# ייצוג נתונים -

## מבוא:

עד עכשיו שלפנו נתונים מבסיסי נתונים, ועכשיו נלמד כיצד הם נוצרים ואיך לייצג אותם באופן הולם ושימושי. בסיסי נתונים נבנים מתוך צורך - לקוח מתאר דרישות, ואנחנו מתכננים בסיס נתונים שיכול לעמוד בהן. ישנם סוגים שונים של בסיסי נתונים, ולכל אחד יתרונות בהתאם לדרישות. במהלך הקורס נלמד גם תכונות שבסיס נתונים תקין צריך לעמוד בהן, כמו נרמול והימנעות מכפילויות, וכן את הבעיות שעלולות לנבוע מהפרת כללים אלו.

ייצוג נכון של נתונים הוא הבסיס לתפקוד תקין, יעיל ומדויק של המערכת. באמצעות שפת SQL מגדירים טבלאות, קשרים בין ישויות, מגבלות על שדות, ומוודאים שלמות לוגית של המידע. כל ישות (כמו לקוח, מוצר או הזמנה) מיוצגת כטבלה נפרדת, וכל טבלה כוללת שדות (עמודות) המייצגים תכונות של אותה ישות.

הגדרה נכונה של מפתחות ראשיים (Primary Keys) ומפתחות זרים (Foreign Keys) מבטיחה עקביות בין טבלאות, ומאפשרת לנו לבנות קשרים מסוג 1:1, N:1 או N:N, כאשר לכל קשר קיימת דרך מימוש ברורה – למשל באמצעות עמודת FK או יצירת טבלת קשר.

בנוסף, חשוב להוסיף אילוצים (Constraints) כמו NOT NULL, UNIQUE, CHECK ו־DEFAULT כדי להבטיח את תקינות ואיכות הנתונים, ולמנוע ערכים חסרים, כפולים או שגויים. עוד עיקרון מרכזי הוא **נרמול (Normalization)** – תהליך שמטרתו למנוע כפילויות, להפחית סתירות ולשפר את ארגון הנתונים.

באמצעות תכנון נכון - החל מזיהוי הישויות, דרך הגדרת מפתחות ותכנון קשרים ועד הוספת מגבלות - ניתן לבנות מסד נתונים יעיל, קריא ועמיד לאורך זמן.

**לאורך הסיכום נעבוד עם דוגמה ממבחן מועד א 2025**

עצבו בסיס נתונים עבור ספרית ספרים.
המערכת צריכה לתמוך בשימושים הבאים.

1. הצגת שם, סופר, תיאור ותאריך הוצאה לכל ספר.
2. הצגת מספר הכותרים (עותקים) של כל ספר בספריה.
3. הצגת כל הכותרים הנמצאים אצל קורא מסויים.
4. הצגת כל הקוראים שלא השאילו מעולם ספר.

<u>תהליך המעבר מדרישות לסכימה</u>

**שלב 1: ניתוח הדרישות העסקיות**

- נעבור על הטקסט ונבין **מה המערכת צריכה לנהל**:

ספרים? השאלות? קוראים? העתקים?

- נזהה **ישויות מרכזיות** (Entities).

עצמים (כמו ספר) ופעלים (להשאיל) הם לרוב רמזים לעצמים.

**שלב 2: הגדרת הEntities**

ישות היא דבר ממשי או לוגי שהמערכת צריכה לשמור עליו מידע נפרד (ספר, עותק, קורא, השאלה). כל ישות כזו קיבלה טבלה נפרדת (Books, Copies, Readers, Loans).

כלומר, כדי לדעת שזו ישות - שואלים: *האם נרצה לזהות אותה באופן ייחודי ולנהל עליה מידע משלה?*
**אם כן - זו ישות.**

לרוב, כל ישות היא טבלה.

