נערך על ידי יובל שוורץ ויובל ימין

סינטקס לכתיבה- עם או בלי שימוש ב CASES (תנאים)

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    select_expr [, select_expr] ...
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
    [HAVING where_condition]
        [ORDER BY {col_name | expr | position}
    [ASC | DESC], ..]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

העמודות ב- SELECT חייבות להופיע ב- GROUP BY

```
SELECT Column name(s)
CASE
      WHEN Condition 1 THEN result1
      WHEN Condition 2 THEN result2
      ELSE Else_result
END AS [Column Name for the Cases' results]
FROM Table_name
JOIN
WHERE condition(s)
GROUP BY Column name(s)
HAVING aggregate column(s)
ORDER BY Column_name(s);
```

פונקציות ופעולות אגרגציה

פונקציות בסיסיות	
Sum()	חישוב סכום ערכים
Min()	הפונקציה מחזירה את הערך הנמוך ביותר
Max()	הפונקציה מחזירה את הערך הגבוה ביותר
Avg()	מחשב ממוצע
Count()	הפונקציה סופרת את מספר שורות
Group concat()	משרשרת את ערכי העמודה למחרוזת אחת

פונקציות שימושיות	
Now()	תאריך ושעה נוכחיים
Round(כמה ספרות להשאיר אחרי הנקודה, המספר)	עיגול מספר-
Concat(str1, str2)	חיבור מחרוזות
Length(str)	מחזיר את אורך המחרוזת
Lower/Upper	שינוי לאותיות קטנות/גדולות
Trim(str)	הסרת רווחים מיותרים
Substring(str, start, length)	קטע מהמחרוזת
Len()	מחזיר את מספר האותיות בביטוי

שימוש ב-%	
%abc%	טקסט שמכיל "abc" בכל מקום
abc%	טקסט שמתחיל ב- abc
%abc	טקסט שמסתיים ב- abc
%a_b%	טקסט מכיל "a", אחריו תו כלשהו ואחריו "b"
'abc' (% ללא)	בדיוק "abc"

סימונים בסיסיים	
AND	תנאים לוגיים
OR	
NOT	
BETWEEN (V1 AND V2)	טווח
IN (רשימה של ערכים / טווח נתון או תת שאילתא)	
LIKE	חיפוש ספציפי
IS NULL	בדיקה אם ערך ריק
Desc	מיון בסדר יורד
Asc	מיון בסדר עולה

קיבוץ ומיון	
Group by	מקבץ לפי עמודה
Having	סינון לאחר קיבוץ
Order by	מיון על פי סדר עולה/ יורד
Limit n	הגבלת תוצאות
Distinct	מסיר כפילויות

```
table1 AS t1
[INNER] JOIN / LEFT JOIN / RIGHT JOIN
table2 AS t2
ON t1.col = t2.col;
```

מחבר בין טבלאות לפי עמודות משותפות – JOIN	
Inner join	מציג רק שורות שיש התאמה בשתי הטבלאות
Left join	הכל מהטבלה השמאלית+ התאמות מהימין
Right join	ההפך מ-left
Full join	כל מה שיש בכל הטבלאות

שימוש בSELECT-נוכל להשתמש בפונק' אגרגציה

- `COUNT(*)` → כמה שורות יש.
- `COUNT(col)` → כמה ערכים קיימים בעמודה (לא כולל NULL).
- `COUNT(DISTINCT col)` → כמה ערכים שונים יש בעמודה (לא כולל NULL).

- **SELF JOIN** = טבלה שמצטרפת לעצמה באמצעות Alias שונה.
שימושי כשיש **יחסים פנימיים** באותה טבלה (למשל זוגות במאים).
מחייב שימוש ב־Alias כדי להבדיל בין ההופעות.

דוגמא ל- distinct ⟸ `SELECT DISTINCT column_name FROM table;`

דוגמא לאליאס ⟸ `SELECT name AS full_name`

ספירת ערכים ייחודיים ⟸ `COUNT (distinct A)`

VIEW הוא בעצם טבלה וירטואלית – שכפול לוגי של תוצאה משאילתה – שאפשר להתייחס אליו בדיוק כמו לטבלה רגילה.

