סיכום - DATA INTEGRITY

נתונים באיכות גבוהה קריטיים לקבלת החלטות. אלו יכולות להיות החלטות טקטיות כמו מרשם לחולה או אסטרטגיות על תחום פעילות עדיף. לרוע המזל, נראה שיש אין ספור דרכים בהם נתונים יכולים להיות מוטעים. נתונים ללא שגיאות הם גיזת זהב שאולי תמצא יום אחד.
מה שאנו יכולים לעשות הוא לזהות בעיות פוטנציאליות שנתונים שלנו ולהגן מפני חלקן.

בשל ריבוי הפנים של בעיות אפשריות אין הגדרה טובה של data integrity. כעקרון, נתונים טובים הם נתונים שפעולה על פי הם תוביל לתוצאה נכונה.
כהגדרה יותר קוקרטית data integrity היא שמירה על נכונות, שלמות, ועקביות הנתונים לאורך כל מחזור החיים שלהם - מהקליטה ועד השימוש.
המטרה - שהנתונים במערכת יהיו **נכונים, אמינים, ולא סותרים את עצמם**.

דוגמאות לחשיבות

- החלטות שגויות עלולות להתקבל על סמך נתונים לא נכונים.
  - דוגמה: אם במחסן רשום שיש 50 יחידות במלאי בפועל ויש רק 5 - המערכת תאשר מכירה של 50 לקוחות, מה שיוביל לאי־עמידה בהתחייבויות.
- תיקון טעויות בדיעבד יקר ומסובך יותר מלעצור אותן מראש.
  - דוגמה: אם חשבונית נשלחה ללקוח עם סכום שגוי ונדרשת הוצאת זיכוי ותיקון מול הנהלת חשבונות - זה הרבה יותר יקר ומסורבל מאשר לוודא מראש שהנתון מוזן נכון.
- נתונים לא עקביים עלולים לפגוע באמינות הארגון.
  - דוגמה: אם בדו"ח אחד כתוב שלחברה 200 לקוחות ובדו"ח אחר 180 - ההנהלה או משקיעים יאבדו אמון באמינות הנתונים.
- **עיקרון מרכזי**: אם לא בדקת – תתייחס לנתונים כאילו הם שגויים.

בעיות נפוצות בדאטה – ודוגמאות מהשטח

- שבירה של uniqueness **-** אותו נתון מופיע פעמיים או יותר בצורה מיותרת.

דוגמה:
נועה לוי רשומה פעם כ-"Noa Levi" ופעם כ-"Levi, Noa" עם אותו טלפון ואימייל.

customer_id	full_name	phone	email
1024	Noa Levi	054-5551234	noa@example.com
1857	Levi Noa	054-5551234	noa@example.com

המערכת רואה אותם כשני לקוחות נפרדים.
בפועל מדובר באותה נועה לוי – אבל השם הוזן בפורמט שונה.
התוצאה:
היסטוריית ההזמנות שלה מפוצלת בין שני לקוחות שונים.
שליחת דיוור כפול - חוויית משתמש גרועה.
נתוני המכירות שגויים (נראים כאילו יש שני לקוחות שונים).

- ערכים חסרים או NULL - עמודה שחייבת להכיל ערך - נשארת ריקה.

דוגמה:

הזמנה במערכת ללא כתובת למשלוח

order_id	customer_id	shipping_address
5012	1024	NULL

שדות חובה לא מולאו, למרות שהמערכת הייתה אמורה לדרוש אותם.

התוצאה:

אי אפשר לשלוח את המוצר.

נדרשת פנייה ידנית ללקוח, מה שמעכב את המשלוח.

ירידה בשביעות רצון הלקוח.

- פורמט נתונים שגוי - ערכים נשמרים בפורמט לא נכון שמונע מהמערכת להשתמש בהם.

דוגמה:

שדה price במוצר נשמר כטקסט במקום מספר.

התוצאה:

אי אפשר לחשב סיכום הזמנה אוטומטית.

מנוע החיפוש לפי מחיר לא עובד.

פגיעה במהירות ובדיוק של הדוחות.

- חוסר אחידות (Inconsistency) - אותה משמעות מוזנת בצורות שונות, מה שמקשה על עיבוד הנתונים.

דוגמה:

בתפקידים של עובדים מופיעים הערכים: "Accountant", "Acct".

employee_id	full_name	job_title
301	Dan Cohen	"Accountant"
302	Dana Levi	"Acct."

