

שימו לב לשמספר שאלות תתכנה מספר תשובות נכונות.
הפיתרון מציג תשובות נכונות מסוימות והדבר לא אומר שכל תשובה אחרת שגויה.

שאלה 1

A.

```
Select name  
From  
movies  
where  
id = 7;
```

B

```
Select max(m.rank)  
From imdb_ijs.movies as m  
;
```

C.

```
SELECT  
distinct r1.role  
FROM  
imdb_ijs.roles as r1  
LEFT JOIN(  
select  
DISTINCT r.role  
FROM  
imdb_ijs.roles AS r  
JOIN  
imdb_ijs.movies_genres AS mg  
ON  
r.movie_id = mg.movie_id  
WHERE  
mg.genre = 'Comedy') AS comedy_roles  
ON  
r1.role = comedy_roles.role  
WHERE  
comedy_roles.role IS NULL  
;
```

D.

```
select r.actor_id  
, count(distinct mg.genre) as genres  
from
```

```
imdb_ijs.roles as r
join
imdb_ijs.movies_genres as mg
on
r.movie_id = mg.movie_id
group by r.actor_id
;
```

E.

```
select actors
, count(*) as movies
from (
select
movie_id
, count(distinct actor_id) as actors
from
imdb_ijs.roles
group by
movie_id ) as inSql
group by actors
order by actors
;
```

F.

```
select actors
, count(*) as movies
from (
select
m.id
, count(distinct actor_id) as actors
from
imdb_ijs.movies as m
left join
imdb_ijs.roles as r
on
m.id = r.movie_id
group by
m.id) as inSql
group by actors
order by actors
;
```

```

G.
select
fmd.director_id
, smd.director_id
, count(fm.id) as movies
, group_concat(fm.name) as movies_names # optional
from
imdb_ijs.movies_directors as fmd
join
imdb_ijs.movies as fm
on
fmd.movie_id = fm.id
join
imdb_ijs.movies as sm
on
fm.name = sm.name
join
imdb_ijs.movies_directors as smd
on
sm.id = smd.movie_id
# Optional
where
fmd.director_id != smd.director_id # avoid reflexivity
and
fmd.director_id > smd.director_id # avoid symmetry
group by
fmd.director_id
, smd.director_id
;

```

```

H.
select max(roles)
from (
select
movie_id
, actor_id
, count(*) as roles
from
imdb_ijs.roles
group by
movie_id
, actor_id
) as inSql
;

```

```

l.
select
d.first_name
, d.last_name
, count(distinct md.movie_id) as directed_movies
, count(*) as roles
, count(distinct r.role) as different_roles
from
imdb_ijs.directors as d
join
imdb_ijs.movies_directors as md
on
d.id = md.director_id
join
imdb_ijs.roles as r
on
md.movie_id = r.movie_id
join
imdb_ijs.actors as a
on
r.actor_id = a.id
and # Note - uncommon condition not on the previous table but the first one
d.first_name = a.first_name
and d.last_name = a.last_name
group by
d.first_name
, d.last_name
order by
count(*) desc
;

```

שאלה 2

1. האם ייתכנו Genre שאין להם סרטים ?

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 (למשל: "Drama" מול "drama").
- 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

2. השדה Genre הוא מסוג Varchar. האם כדאי לשנות את זה?

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 ("Actionn") בלי שגיאה. אם זה foreign key, מוד 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.

ג. כיצד נייצר את טבלת הז'אנרים החדשה?

1. יצירת מזהה רץ לכל ז'אנר:

```
CREATE TABLE genres
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;
```

2. יצירת טבלת קשר new_movies_genres שתכיל movie_id ו-genre_id, לשם ניהול קשרים מסוג רבים-לרבים.

```
CREATE TABLE new_movies_genres
as
SELECT mg.movie_id, gc.genre_id
FROM
  movies_genres mg
```

JOIN genres AS gc

ON mg.genre = gc.name;

ד. מה סוג הקשר בין Movies ל-Director ?

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. יחס הרבים לרבים מיוצג בטבלת קשר movies_directors.





ה. מה סוג הקשר בין Movie ל-Genre ?

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

ו. מה הבעיה בטבלת directors_genres ?

- 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. (זו טעות נגררת, אי אפשר היה לתקנה פה).

שאלה 3

-  CREATE TABLE DOCTORS (
DOC_ID INT,
DOC_Name varchar(100),
SPECIALITY varchar(100),
PRIMARY KEY (doctor_id));
-  CREATE TABLE PATIENTS(
PATIENTS_ID INT,
PATIENT_NAME varchar(100),
PRIMARY KEY (PATIENTS_ID));
-  CREATE TABLE TREATMENTS(
TREATMENT_ID INT,
DOC_ID INT NOT NULL,
Treatment_type VARCHAR (100) NOT NULL,
PATIENT_ID INT NOT NULL,
PRIMARY KEY (TREATMENT_ID),
FOREIGN KEY (PATIENT_ID) REFERENCES PATIENTS(PATIENTS_ID),
FOREIGN KEY (DOC_ID) REFERENCES DOCTORS(DOC_ID));
-  CREATE TABLE TESTS (
id INT PRIMARY KEY,
test_type VARCHAR(100) NOT NULL,
doctor_id INT NOT NULL,
patient_id INT NOT NULL,
result VARCHAR(255),
test_date DATETIME,
FOREIGN KEY (doctor_id) REFERENCES doctors(id),
FOREIGN KEY (patient_id) REFERENCES patients(id)
);

