**Entities and Relationships**

The database consists of 'Photo', 'Disk', and 'RAM' entities .

'Photo' entity encapsulates information related to each photo, including 'id', 'description', and 'disk_size_needed'.

'Disk' entity represents each disk with attributes like 'id', 'cost_per_byte', 'free_space', and 'manufacturing_company'.

'RAM' entity stores data about each RAM unit, featuring attributes such as 'id' and 'company'.

Two relationship entities, 'PhotoInDisk' and 'RAMInDisk', connect the primary entities and represent many-to-many relationships.

'PhotoInDisk' connects each photo to the disk where it's stored, and 'RAMInDisk' links each RAM unit to the disk where it's installed.

**Design Decisions**

1. The many-to-many relationships were implemented via intermediary tables ('PhotoInDisk' and 'RAMInDisk') to maintain normalization, reduce redundancy, and ensure the integrity of data. This decision allows for future flexibility if the relationships between photos, disks, and RAM need to change.

2. The 'Disk', 'RAM', and 'Photo' entities are distinct from their relationships, providing scalability. For example, it's easy to add new attributes to the 'Photo' or 'Disk' entities without disrupting the overall database structure.

**Views and Functions**

The design incorporates several views and functions to perform operations on the data. Four views - 'DiskPhotoCounts', 'TotalRAMInDisk', 'DiskWithAtLeastOnePhoto', and 'DisksWithNoExclusiveCompany' - have been created to facilitate common data access patterns .

'DiskPhotoCounts' and 'TotalRAMInDisk' views enhance performance by pre-calculating the number of photos per disk and the total RAM on each disk, respectively. This way, the system can quickly retrieve this information without needing to recalculate it with every query.

'DiskWithAtLeastOnePhoto' and 'DisksWithNoExclusiveCompany' views cater to specific business rules, giving a list of all disks with at least one photo and disks without exclusive RAM from the manufacturing company.

A range of functions are created that execute more complex operations on the data, such as 'averagePhotosSizeOnDisk', 'getCostForDescription', 'isCompanyExclusive', and 'getClosePhotos'.

**Design Decisions**

1. Creating views to optimize performance for frequently used operations was a crucial decision. This helps reduce load on the database during high traffic.

2. Functions encapsulate specific database operations, providing an interface for the API to interact with the database. This approach adheres to the principle of separation of concerns, making the system more maintainable and less error-prone.

---

# Hw_dry

(1

הבעיה היא בקטע הבא:

**WHERE… l1.CourseName = l2.CourseName…**

**GROUP BY l1.StudentName, l2.StudentName**

**HAVING COUNT(DISTINCT l1.CourseName) = COUNT(Distinct l2.CourseName)**

**AND MIN(l1.Grade)< MIN(l2.Grade)**

התנאי הראשון בודק אם מספר הקורסים שלמד התלמיד הראשון (l1) שווה למספר הקורסים שלמד התלמיד השני (l2) לאחר שלקחנו את הקורסים שהם זהים בהם (שהוא עצמו מנוון כי בהכרח שהסכומים יהיו שווים) אבל התנאי לא בודק אם יש קורסים שהם נפרדים. לכן, לא מובטח ששני התלמידים למדו בדיוק את אותם הקורסים.

למשל, אם תלמיד א' לוקח את הקורסים X ו-Y, ותלמיד ב' לוקח את הקורסים Y ו-Z. התנאי l1.CourseName = l2.CourseName יהיה נכון עבור הקורס Y ששני התלמידים למדו. כאשר אנו מקבצים לפי שמות התלמידים, ה- HAVING COUNT יתקיים גם מכיוון שלשני התלמידים יש רק את Y בקבוצה. עם זאת, הם לא למדו בדיוק את אותם הקורסים (X שונה מ Z), וזו הייתה הדרישה.

התנאי השני בודק שציון המינימום מכל הקורסים של התלמיד הראשון (l1) נמוך מהציון המינימלי מכל הקורסים של התלמיד השני (l2). עם זאת, תנאי זה אינו מבטיח שהתלמיד השני תמיד קיבל ציון גבוה יותר בכל הקורסים שלמדו יחד. זה רק מבטיח שהתלמיד השני קיבל ציון גבוה יותר לפחות בקורס אחד.

לכן, השאילתה מחזירה צמדי סטודנטים שלמדו את אותו מספר של קורסים נפרדים באותם סמסטרים ושם הסטודנט השני קיבל ציון גבוה יותר לפחות באחד מהקורסים. אבל זה לא מבטיח

שצמדי הסטודנטים הללו למדו בדיוק את אותם קורסים ושהתלמיד השני קיבל ציון גבוה יותר בכל הקורסים שלמדו יחד.

(2

**(StudentName, CourseName, Grade, Semester)**

(Beni, Database, 91, Spring 2023)

(Dany, Database, 90, Spring 2023)

(Beni, Math, 79, Spring 2023)

(Dany, Math, 80, Spring 2023)

כאן, גם דני וגם בני למדו את אותם הקורסים באביב 2023. עם זאת, למרות שבני קיבל ציון גבוה יותר ב"מסדי נתונים", דני השיג ציון גבוה יותר ב"מתמטיקה". השאילתה שנוצרה עלולה לקבל באופן שגוי את הצמד הזה מכיוון שבמקרה שבני הוא הסטודנט הראשון (l1) היא תבדוק רק אם הציון המינימלי של בני (79 ב"מתמטיקה") נמוך מהציון המינימלי של בני (80 ב"מתמטיקה") ותקבל בסתירה לדרישה שהציון של בני יהיה נמוך בכל המקצועות מדני.

(3

נגדיר את השאילתה הנכונה בצורה הבאה:

```
SELECT a.StudentName AS n1, b.StudentName AS n2

FROM Learns AS a

JOIN Learns AS b ON a.CourseName = b.CourseName AND a.Semester = b.Semester AND
a.StudentName < b.StudentName

WHERE NOT EXISTS(

(SELECT CourseName, Semester FROM Learns WHERE StudentName = a.StudentName

    EXCEPT

    SELECT CourseName, Semester FROM Learns WHERE StudentName = b.StudentName)

   UNION ALL

(    SELECT CourseName, Semester FROM Learns WHERE StudentName = b.StudentName

    EXCEPT

    SELECT CourseName, Semester FROM Learns WHERE StudentName = a.StudentName)

)

AND EXISTS(

   SELECT 1 FROM Learns AS a2
```

```
    JOIN Learns AS b2 ON a2.CourseName = b2.CourseName AND a2.Semester = b2.Semester
AND a2.StudentName = a.StudentName AND b2.StudentName = b.StudentName

    WHERE b2.Grade > a2.Grade

)

GROUP BY a.StudentName, b.StudentName;
```

ה- NOT EXISTS מבטיח שנקבל רק זוגות של תלמידים שלוקחים בדיוק אותם קורסים.

וה- EXISTS מבטיח שהזוגות המתקבלים הם אלו שבהם התלמיד הראשון תמיד השיג פחות מהשני בכל הקורסים שלמדו יחד.