התוצאה:

חיפושים לפי תפקיד לא מחזירים את כל האנשים הרלוונטיים.

קשה להפיק דוחות נכונים.

נדרש ניקוי ידני של הנתונים.

- בעיות זיהוי (Identification Problems) - לא ניתן לדעת אם שתי רשומות מתייחסות לאותו ישות.

דוגמה:

בטבלת לקוחות: הלקוח "דני כהן" עלול להופיע פעמיים תחת מזהים שונים, בגלל הזנה שונה של השם שלו - ולא ברור אם מדובר באותה בן אדם או בשתיים שונות.

id	name
601	Danny Cohen
755	Danny Moshe Cohen

התוצאה:

בעיה בהצלבת נתונים בין סניפים.
סיכון לשגיאות בחשבוניות ותשלומים.
דוחות כספיים לא מדויקים.

כלים ופתרונות לשמירה על Data Integrity

- הגדרת מגבלות (**Constraints**) -
  - **מה זה:** כללים בבסיס הנתונים שמוודאים שמוזנים רק ערכים תקינים.
  - **סוגים נפוצים:**
    NOT NULL - מונע ערכים ריקים.
    CHECK - בודק טווח או תנאי. נדרש לשים לב שהגדרת תנאי לא גורמת לחלק מהמידע להיפגע - לדוגמא, אם הגדרנו שדה מסוג VARCHAR המייצג שם משפחה של עובדים, יכול להכיל עד 25 תווים, שמות עובדים ארוכים יכולים להיחתך ולהכיל מידע חלקי.
    FOREIGN KEY - מוודא שערך קיים בטבלה אחרת, כלומר הערך שמוזן חייב להיות קיים בטבלה הרלוונטית.
  - **דוגמה:**
    עמודת department_id בטבלת עובדים מוגדרת כ־FOREIGN KEY לטבלת מחלקות, כך שלא ניתן להזין מחלקה שלא קיימת.

- מניעת כפילויות (**Avoid Duplicates**) -
  - **מה זה:** מנגנון המבטיח שרשומה זהה לא תוזן פעמיים.
  - **מה הבעיה בכפילויות?**
    כפילות מתרחשת כאשר אותה ישות (למשל עובד, לקוח או מוצר) נרשמת פעמיים או יותר במערכת. זה גורם לבלבול, בזבוז מקום, ושיבושים בדוחות - למשל, אותו לקוח מופיע פעמיים עם מזהים שונים ולכן ההכנסות ממנו נספרות פעמיים.
  - **כיצד זה מתבטא בנתונים (דוגמא כללית)**

id	name	email
1	Yahel Cohen	yahel@uni.com
2	Yahel Cohen	yahel@uni.com

  - **השלכות:** איבוד אמון במידע, קשיים בדוחות, טעויות עסקיות או תפעוליות.
  - **פעולות למניעת בעיה זו:**
    שימוש ב- **Primary Key.**
    שימוש ב- **Unique Constraint.**
  - **דוגמה:**
    להגדיר את ID כ **Primary Key** על מנת להבטיח ייחודיות של השורה עצמה ולהגדיר על העמודה email אילוץ **UNIQUE** כך שלא ניתן יהיה לרשום את אותה כתובת אימייל פעמיים ולמנוע כפילויות.
    עמודת name כן יכול להופיע פעמיים כי שני אנשים יכולים לשאת את אותו השם.

- בחירת סוג נתונים נכון (**Choose Appropriate Data Type**) -
  - **מה זה:** בחירה בסוג הנתון (TYPE) שמתאים לערך.
  - **פעולות:**
    לבחור את סוג הטיפוס המינימלי שמתאים לנו. כך אוטומטית ערכים מחוץ לו לא יוזנו.
    לבחור אורך מחרוזת מינימלי שעדיין מתאים לנתון (Constraint)
    להגדיר ערכי ברירת מחדל אם נדרש.
  - **דוגמה:**
    עבור גיל - נשתמש בINT עם Constraint שמוודא שנכנס ערך בין 0-120.

Information Schema (מאגר המידע על בסיס הנתונים)

- מה זה בעצם?