המטרה היא להשתמש בו בתת־שאילתות או כחלק מ־JOIN מורכב, בלי להריץ שוב ושוב את אותה שאילתה.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

נוכל להשתמש ב VIEW-כדי לפשט את הקוד, ולא להשתמש ב SUBQUERY- בתשובה.

דוגמא ל- Subquery

```
SELECT t.name, t.total
FROM (
  SELECT
    name,
    SUM(amount) AS total
  FROM Transactions
  GROUP BY name
  HAVING SUM(amount) > 1000
) AS t;
```

דוגמא להכנסת מידע לטבלה קיימת

```
CREATE TABLE TableOne (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20)
);
```

```
INSERT INTO Customers (name, email)
VALUES ('value for name', 'value for email');
```

⬆

הכנסת נתונים לתוך טבלה שיצרנו

יצירת טבלאות- נשים לב בכל עמודה להגדיר את סוג הנתונים (על מנת לאפשר ביצוע פעולות על העמודות בהמשך)

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,        -- מפתח ראשי
    customer_id INT,                                -- מפתח זר
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2),

    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

אינדקסים

אינדקסים במסד נתונים הם מבנה נתונים שמאיץ את ביצוע השאילתות על ידי גישה מהירה לשורות, בדומה לאינדקס בספר שמאפשר להגיע למידע בלי לעבור על כל הדפים.

נורמליזציה

נורמליזציה היא תהליך ארגון נתונים במסד נתונים כדי לצמצם כפילויות ולמנוע חוסר עקביות, והמטרה שלה היא לשפר שלמות נתונים ויעילות תחזוקה. נעשה זאת ע"י פירוק טבלאות גדולות עם מידע חוזר לטבלאות קטנות יותר עם קשרים ביניהן.

- **1NF (צורת נרמול ראשונה):** אין ערכים מרובי־ערכים, כל שדה מכיל ערך יחיד בלבד.
- **2NF (צורת נרמול שנייה):** אין תלות חלקית בין עמודות למפתח הראשי; כל עמודה שאינה מפתח תלויה במפתח כולו.
- **3NF (צורת נרמול שלישית):** אין תלות טרנזיטיבית – כלומר, עמודות שאינן מפתח לא תלויות אחת בשנייה אלא רק במפתח הראשי.

סוגי נתונים	
מספר שלם	INT
מספר עשרוני	NUMERIC(p,s)
מחרוזת באורך משתנה	VARCHAR(n)
תאריך בלבד	DATE
תאריך+שעה	DATETIME
FALSE/TRUE	BOOLEAN

PRIMARY KEY
מזהה ייחודי לכל שורה בטבלה
לא יכול להיות בו ערך כפול (לכן UNIQUE),
או ריק (NULL),כל טבלה צריכה מפתח ראשי אחד לפחות
FOREIGN KEY
עמודה בטבלה שמצביעה על המפתח הראשי בטבלה אחרת

Data Integrity

שמירה על נתונים במסד נעשית כך שהם יהיו:

- נכונים – ערכים הגיוניים (אין מחיר שלילי).
- שלמים – אין רשומות חסרות או שבורות.
- עקביים – אין סתירות בין טבלאות (אין הזמנה ללקוח שלא קיים).
- מותאמים לחוקים העסקיים – למשל לא מזמינים מוצר שלא במלאי.

המטרה: שהמידע יהיה אמין, עדכני ונכון לקבלת החלטות. אם לא נשמור על כך – נקבל נתונים שגויים, סותרים וחסרים.

בעיות בייצוג הנתונים

- **חזרתיות** – אותו מידע נשמר פעמיים.
- **חוסר עקביות** – עדכונים לא תואמים (מידע שאינו עדכני).
- **נתונים חסרים** – שדות ריקים.
- **ערכים לא תקינים** – נתונים בפורמט שגוי.
- **כפילויות** – רשומות חוזרות.