**שלב 3: זיהוי מפתחות ראשיים (Primary Keys)**

- לכל טבלה צריך מזהה ייחודי (id). כשאפשר, עדיף להשתמש במפתח "מהטבע", כזה שיש לו משמעות מחוץ למערכת כמו תעודת זהות. אם אין, ניתן להשתמש במפתחות שמומצאים במערכת, לדוגמה reader_id, book_id

**שלב 4: קביעת הקשרים בין ישויות (Relationships)**

**חשוב לציין כי לפעמים זה תלוי בנו ( הקשרים)**

- איזה קשרים קיימים בין טבלאות?

1:1

N:1

N:N

**דוגמה:**

- בין ישות ספרים לישות עותקים – קשר של אחד-לרבים: כל ספר יכול להופיע בכמה עותקים, ולכל עותק יש-אך-ורק ספר אב אחד (עותק ≠ מהדורה).

**שלב 5: הגדרת עמודות ותכונות**

עבור כל ישות נגדיר את התכונות (מאפיינים) שלה. לאחר מכן נתרגם את התכונות לעמודות בטבלה.

לדוגמה - עבור ישות **BOOKS** :

- מזהה ספר (BookID)
- שם ספר (BookName)
- מחבר (Author)
- תאריך פרסום (PublishDate)

אלו התכונות של הישות, והן מתורגמות לעמודות בטבלה:
BookID (PK), BookName, Author, PublishDate

**שלב 6: הגדרת אילוצים (Constraints)**

האילוצים נועדו להגן על שלמות הנתונים ולהבטיח שימוש עקבי בהם. הם מגדירים כללים שמונעים טעויות והכנסת מידע שגוי למערכת.

**שלב 7: שרטוט התרשים (רשות)**

(ראו הרחבה בעמ 12)

יצירת טבלאות – שתי שיטות

- יצירת טבלה מתוך אחרת:

```
CREATE TABLE new_table AS SELECT * FROM old_table;
```

- יצירת טבלה חדשה עם הגדרות

```
CREATE TABLE Books (
book_id INT PRIMARY KEY,
book_name VARCHAR(100) NOT NULL,
author VARCHAR(50) NOT NULL,
publish_date DATE NOT NULL
);
```

```
CREATE TABLE loans (
    loan_id SERIAL PRIMARY KEY, -- מפתח ראשי מזהה ייחודי לכל השאלה
    reader_id INT NOT NULL, -- חייב להיות קורא
    copy_id   INT NOT NULL,   -- חייב להיות עותק ספר
    loan_date   DATE NOT NULL, -- תאריך השאלה חובה
    return_date DATE, -- תאריך החזרה (יכול להיות NULL אם עדיין לא הוחזר)
    status TEXT DEFAULT 'ongoing'
        CHECK (status IN ('ongoing', 'returned', 'late')),
        -- סטטוס ההשאלה: מתבצעת / הוחזרה / באיחור

    -- מפתחות זרים
    FOREIGN KEY (reader_id) REFERENCES readers(reader_id),
    FOREIGN KEY (copy_id)   REFERENCES copies(copy_id),

    -- אילוץ נוסף: תאריך החזרה חייב להיות שווה או אחרי תאריך ההשאלה
    CHECK (return_date IS NULL OR return_date >= loan_date)
);
```

## סוגי קשרים

### קשר 1:1 –

ספר↔ ID שלו

קשר שבו לכל שורה בטבלה אחת יש התאמה בדיוק לשורה אחת בטבלה השנייה. בפועל קשר כזה נדיר יחסית, משום שלרוב יש יותר משורה אחת המקושרת לישות אחרת. דוגמה: מספר אישי ות"ז של אדם/ ספר ומספר סידורי שלו - לכל אחד יש רק אחד ייחודי.
מימוש: לרוב נשתמש ב־FOREIGN KEY עם UNIQUE כדי להבטיח שאין יותר מתעודה אחת לאותו אזרח.

### קשר N:1 -

עותקים → ספרים

קשר שבו לשורה אחת בספרים יכולים להיות כמה שורות בעותקים,
אבל לכל עותק יש ספר אחד בלבד.
מימוש: בטבלת Copies נוסיף עמודה book_id שתהיה FOREIGN KEY לעמודת ID בטבלה BOOKS.