Information Schema הוא אוסף של טבלאות ותצוגות (Views) פנימיות שקיימות בכל מסד נתונים סטנדרטי (MySQL, PostgreSQL, SQL Server ועוד).
הוא לא מכיל את הנתונים העסקיים עצמם, אלא מידע (Metadata) על מבנה מסד הנתונים:

  - אילו טבלאות קיימות.
  - אילו עמודות יש בכל טבלה.
  - אילו מגבלות (Constraints) ומפתחות מוגדרים.
  - אילו אינדקסים קיימים.
  - אילו קשרים בין טבלאות (Foreign Keys) מוגדרים.

- למה זה חשוב ל-Data Integrity?
  - מאפשר לייצר בדיקות אוטומטיות על המבנה עצמו, בלי לעבור ידנית על כל טבלה.
  - עוזר לזהות מקומות שיכולים לגרום לשגיאות עתידיות (למשל, עמודות שמאפשרות NULL למרות שלא אמור להיות).
  - כלי מעולה לכתיבת שאילתות אוטומטיות שבודקות ומנטרות את המערכת באופן קבוע.
- יתרונות בשימוש ב - Information_schema -
  - **סטנדרטי** – קיים כמעט בכל סוגי מסדי הנתונים.
  - **קריא ואחיד** – אין צורך לדעת את המימוש הפנימי של כל DB.
  - **מאפשר אוטומציה** – אפשר לבנות סקריפטים שמבצעים בדיקות על בסיס הנתונים בצורה מחזורית.
  - **מונע טעויות מבניות** – קל לגלות בעיות לפני שהן פוגעות בנתונים.

- **דוגמאות -**

```
select *
from
INFORMATION_SCHEMA.Tables;
```

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE	VERSION	ROW_FORMAT	TABLE_ROWS	AVG_ROW_LE
def	sakila	sales_by_store	VIEW	NULL	NULL	NULL	NULL	NULL
def	sakila	sales_by_film_category	VIEW	NULL	NULL	NULL	NULL	NULL
def	sakila	actor_info	VIEW	NULL	NULL	NULL	NULL	NULL
def	world	city	BASE TABLE	InnoDB	10	Dynamic	4046	101
def	world	country	BASE TABLE	InnoDB	10	Dynamic	239	479
def	world	countrylanguage	BASE TABLE	InnoDB	10	Dynamic	984	99
def	imdb_ijs	actors	BASE TABLE	InnoDB	10	Dynamic	816825	44
def	imdb_ijs	directors	BASE TABLE	InnoDB	10	Dynamic	87304	42
def	imdb_ijs	directors_genres	BASE TABLE	InnoDB	10	Dynamic	156547	43
def	imdb_ijs	movies	BASE TABLE	InnoDB	10	Dynamic	389451	52
def	imdb_ijs	movies_directors	BASE TABLE	InnoDB	10	Dynamic	371304	29
def	imdb_ijs	movies_genres	BASE TABLE	InnoDB	10	Dynamic	393900	33
def	imdb_ijs	roles	BASE TABLE	InnoDB	10	Dynamic	3370833	39
def	imdb_ijs	directors_tmp	BASE TABLE	InnoDB	10	Dynamic	85372	55
def	imdb_ijs	tmp_movies	BASE TABLE	InnoDB	10	Dynamic	2	8192
def	imdb_ijs	tmp_roles	BASE TABLE	InnoDB	10	Dynamic	2	8192

```
# Generate queries to check each column for being unique
Select
concat('Select count(*) as cnt, count(distinct '
, Column_Name
, ') as dis_values from imdb_ijs.', TABLE_NAME, ' having cnt != dis_values;') as num_query
from
INFORMATION_SCHEMA.Columns
where
table_schema = 'imdb_ijs'
order by table_name;
```