שמירה על **Data Integrity** נעשית באמצעות הגבלות (Constraints) שונות במסד הנתונים:

1. **Primary Key** – מזהה ייחודי לכל שורה בטבלה.
2. **Foreign Key** – יוצר קשר בין טבלאות ומבטיח שלא יוזן ערך שלא קיים בטבלה הראשית.
3. **Not Null** – מחייב ששדה מסוים יכיל ערך.
4. **Unique** – מונע כפילויות בשדה מסוים.
5. **Check** – מאפשר להגדיר תנאים חוקיים לשדה.
6. **Default** – מגדיר ערך ברירת מחדל לשדה שלא מולא.

יחסים בין טבלאות- חלק מנורמליזציה

יחסים בין טבלאות נדרשים כדי למנוע כפילויות, לשמור על עקביות ושלמות הנתונים, לאפשר שליפות מורכבות, ולייעל את התחזוקה והניהול של בסיס הנתונים.

טיפים:

- כאשר רואים טבלה עם כמה שדות שמצביעים לטבלאות אחרות – בדיקה לקישור N:N.
- כאשר רואים שדה שמופיע הרבה עם אותו ערך – N:1.
- כאשר נראה יחס של 1:1 – הקשר יתווסף לטבלה קיימת של N:1 לשמירת מקום.

יחסים בין טבלאות	
1:1	**אחד לאחד**- כל רשומה בטבלה א מקושרת לרשומה אחת בלבד בטבלה ב
N:1	**אחד לרבים** – רשומה אחת בטבלה א יכולה להיקשר לרבות בטבלה ב, אך כל רשומה בטבלה ב שייכת רק לאחת בטבלה א
N:N	**רבים לרבים** – רשומות רבות מטבלה א מקושרות לרשומות רבות בטבלה ב

ייצוג נתונים במסד הנתונים

ייצוג הערכים הוא חלק מרכזי בשמירה על **Data Integrity**, כי סוג הנתון שנבחר קובע אם המידע נשמר נכון, עקבי ואמין.

- מספרים – לחישובים (INT, DECIMAL).
- טקסט – לשמות ותיאורים (CHAR, VARCHAR).
- תאריכים ושעות – לפעולות על זמן (DATE, DATETIME).
- בוליאני – לשדות של כן/לא.
- NULL – מציין שאין ערך (לא 0 ולא ריק).

באמצעות ייצוג נכון מונעים ערכים שגויים (כמו מחיר שלילי, תאריך כטקסט או כפילויות) ושומרים על אמינות הנתונים לצורך החלטות נכונות.

ייצוג ערכים – **כיצד לייצג מידע חוזק או קבוע** – כמו מדינות,מטבעות,קטגוריות,סטטוסים ,מגדר,סוגים וכו'

ייצוג לא נכון – טקסט חופשי
טעויות כתיב
קושי לסנן , לקבץ או לנתח
אין שליטה בערכים

ייצוג נכון – רשימת ערכים קבועים
ערכים מוגדרים מראש
שינוי שם של מדינה מתבצע ממקום אחד בלבד
קוד נקי וברור

תכנון מסד נתונים

- **כפילויות** → מטפלים בנורמליזציה.
- **ערכים חוזרים** → מפרקים לטבלה נפרדת עם FK.
- **שדות שחייבים ערך** → משתמשים ב־NOT NULL או DEFAULT.
- **יחסים בין טבלאות** → מגדירים FOREIGN KEY לשמירה על עקביות.
- **קשרים N:N** → שוברים בעזרת טבלת קשר (Junction Table) עם שני FK.

אלו סוגי נתונים מתאימים?

מזהים	`BIGINT` + `AUTO_INCREMENT` או `INT`
טקסט קצר	`VARCHAR(n)`
טקסט ארוך	`TEXT`
כסף	`DECIMAL(10,2)`
תאריך	`DATETIME` או `DATE`
לוגי	`BOOLEAN`