעת נושא חדש – איחוד טבלאות.

ניזכר בבסיס הנתונים שראינו בדוגמא בפרק 1.2. הטבלאות הינן:

Users			
UserID	**UserName**	**Password**	**FullName**
1	admin	hg&g2hj	Administrator
2	underwar	123456	Nir Adar
3	avi	aviavi	Avi Cohen
4	RPG	rpg2kill	Rotem Grosman

Files			
FileID	**Owner**	**FullPath**	**Description**
1	2	C:\nir\tr.mp3	Cool music
2	2	C:\nir\html.doc	Document about the HTML Language
3	2	C:\nir\pass.txt	My passwords list - keep private!
4	4	C:\rpg\שירה_אילמת.txt	none

לשתי טבלאות שדה "משותף" – Users.UserID מתאים ל-Files.Owner.

נניח כעת כי אנחנו רוצים לעשות שאילתה שתחזיר לנו טבלה חדשה, שתכיל עבור כל קובץ את ה-ID שלו, את ה-path, ובנוסף את שם המשתמש המתאים לו.

מה שאנחנו בעצם רוצים לבצע זה איחוד של הטבלאות לפי שדה מסויים.

נעשה זאת על ידי המילה INNER JOIN. עבור הדוגמא ההיא, נכתוב:

```
SELECT Files.FileID, Files.FullPath, Users.UserName
FROM Users INNER JOIN Files ON Users.UserID = Files.Owner;
```

והתוצאה:

FileID	FullPath	UserName
1	C:\nir\tr.mp3	underwar
2	C:\nir\html.doc	underwar
3	C:\nir\pass.txt	underwar
4	C:\rpg\שירה_אילמת.txt	RPG

Inner Join מחזיר תוצאות משתי הטבלאות עבורן יש התאמה. אם יש שורה עבורה אין התאמה בטבלאות, היא לא תוחזר.

תחביר נוסף ל-inner join הוא כזה:

```
SELECT Files.FileID, Files.FullPath, Users.UserName
FROM Users, Files WHERE Users.UserID = Files.Owner;
```

LEFT JOIN .1.3.13

LEFT JOIN עובד בצורה דומה ל-INNER JOIN, אבל אם בטבלה השמאלית קיים נתון שעבורו לא קיים נתון מתאים בטבלה הימנית, הוא עדיין ייכלל.

נדגים שוב עם שאילתה הזהה לשאילתה הקודמת, מלבד שינוי ה-INNER JOIN ל-LEFT JOIN. נציין כי במקרה זה אין לתוצאה יותר מדי משמעות, אולם יש מקומות רבים בהם ניתן להשתמש בסוג חיבור טבלאות כזה. אנחנו מביאים פה דוגמא זו רק כדי שיהיה קל להשוות בין התוצאות של ה-JOIN השונים.

השאילתה החדשה:

```
SELECT Files.FileID, Files.FullPath, Users.UserName
FROM Users LEFT JOIN Files ON Users.UserID = Files.Owner;
```

התוצאה של השאילתה הינה:

FileID	FullPath	UserName
		admin
3	C:\nir\pass.txt	underwar
2	C:\nir\html.doc	underwar
1	C:\nir\tr.mp3	underwar
		avi
4	C:\rpg\שירה_אילמת.txt	RPG

המשתמשים עבורם לא מוגדרים קבצים הופיעו גם בטבלה.

RIGHT JOIN .1.3.14

בצורה מקבילה לחלוטין לפעולתה של LEFT JOIN, ניתן להשתמש גם ב-RIGHT JOIN על מנת לאחד טבלאות.

UNION ALL-ו UNION .1.3.15

UNION הינו סוג נוסף של איחוד. UNION מאפשר לנו לאחד תוצאות של שתי שאילתות שונות (ולא כמו שעשינו עד כה – איחוד טבלאות שונות לטבלה אחת).