num_query
Select count(*)as cnt, count(distinct id) as dis_...
Select count(*)as cnt, count(distinct first_name...
Select count(*)as cnt, count(distinct last_name)...
Select count(*)as cnt, count(distinct gender) as...
Select count(*)as cnt, count(distinct movie_id) ...
Select count(*)as cnt, count(distinct number_of...
Select count(*)as cnt, count(distinct movie_id) ...
Select count(*)as cnt, count(distinct number_of...
Select count(*)as cnt, count(distinct movie_id) ...
Select count(*)as cnt, count(distinct number_of...
Select count(*)as cnt, count(distinct prob) as di...
Select count(*)as cnt, count(distinct id) as dis_...
Select count(*)as cnt, count(distinct number_of...
Select count(*)as cnt, count(distinct movie_id) ...
Select count(*)as cnt, count(distinct ifnull(numb...
Select count(*)as cnt, count(distinct movie_id) ...
Select count(*)as cnt, count(distinct number_of...
Select count(*)as cnt, count(distinct id) as dis_...
Select count(*)as cnt, count(distinct name) as d...
Select count(*)as cnt, count(distinct importance...

```
# Find, using the information_schema, all string columns and their number of records
#of length as the column length.

SELECT

table_name,

column_name,

character_maximum_length,

CONCAT('SELECT COUNT(*) AS count_full_length FROM imdb_ijs.',

table_name,' #The name of the Table',

 ' WHERE CHAR_LENGTH(', column_name, ') = ',

character_maximum_length,';',' #The name of the Colomn') AS query

FROM information_schema.columns

WHERE data_type IN ('varchar', 'char', 'text')

AND character_maximum_length IS NOT NULL

AND table_schema = 'imdb_ijs';
```

	TABLE_NAME	COLUMN_NAME	CHARACTER_MAXIMUM_	query
▶	actors	first_name	100	SELECT COUNT(*) AS count_full_length FROM imdb_ijs.actors #The name of the Table WHERE CHAR_LENGTH(first_nan
	actors	last_name	100	SELECT COUNT(*) AS count_full_length FROM imdb_ijs.actors #The name of the Table WHERE CHAR_LENGTH(last_nam
	actors	gender	1	SELECT COUNT(*) AS count_full_length FROM imdb_ijs.actors #The name of the Table WHERE CHAR_LENGTH(gender)
	categories	name	100	SELECT COUNT(*) AS count_full_length FROM imdb_ijs.categories #The name of the Table WHERE CHAR_LENGTH(nam
	cities	name	100	SELECT COUNT(*) AS count_full_length FROM imdb_ijs.cities #The name of the Table WHERE CHAR_LENGTH(name) = 1
	close_years	reason	4	SELECT COUNT(*) AS count_full_length FROM imdb_ijs.close_years #The name of the Table WHERE CHAR_LENGTH(rea

<u>השורה רשומה הראשונה שמוכלת בעמודת QUERY -</u>

'SELECT COUNT(*) AS count_full_length FROM imdb_ijs.actors #The name of the Table WHERE CHAR_LENGTH(first_name) = 100; #The name of the Colomn'

הרחבה - נרמול (Normalization) - שמירה על סדר בנתונים

- מה זה נרמול?

נרמול הוא תהליך **עיצוב מבנה מסד נתונים** כך שהוא:

  - **מצמצם כפילויות**
  - **מונע חוסר עקביות**
  - **שומר על שלמות הנתונים (Integrity).**
  - מאפשר תחזוקה והרחבה קלים יותר.

- למה נרמול חשוב ל־Data Integrity?

מונע מצב בו אותו מידע נשמר במקומות שונים עם ערכים שונים (Inconsistency):

  - **דוגמה:** שם המחלקה נשמר גם בטבלת עובדים וגם בטבלת פרויקטים, ובמקום אחד כתוב "מחלקת שיווק" ובשני "שיווק" - בלבול בדוחות.

מקטין את הסיכוי לשגיאות מבניות שעלולות לשבור קשרים לוגיים בין נתונים.

  - **דוגמה:** בטבלה לא מנורמלת שמכילה גם פרטי לקוח וגם פרטי הזמנה - מחיקת ההזמנה תמחק גם את פרטי הלקוח ותשבור קשרים לוגיים בין הטבלאות.

מונע **Anomalies**:

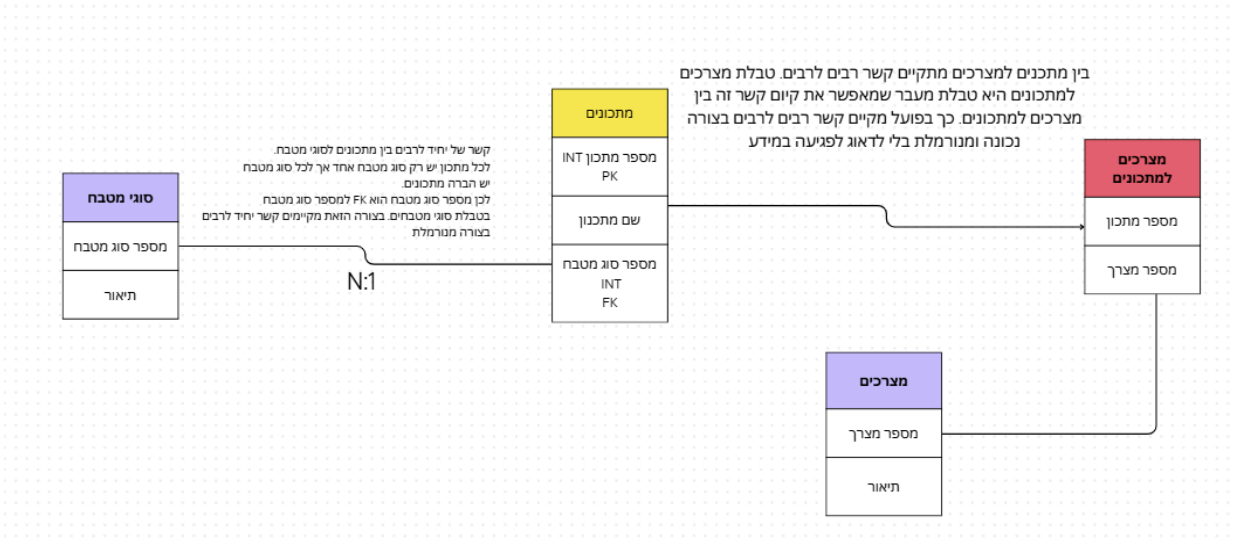
  - **Insertion anomaly** - לא ניתן להוסיף נתון כי חסר מידע שקשור אליו.
  - **Update anomaly** - צריך לעדכן נתון במקומות רבים - סיכון לטעות.
  - **Deletion anomaly** - מחיקת נתון גוררת אובדן מידע שלא רצינו למחוק.

- קשרים בין ישויות (Relationships)

  - **1:1 (אחד לאחד)** - רשומה אחת מקושרת לרשומה אחת בלבד בטבלה אחרת.
  דוגמה - עובד וכרטיס עובד מגנטי. לכל עובד יש כרטיס אחד וכל כרטיס משויך רק לעובד אחד.
  איך מיוצג ב - DB - ניתן לממש בטבלה אחת בשתי עמודות שונות.

  - **N:1 (יחיד לרבים) -** רשומה אחת מקושרת לרבות בטבלה אחרת.
  דוגמה - מחלקה ועובדים. בכל מחלקה יש הרבה עובדים אבל כל עובד שייך למחלקה אחת
  איך מיוצג ב - DB - שתי טבלאות נפרדות, כאשר הטבלה של ה־"Many" כוללת מפתח זר (FK) שמצביע על המפתח הראשי של ה־"One".

  - **M:N (רבים לרבים) -** קשר הדדי בין רשומות רבות משני הצדדים.
  דוגמה: סטודנטים וקורסים. לכל סטודנט יש כמה קורסים ובכל קורס יש כמה סטודנטים.
  איך מיוצג ב - DB - ממומש באמצעות טבלת קשר שמכילה שני מפתחות זרים - אחד לכל טבלה.
  טבלה זו מאפשר קיום קשר רבים לרבים בין שתי ישויות.



- איך נרמול מתקשר ל־Data Integrity?
  - כשאנחנו שומרים נתונים בבסיס נתונים, חשוב לוודא שהם **מדויקים, עקביים ושמורים בצורה מסודרת**. נרמול הוא תהליך של ארגון הנתונים כך שכל מידע נשמר **רק במקום אחד מתאים**, ויש קשרים נכונים בין הטבלאות. זה מגן על הנתונים מפני כפילויות, חוסר התאמה ואובדן מידע.
  - **סוגי השלמות (Integrity) שנרמול שומר -**
    - שלמות ישות (Entity Integrity)
      כל רשומה מקבלת מזהה ייחודי (Primary Key) שלא חוזר על עצמו, כך שאי אפשר לבלבל בין שתי רשומות שונות.
    - שלמות התייחסות (Referential Integrity)
      קשרים בין טבלאות נשמרים באמצעות מפתחות זרים (Foreign Keys), כך שלא תישאר רשומה שמצביעה על משהו שלא קיים.
  - **נרמול נכון מונע -**
    - כפילויות (Duplicates) – שמירת אותו מידע פעמיים במקומות שונים.
    - חוסר עקביות (Inconsistencies) – מצבים שבהם אותו מידע מופיע עם ערכים שונים.
    - אובדן נתונים עקב מחיקות לא זהירות – למשל, מחיקת הזמנה לא תגרום למחיקת פרטי הלקוח.