**קשר N:N -** קוראים ←→ ספרים

קורא אחד יכול לשאול הרבה ספרים לאורך זמן, וכל ספר יכול להישאל ע״י קוראים רבים.
בפועל ההשאלה נעשית על עותק, ולכן מממשים את ה־N:N באמצעות טבלת גישור loans שמקשרת בין readers ל־copies ומתוך copies ל־books .

טבלה מסכמת

| סוג קשר | דוגמה | מימוש |
|---|---|---|
| 1:1 | **Readers ↔ Reader_id** | בדר"כ עם UNIQUE |
| N:1 | **BOOKS ↔ COPIES** | מוסיפים FK בטבלת העותקים ל ID בטבלת הספרים |
| N:N | **Readers ↔ Books** | נעשה טבלת קשר בה שתי העמודות הן מפתחות זרים<br>ועוד שתי טבלאות נפרדות אחת של ספרים ואחת של קוראים |

## הגדרות מפתחות ואילוצים

### Primary key

- מזהה ייחודי לכל שורה בטבלה.
- מבטיח שאין שתי שורות עם אותו מזהה.
- כל טבלה יכולה להכיל רק PK אחד, אבל ה־PK יכול להיות מורכב מכמה עמודות (Composite Key).
- שימוש: כדי שנוכל לזהות באופן חד־חד ערכי כל רשומה , לדוגמה book_id בטבלת Books

---

### FOREIGN KEY (FK)

- מגדיר קשר בין טבלאות.
- הערכים בעמודה חייבים להתאים לערכים הקיימים בעמודת ה־PK בטבלה אחרת.
- יכולה להיות בטבלה יותר מ־FK אחד.
- שימוש: לשמור על שלמות קשרים - למשל, לא תהיה השאלה (Loans) עם reader_id שלא קיים בטבלת Readers

---

### UNIQUE

- אילוץ שמבטיח שערך מסוים לא יחזור על עצמו בעמודה.
- בניגוד ל־PK, אפשר לשים כמה אילוצים UNIQUE באותה טבלה.
- שימוש: למנוע כפילויות – למשל phone בטבלת Readers :כדי שלא יהיו שני קוראים עם אותו מספר טלפון.

---

### NOT NULL

- אילוץ שמכריח להזין ערך בעמודה.
- מונע יצירת רשומה חסרה בשדות קריטיים.
- שימוש: לדוגמה, שם של קורא (full_name) חייב להיות מלא, אחרת הרשומה לא הגיונית.

---

### CHECK

- מוודא תנאים לוגיים על הערכים.

- שימוש: למשל CHECK (price > 0) כדי שלא יוזן מחיר שלילי, או

CHECK (loan_date <= return_date)

---

**DEFAULT**

דיפולט אינו אילוץ אך משמש לנוחות- נגדיר ערך הגיוני אם לא הזינו כלום.

אנחנו בעצם אומרים למערכת : אם לא הזינו את התאריך, נזין את התאריך של היום כברירת מחדל.

- שימוש: לדוגמה 'status DEFAULT 'available בעותק של ספר.

---

**SUPERKEY**

Superkey הוא כל צירוף של עמודות שמזהה שורה בטבלה בצורה **חד-משמעית**.
כלומר - לא קיימות שתי שורות בטבלה שיש להן את אותו ערך בצירוף הזה.

- כל **מפתח ראשי (Primary Key)** הוא Superkey
- אבל לא כל Superkey הוא מפתח ראשי – כי ייתכן שהוא מכיל **שדות מיותרים**.
- ייתכנו מספר Superkeys שונים לאותה טבלה.
- מפתח עם **שדה נוסף מיותר** עדיין נחשב Superkey – כל עוד הוא מזהה שורה באופן ייחודי.