התחביר של UNION:

```
[SQL Statement 1]
UNION
[SQL Statement 2]
```

ניקח לדוגמא את הטבלאות הבאות:

Employees_Haifa:

EmployeeID	Name
01	Adar, Nir
02	Grossman, Rotem
03	Ofnik, Moshe
04	Cohen, Moshe

Employees_TelAviv:

EmployeeID	Name
01	Cohen, Moshe
02	Levi, Oren
03	Tal, Eynat
04	Cohen, David

נריץ את השאילתה הבאה:

```
SELECT Name FROM Employees_Haifa
UNION
SELECT Name FROM Employees_TelAviv
```

התוצאה שנקבל תהיה:

Name
Adar, Nir
Grossman, Rotem
Ofnik, Moshe
Cohen, Moshe
Levi, Oren
Tal, Eynat
Cohen, David

נשים לב שתוצאות כפולות לא הופיעו פעמיים. על ידי שימוש ב-UNION אנחנו מקבלים כל ערך רק פעם אחת.

כדי לקבל את כל התוצאות, נשתמש ב-UNION ALL.

התחביר של UNION ALL:

```
[SQL Statement 1]
UNION ALL
[SQL Statement 2]
```

לדוגמא:

```
SELECT Name FROM Employees_Haifa
UNION ALL
SELECT Name FROM Employees_TelAviv
```

במקרה זה הטבלה תכיל את העובדים המופיעים ב-2 הטבלאות פעמיים.

1.3.16. שמות נרדפים - Alias

המטרה: הגדרת שם נוסף (בד"כ קצר יותר) לטבלה.

דוגמא:

```
SELECT C.Cust_Id, Cust_Name, Book_Name
FROM Customer C, Ordered O
WHERE C.Cust_Id = O.Cust_Id
```

Customer קיבל את השם הנרדף C ו-Ordered קיבל את השם O.

שימוש נוסף: שימוש במספר עותקים של אותה טבלה. לדוגמא:

```
SELECT DISTINCT C1.Cust_Name FROM Customer C1, Customer C2
WHERE C1.Cust_Name = C2.Cust_Name AND C1.Cust_Id <> C2.Cust_Id;
```

השאילתה מחזירה את כל הלקוחות כך שעבורם מתקיים שיש לקוח נוסף עם אותו שם.

שימוש נוסף: לתת שם לביטוי סטטיסטי או אריתמטי. לדוגמא:

```
SELECT Year, COUNT(Book_Id) AS Num_Books FROM Books GROUP BY Year;
```

COUNT(Book_Id) יקבל את השם Num_Books בטבלה שתוחזר כתוצאה מהפעלת השאילתה.

1.3.17. תתי-שאילתות – SUBQUERIES

מוטיבציה: איפשור מקרים בהם התנאי ב-WHERE או ב-FROM מכיל ביטוי שלא ידוע מראש, אלא תלוי בתוכן המסד.

הערה: הדוגמאות בפרק זה לקוחות מחומר הקורס "מבוא למסדי נתונים" בטכניון.

1.3.17.1. תת שאילתה כתנאי ה-WHERE

דוגמא: שליפת כל הספרים שיצאו לאור באותה שנה כמו ספר מס' 1112.

```
SELECT Book_Name FROM Books
WHERE Year = (SELECT Year FROM Books WHERE Book_Id = 1112);
```

הערך המוחזר יכול להיות גם מספר ערכים, אליהם נוכל להשוות בו זמנית את האיברים, לדוגמא:

שליפת כל הספרים שיצאו לאור באותה שנה ונמצאים באותה פקולטה כמו ספר מס' 1112:

```
SELECT Book_Name FROM Books WHERE (Year, Faculty) =
(SELECT Year, Faculty FROM Books WHERE Book_Id = 1112);
```

1.3.17.2. תת שאילתה כתנאי ה-FROM

כאמור – תת שאילתה יכולה להופיע גם בחלק של ה-FROM. במקרה כזה, היא חייבת לקבל ALIAS.

לדוגמא: שליפת שמות הספרים שהוזמנו אחרי 20-Oct-98 ע"י לקוח 12345.

```
SELECT Book_Name
FROM (SELECT * FROM Ordered Where Order_Date > 20-Oct-98) O
WHERE O.Cust_Id = 1234;
```

EOF