- צורות הנרמול (Normal Forms) -
  צורות נרמול הן **שלבים הדרגתיים** בעיצוב בסיס הנתונים, שכל אחד מהם מוסיף כללי סדר נוספים.
  המטרה: לצמצם כפילויות, למנוע חוסר עקביות, ולהבטיח שלמות נתונים.
  - **1NF - First Normal Form**
    הכללים:

    ★ כל עמודה בטבלה מכילה ערך יחיד (Atomic Value) בלבד.
    ★ אין רשימות/קבוצות של ערכים באותה עמודה.
    ★ כל רשומה ייחודית (אין שורות זהות).

    למה זה חשוב ל־Data Integrity?

    ★ מונע בלבול וחוסר עקביות בערכים מרובי פריטים.
    ★ מאפשר שליפה, עדכון וחיפוש פשוטים יותר.

    דוגמא:

לא תקין

id	phone
1	03-9876543, 050-1234567

תקין

id	phone
1	050-1234567
1	03-9876543

- **2NF – Second Normal Form**

<u>הכללים:</u>

★ עמידה ב־1NF.
★ כל עמודה שאינה מפתח ראשי **תלויה בכל המפתח הראשי** ולא רק בחלק ממנו (חשוב במיוחד כשיש מפתח מורכב).

<u>למה זה חשוב ל־Data Integrity?</u>

★ מונע כפילויות ועומס מיותר של נתונים שאינם קשורים ישירות לרשומה.
★ מפחית את הסיכון לעדכן נתונים במקומות רבים.

<u>דוגמא:</u>
נניח שקיימת טבלה עם המבנה הבא - (order_id, product_id, product_name).
המפתחות הראשיים לטבלה הם השדות - order_id, product_id.
העמודה product_name תלויה **רק** ב־product_id ולא ב־order_id מכיוון ששם המוצר נקבע לפי מזהה המוצר, והוא לא תלוי באיזו הזמנה מדובר, **כלומר היא תלויה <u>רק</u> בחלק מהמפתח הראשי.**
לכן המשמעות היא שאם מוצר מופיע בעשר הזמנות שונות, שם המוצר יופיע עשר פעמים בטבלה - כפילות מיותרת.
הפתרון הוא נרמול ל2NF - ליצור טבלת מוצרים נפרדת עם המבנה - product_id, product_name ולהשאיר בטבלה הראשונה רק את order_id, product_id.
product_id יהיה FK לטבלה החדשה ושם המוצר יישלף על ידי JOIN בין הטבלאות.

- **3NF – Third Normal Form**

<u>הכללים:</u>

★ עמידה ב־2NF.
★ עמודות שאינן מפתח לא תלויות זו בזו.

<u>למה זה חשוב ל־Data Integrity?</u>

★ מונע מצב בו ערך אחד משפיע על ערך אחר שלא דרך המפתח הראשי.
★ מונע חוסר עקביות כאשר מידע משותף נמצא בכמה מקומות.