**דוגמאות ל-SUPERKEY בטבלת Readers**

1. {reader_id} - המפתח הראשי ולכן מזהה ייחודי.
2. {phone} - עשוי להיות SUPERKEY אם נגדיר אילוץ UNIQUE, אך בפועל לא תמיד מתאים לדרישות (למשל כמה ילדים המשתמשים באותו מספר של ההורה).
3. {reader_id, full_name} - גם SUPERKEY, אך מיותר מאחר ש־reader_id כבר מזהה ייחודי.
4. {reader_id, phone, full_name} - עדיין SUPERKEY אך עודף עמודות ולא נדרש בפועל.

**לא כל אילוץ אפשרי מתאים לדרישות האמיתיות של המערכת. יש לבחור אילוצים בהתאם למציאות העסקית, כדי למנוע הגדרות שגויות.**

**לסיכום קצר למה צריך כל אחד:**

- PK לזהות כל שורה חד־משמעית.
- FK לקשר טבלאות ולשמור על עקביות.
- UNIQUE למנוע כפילויות בערכים מסוימים.
- NOT NULL להכריח מילוי שדות חשובים.
- CHECK לאכוף כללים לוגיים.
- DEFAULT למלא ערך אוטומטי כשלא הוזן ערך.

**<u>איך TYPE מגן על הנתונים?</u>**

1. בחירת טיפוס נכון מונעת ערכים לא רלוונטיים- למשל, אם נגדיר INT במקום FLOAT במספר ההשאלות, נבטיח שהנתון יהיה תמיד שלם.
2. שומר על עקביות - אי אפשר לשים טקסט בעמודה שהוגדרה כINT
3. מונע שגיאות עסקיות - תאריכים חייבים להיות חוקיים.
4. מבטיח דיוק - למשל שימוש ב־DECIMAL לערכים כספיים.
5. מאפשר אילוצים נוספים- CHECK, NOT NULL, UNIQUE עובדים טוב יותר כש־TYPE מתאים.

**הטיפוס עצמו הוא שכבת ההגנה הראשונה על הנתונים - הוא קובע איזה ערכים יתקבלו ואיזה יידחו מראש.**

איפה שהדטה בייס יכול להגן עלינו, עדיף שיעשה זאת.

עדיין, יש מקומות שאי אפשר להגן, למשל: שם של ספר- נהיה חייבים להגדיר את העמודה כvarchar/ TEXT .

## סוגי TYPE:

| | |
|---|---|
| `DATE` | תאריך בלבד ( `YYYY-MM-DD` ) |
| `TIMESTAMP` | תאריך ושעה |
| `BOOLEAN` | אמת / שקר |
| `INT` | מספר שלם |
| `DECIMAL(p,s)` | מספר עשרוני מדויק |
| `FLOAT` | מספר עשרוני לא מדויק |
| `VARCHAR(n)` | מחרוזת באורך משתנה |
| `TEXT` | טקסט ארוך ללא מגבלה |

## דרישות מיצוג

נרמול הוא תהליך בתכנון מסדי נתונים שבו שומרים את הנתונים בצורה מסודרת, עקבית ויעילה, על ידי חלוקה לטבלאות קטנות יותר והגדרת קשרים ביניהן. המטרה היא **למנוע כפילויות, שגיאות ועדכונים מיותרים** במסד הנתונים.

**מטרות הנרמול:**

1. **הפחתת כפילויות**
לדוגמה: במקום לרשום שוב ושוב את שם העיר של כל לקוח – יוצרים טבלת ערים נפרדת, ומקשרים כל לקוח לעיר אחת.

2. **שיפור שלמות הנתונים**
הנתונים נשמרים בצורה מדויקת, כך שלא ניתן להזין ערכים לא תקפים או סותרים.

3. **קלות בעדכון נתונים**

שינוי במקום אחד (כמו שם של עיר) משפיע על כל הטבלה דרך המפתח הזר - בלי צורך לעדכן שורות רבות.

4. **הפרדת ישויות שונות**
כל טבלה מתארת ישות אחת בלבד (למשל: לקוחות, הזמנות, מוצרים), וזה מאפשר להבין את הנתונים טוב יותר ולנהל אותם בצורה מסודרת.