2. היחסים בין הישויות:

1. בין n:1 Treatment to Doctor
2. בין n:1 Treatment to Patient
3. בין n:1 Test to Patient
4. בין n:n Patient to Doctor

3. המבנה נבחר כדי לשמור על 3NF, כל ישות (רופא, מטופל, טיפול ובדיקה) מופרדת כך שלא נוצרים כפילויות מידע. קשר הרופאים-מטופלים (Many to Many) מגושר בטבלת הטיפולים, שמאפשר לשמור סוג טיפול ותאריך מבלי להכפיל נתונים (לדוגמה עם רופא נפגש עם מטופל יותר מפעם אחת, קל לדעת שמדובר בשני פגישות שונות, וגם ניתן לדעת איזו התרחשה קודם). ניתן להשתמש במבנה בקלות כדי לענות על שאלות בנוגע למטופלים, כמות הפגישות, הרופאים איתם נפגשו וכו'.

כדי להגן על הנתונים יהיה שימוש ב foreign keys מהטבלאות המייצגות יחסי n:n לישויות המשתתפות בהן (לדוגמה רופא בטיפול חייב להופיע בטבלת הרופאים).

התמחויות לא משחקות תפקיד חשוב בתיאור המערכת ולכן לא הקצתי לה טבלה משלה והקטנתי בכך את מורכבות המערכת.

שדה תוצאה ושדה טיפול מיוצגים כ varchar. במודע, יש כאן ויתור על הגנה על נתונים חשובים. זאת מכיוון שככל הנראה יהיה צורך ליצג תוצאות מסוגים שונים שלא מפורטים בשאלה וכנראה ידרשו מורכבות רבה.

4. לייצוג היחס "רופא-מטופל-תחום" בחרתי בטבלת הטיפולים על מנת לגשר על היחס. הטבלה מכילה את מזהי הרופא והמטופל לצד פרטים נוספים על הפגישה. דרך זו שומרת על נרמול מלא, פרטי הרופא והמטופל נשמרים בטבלאות המקוריות שלהם ללא כפילויות, ואילו תחום הטיפול (ההתמחות של הרופא) נשמרת ברשומה נפרדת שאינה מכילה נתונים מיותרים. יש לשים לב כי הבחירה שלרופא יש תחום טיפול יחיד היא הנחה מצמצמת, שקיבלתי לאור משך ההתמחות הגבוה של רופאים.

היתרונות – רופא אחד יכול להיפגש עם מטופלים רבים, ולכן יש צורך לאפשר את הקשר בצורה הזו. והפוך ללא צורך בשינוי הנתונים. השאילתות פשוטות לבנייה, ניתן לשאול בקלות איזה רופאים פגשו איזה מטופל, מתי הייתה הפגישה האחרונה שלו במרפאה או כמה טיפולים הוא ביצע השנה.

החסרונות – יש צורך בביצוע JOIN בכל שאילתה שמבקשת לדעת את פרטי המטופל ו/או הרופא. כרגע כל רופא יכול להחזיק במומחיות אחת, ייתכן שאם רופא יחזיק ביותר ממומחיות אחת נצטרך להוסיף טבלה נפרדת וייעודית.

5. לייצוג בדיקות ותוצאות בחרתי בטבלה ייעודית, שבה כל רשומה היא הזמנה יחידה של בדיקה, היא כוללת מזהה בדיקה ומפתחות זרים לרופא ומטופל. בנוסף, הטבלה כוללת עמודה של תוצאת הבדיקה

שמוגדרת כ-Null עד לקליטת התשובה. מבנה זה ממלא כמה מטרות חשובות. 1 – שמירה על נרמול והבנה קלה.

2 – ניתן בקלות לפלטר על בדיקות עם תוצאה לעומת בדיקות ללא תוצאה

3- ניתן לעדכן את הטבלה באופן פשוט (כאשר מגיעה תוצאה חדשה לבדיקה).

חסרונות

1- כל בדיקה יכולה לכלול תוצאה אחת, אמנם אם בעתיד נרצה לשמור כמה ערכים לכל בדיקה (למשל ספירת-דם מלאה) נצטרך להשתמש בטבלה נפרדת.

6. שאילתא להוציא את כל הבדיקות ללא תוצאה

```
Select * from TESTS
where TESTS.Result IS NULL
ORDER BY TESTS.test_date;
```

7. בונוס:

```
SELECT t1.*
FROM Tests AS t1
LEFT JOIN Tests AS t2
    ON t2.patient_id = t1.patient_id
    AND t2.test_type = t1.test_type
    AND t2.test_date < t1.test_date -- בדיקות קודמות
    AND t2.result < t1.result -- תוצאות טובות יותר
WHERE t1.result IS NOT NULL
    AND t2.test_id IS NULL -- לא נמצאה בדיקה קודמת טובה יותר
ORDER BY t1.patient_id, t1.test_type, t1.test_date;
```