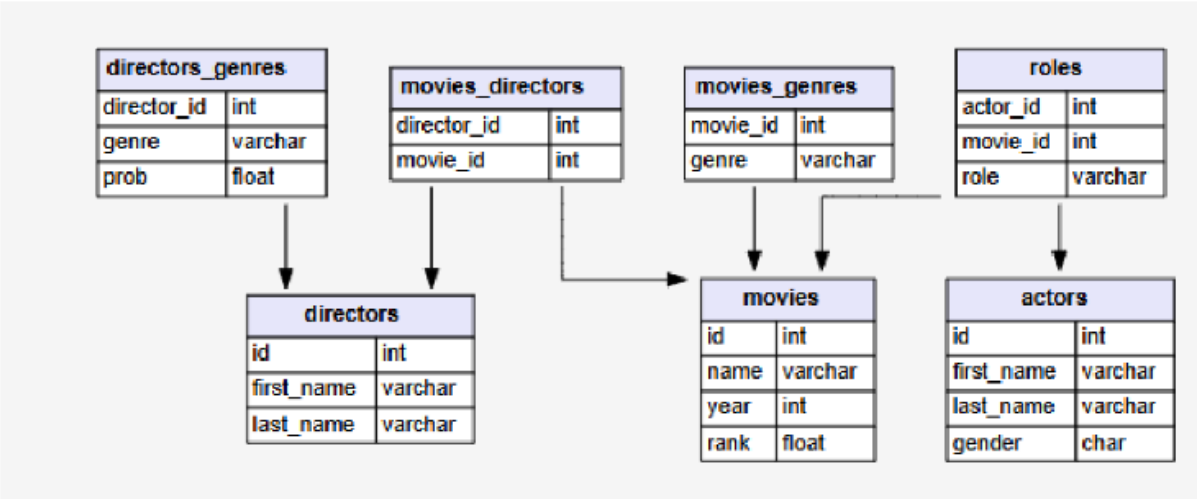
<u>דוגמא:</u>
נניח שקיימת טבלה עם המבנה הבא - (employee_id, department_id, department_name)
העמודה department_name (שם המחלקה) **תלויה אך ורק** ב־department_id - כלומר, אם יודעים את מזהה המחלקה, אפשר לדעת את שם המחלקה בלי קשר לעובד הספציפי (employee_id), **לכן קיימת פה תלות של עמודה שאינה המפתח הראשי שתלויה בעמודה אחרת שהיא גם לא המפתח הראשי.**
דבר זה בעייתי מכיוון שיש פה חוסר עקביות - כלומר אם נרצה לשנות את שם המחלקה, נצטרך לעדכן אותו בכל שורה של כל עובד באותה מחלקה, ויש סיכון שנשאיר נתונים לא אחידים. בנוסף יהיה לנו הרבה מאוד כפילויות של שם המחלקה ונבזבז הרבה מקום לאחסון של אותו נתון, במקום להציג אותו פעם אחת בטבלה נפרדת.
הפתרון יחסית דומה לפתרון של 2NF - נוציא את department_name לטבלת departments נפרדת, ונגדיר את department_id כFK לטבלה זו.
כך שם המחלקה נשמר פעם אחת בלבד, וכל עובד מקושר למחלקה דרך department_id.

- <u>סיכום צורות נרמול -</u>

ההבדל בין 1NF, 2NF ו-3NF נובע מרמת הפירוק של הנתונים ומסוג התלות שמותרת בין השדות. ב-**1NF** אנו דואגים שכל תא בטבלה יכיל ערך בודד בלבד (ערך אטומי), בלי קבוצות ערכים ובלי שדות מרובי נתונים, כך שכל שורה ושדה יהיו מובנים וברורים. ב-**2NF** אנו מתקדמים צעד נוסף: מעבר לעמידה ב-1NF, כל עמודה שאינה מפתח חייבת להיות תלויה בכל המפתח הראשי, ולא רק בחלק ממנו (מה שמונע תלות חלקית). ב-**3NF** אנחנו מחמירים עוד יותר: מעבר לעמידה ב־2NF, אנו מוודאים שאין תלות עקיפה (תלות מעברית) - כלומר עמודה שאינה מפתח לא יכולה להיות תלויה בעמודה אחרת שגם היא אינה מפתח, אלא רק במפתח הראשי עצמו. כך, כל שלב מוסיף שכבת ניקיון והפרדה לוגית שמקטינה כפילויות ומונעת חוסר עקביות.

שאלה ממבחן ופתרון

1. נענה על שאלות המתייחסות ל - DB הנ"ל.



a. <u>האם יתכנו genre שאין להם סרטים? מה היתרונות והחסרונות בייצוג הזה.</u>

אם מסתכלים על טבלת movies_genres בלבד הרי לכל רשומה בה מופיע ז'אנר מופיע גם סרט.
אין טבלה מרכזית לז'אנרים שמגדירה את רשימת כל הז'אנרים האפשריים, בלי קשר להופעתם בסרטים.

יתרונות:

גמישות: ניתן להוסיף ז'אנר חדש באופן מיידי על ידי הוספת שורה עם ערך טקסטואלי מתאים, ללא צורך בעדכון טבלה מרכזית.
אין צורך לנהל טבלת ישויות נפרדת לז'אנרים.