חשוב לציין שנורמליזציה אמנם מבטיחה תקינות ועקביות של הנתונים, אך לרוב היא גם מסרבלת את בסיס הנתונים, משום שהיא יוצרת יותר טבלאות.
**הסרבול הזה הוא המחיר שאנחנו משלמים כדי להבטיח אמינות ושלמות במידע.**

**דרגות נרמול:**

- **NF1** אין ערכים מרובים בעמודה אחת (כמו רשימה)

נניח שבטבלת ספרים יש עמודה "מחברים" ושם רשום *"נועה, אור, מאיה"*.
זו בעיה - כי בעמודה אחת יש רשימה של ערכים, אי אפשר לחפש או לסנן לפי מחבר אחד.

ב־NF1 מפרקים – לכל מחבר יש שורה משלו בקשר אל הספר.

- **NF2** כל עמודה תלויה בכל המפתח הראשי

נניח שבטבלת ההשאלות המפתח הוא (קורא, עותק), ובטבלה מופיע גם שם הקורא.
זו בעיה – כי שם הקורא תלוי רק בקורא, לא במפתח המלא (קורא + עותק).

ב־NF2 מזיזים את שם הקורא לטבלת הקוראים, ובטבלת השאלות משאירים רק מזהי קורא ועותק.

הנרמול הוא **היררכי**, ז"א NF2 כולל גם את NF1

- **NF3** אין תלות עקיפה בין עמודות – כל עמודה צריכה לתאר את המפתח ורק אותו.

נניח שבטבלת ספרים רשמנו גם את שם המחבר וגם את האימייל שלו.
זו בעיה – כי אימייל תלוי במחבר, לא ישירות בספר. זו תלות עקיפה.

ב־NF3 מפרידים – שמים טבלת מחברים עם שם ואימייל, ובטבלת ספרים נשאר רק מזהה מחבר.

למידע נוסף:

https://en.wikipedia.org/wiki/Database_normalization

**הנחיות לעיצוב נכון:**

- כל ישות → טבלה נפרדת
- כל טבלה → מפתח ראשי מינימלי
- נמנע מכפילויות ושדות מחושבים
- קשרי N:N תמיד דורשים טבלת קשר
- כל עמודה צריכה לייצג עובדה ישירה על המפתח

ביל קנט (Bill Kent) סיכם באופן שנון את אופן ההתייחסות למפתחות:

"[every] non-key [attribute] must provide a fact about the key, the whole key, and nothing but the key".

כאשר אנחנו בונים טבלאות, אנחנו צריכים להבין מה יישות ומה מאפיין. מיישות אנחנו בונים טבלה, היא נושא מרכזי, ומאפיין נכניס בתוך טבלה של יישות.

נניח טבלת **ספרים**:

- **מפתח ראשי :** book_id
- **תכונות מותרות :** title, publish_year, author_id

כל אחת מהתכונות האלו היא **עובדה ישירה על הספר (book_id)** .
לעומת זאת, אם נוסיף לטבלה גם את author_name - זו תהיה תלות **מעברית**, כי השם של המחבר תלוי ב־ author_id ולא ישירות ב־book_id .