חסרונות:

כפילויות: אין בקרה על שמות הז'אנרים, וייתכנו טעויות כתיב או וריאציות כמו (DRAMA,darama).
קושי בשליפה: קשה לוודא אילו ז'אנרים קיימים במערכת אם הם לא מופיעים בטבלאות הסרטים או הבמאים.

כדאי לשים לב שגם במבנה זה - אם אין אילוץ not null או foreign key על השדה movie_id בטבלת movies_genres ניתן לשייך ל ז'אנר ערך שאינו מייצג סרט - טבלת directors_genres גם כוללת רשימת ז'אנרים ללא כל אילוץ ויכולים להופיע בה ערכים שלא מופיעים בטבלת movies_genres

b. <u>שדה genre בטבלת movies_genres הוא מסוג varchar. האם כדאי להחליפו בשדה המפוענח כ- foreign key מטבלה חדשה של ז'אנרים?</u>

כן, כדאי.

חשיבות השדה - הז'אנר הוא שדה מרכזי שמאפשר סינון, חיפוש וניתוח. חשוב לשמור אותו תקני ואחיד.

סט ערכים אפשרי - מספר הז'אנרים מוגבל וידוע מראש, לכן מתאים לטבלה נפרדת.

משמעות הטעויות ואופן הטיפול בה - כיום אפשר להקליד ז'אנר עם טעות בלי שגיאה. אם זה מפתח זר, מסד הנתונים לא יאפשר את זה. שימוש בטבלת ז'אנרים יוודא שאם הערך בה לא שגוי, כך גם ההתייחסויות אליו.

האפשרויות האחרות למניעת שגיאות אפשר לבדוק שגיאות בתוכנה, אבל זה פחות בטוח ממפתח זר. מפתח זר מבטיח שמכניסים רק ערכים תקפים.

c. נחליף את יצוג ה - genre לטבלה. כיתבו שאילתה היוצרת טבלת genre.

```
CREATE TABLE geners as
 SELECT ROW_NUMBER() OVER (ORDER BY name) AS genre_id,
name
FROM (
SELECT DISTINCT
genre AS name
FROM movies_genres
UNION
SELECT DISTINCT
genre AS name
FROM directors_genres ) AS genres_clean;
```

d. יצרו טבלת new_ movies_genres בה ה genre מיוצג בעזרת טבלת ה - genres.

```
CREATE TABLE new_movies_geners as
SELECT mg.movie_id,
gc.genre_id
FROM
movies_genres mg
JOIN
geners AS gc
ON
mg.genre = gc.name;
```

e. איזה סוג יחס יש בין director ו movies (אחד לאחד, אחד לרבים, רבים לרבים)

אמנם, לרוב מדובר בקשר של אחד לרבים (במאי אחד ביים מספר סרטים). למרות זאת לעיתים יש במאים משותפים לסרט, מדובר בקשר רבים לרבים ומספיק מקרה יחיד לשנות את סוג היחס. ניתן להסיק זאת מהסכמה משום בקשר של אחד לרבים היה עדיף ליצג בשדה במאי בטבלת סרטים כFK לטבלת במאים (כפי שמתואר בהסברים). ב - DB הזה, יחס הרבים לרבים מיוצג בטבלת קשר movies_directors ולכן מדובר על יחס רבים לרבים. חשוב לשים לב למבנה של הטבלאות שנותנות אינדיקציה ליחסים, כפי שמתואר בהסברים, טבלאות קשר קיימות כדי לאפשר יחס רבים לרבים

f. איזה סוג יחס יש בין genre ו movie ?במה תלויה תשובתכם?

זהו קשר רבים לרבים: סרט יכול להשתייך ליותר מז'אנר אחד, וכל ז'אנר כולל סרטים רבים. הדרך התקנית לייצוג קשר זה היא באמצעות טבלת קשר movies_genres.

g. מה הבעיה בטבלת directors_genres. מה התועלת בשימוש בה?

כפילות מידע:

הקשר בין במאי לז'אנר כבר נובע דרך movies_directors ו־movies_genres. אין פה מידע חדש אלא חישוב על בסיס מידע קיים. נוצרת כפילות שעלולה להוביל לסתירות.

הנתונים לא מתעדכנים.

הטבלה תופסת מקום נוסף (אבל חוסכת זמן ריצה).

genre הוא VARCHAR ולא foreign key, חשוף לטעויות כתיב, אין בקרה על הערכים.