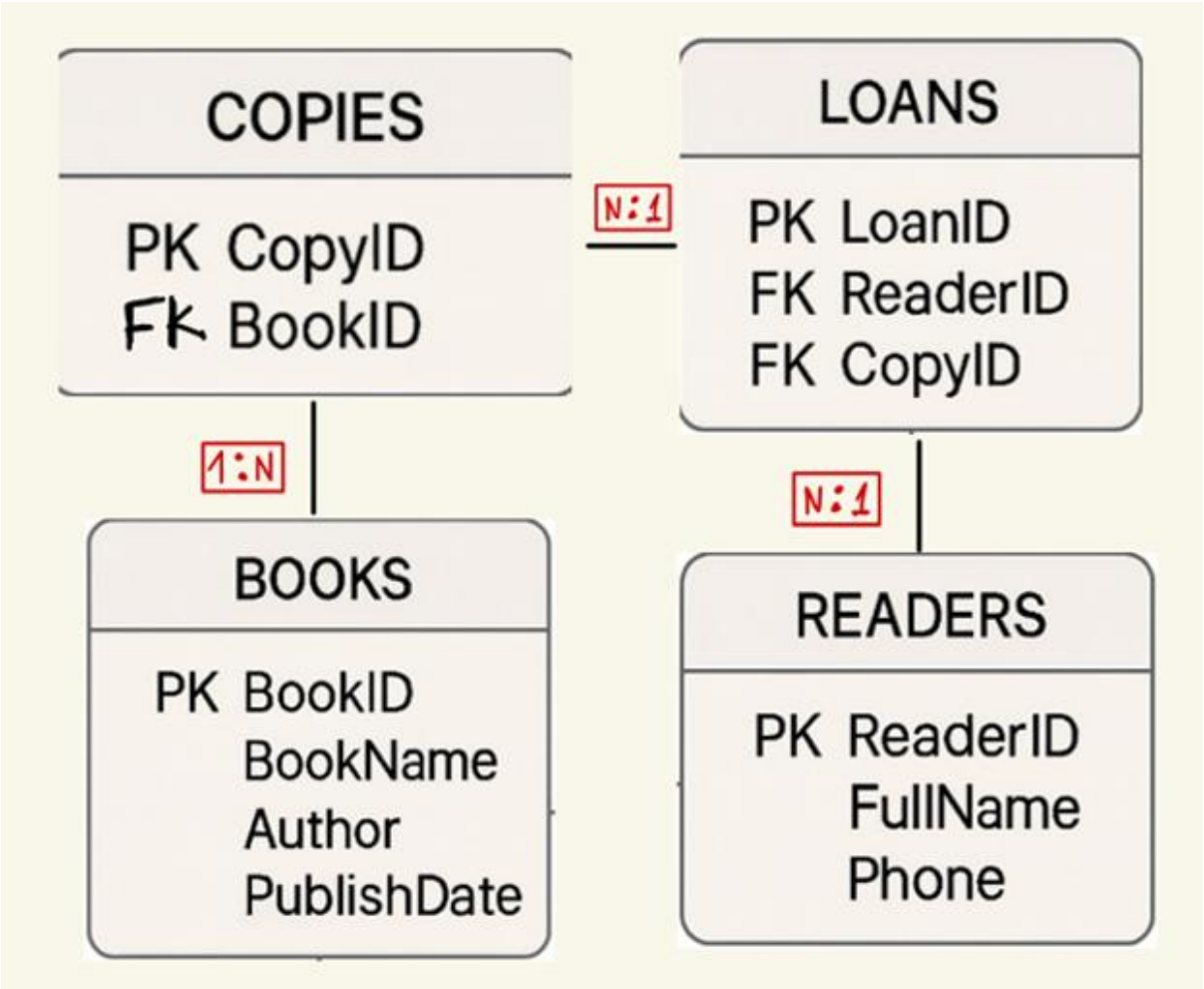
**ERD (Entity Relationship Diagram)**

(רשות – לא חובה לצייר)

תרשים ישויות־קשרים. זהו כלי גרפי שמציג את הישויות במערכת ואת הקשרים ביניהן.
היתרון שלו הוא פשטות והמחשה חזותית: קל להבין במהירות את מבנה בסיס הנתונים גם בלי להיכנס לטבלאות עצמן.

כל מלבן מייצג ישות (טבלה), השדות בפנים הם המאפיינים שלה (עמודות), והקווים בין הישויות מתארים את סוגי הקשרים (1:1, N:N:1, N).

למידע נוסף:
https://he.wikipedia.org/wiki/%D7%9E%D7%95%D7%93%D7%9C_%D7%99%D7%A9%D7%95%D7%99%D7%95%D7%AA_%D7%A7%D7%A9%D7%A8%D7%99%D7%9D


COPIES
PK CopyID
FK BookID
N:1
LOANS
PK LoanID
FK ReaderID
FK CopyID
1:N
N:1
BOOKS
PK BookID
BookName
Author
PublishDate
READERS
PK ReaderID
FullName
Phone