Linnaeus University

2DV513 - Database Theory - Assignment 2

Students
Albert TÉLLEZ
Heysem ASKAN
Student IDs
at222vk, ab224hv

Contents

1 Task 1

In this section we will use relational algebra to represent the requested queries.

1. $\pi_{(name)}(\sigma_{(code=2DV513)}(student \bowtie enrolledIn))$
2. $\pi_{(name)}(\sigma_{(code=2DV513\ \&\ code=1dv513)}(student \bowtie enrolledIn))$
3. $\pi_{(lecturer)}(\sigma_{(code=2DV610)}(subject))$
4. $\pi_{(lecturer)}(\sigma_{(code=1DV513\ \&\ code=2DV513)}(subject))$
5. $\pi_{(name)}((subject - \sigma_{lecturer=ilir} subject) \bowtie enrolledIn \bowtie student)$

2 Task 2

2.1 Part 1

Room has a dependency with day, time and manager. The reason we need manager in this dependency is because since two different rooms in the building could be booked at the same time in a day it is the manager that distinguishes the specific room.

Time has a dependency with manager, applicant, and day. Since for a specific applicant and manager at a certain date there will only be one time for the interview. If another interview must be conducted for the same applicant and manager, it will occur on a different date.

2.2 Part 2

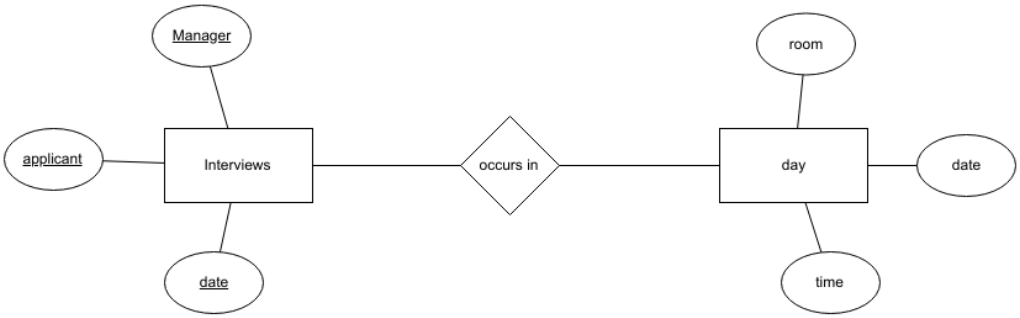
We believe that in this relation the primary key is [manager,applicant,day] since each single manager can interview a single applicant on a single day at once. With these 3 columns we could find any individual row in the table.

2.3 Part 3

Firstly, the relation is not in BCNF because it contains a transitive dependency between room and time. Time is a prime value that depends on the key, and room is a prime value that depends on the time.

2.4 Part 4 and 5

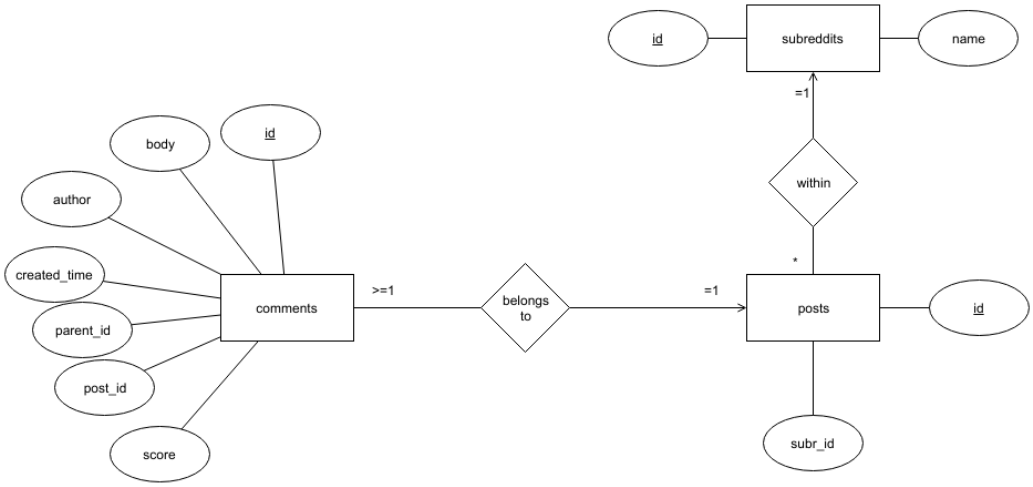
In the following diagram the relation was split into a reasonable arrangement which complies with BCNF.



The interviews have as attributes manager, applicant and date. The date is a new attribute that can be used to relate to the new "day" entity set. All 3 attributes of interview can be used to identify any row in the table as mentioned before. Since a single applicant can only have an interview with a specific manager once a day it means that if any of these values is different, it will point to another row in the table. This is why they are considered keys in the diagram. The day entity set exists to hold information regarding the time of the interview for that specific day as well as the room it will occur in. It is a many to many relationship since overall many interviews can happen in a day (so long as they are done in different rooms or by different managers) and a day can have many interviews.

3 Task 3

In this section we modelled what our database should look like as well as the attributes that we would need to use. The result was a database with the following structure:



We decided to create 3 different tables. One for the comments, another for the posts, and finally one for the subreddits. In the constrained database these are related using foreign keys.

In our specific scenario since it is a comment database there cannot be post without a comment. This explains the multiplicity in the diagram between comments and posts.

Each post must also belong to a subreddit in our case, even though in the real scenario in Reddit there could be a subreddit without posts. In the constrained database the id attribute for each entity set will be the primary key.

In both constrained and non-constrained databases the data types will be the following:

- subreddits
  -id: VARCHAR(15)
  -name: VARCHAR(255)

- posts
  -id: VARCHAR(15)
  -subr_id: VARCHAR(15)

- comments
  -id: VARCHAR(15)
  -parent_id: VARCHAR(15)
  -body: TEXT
  -score: INT
  -created_time: INT(11)
  -author: VARCHAR(100)
  -post_id: VARCHAR(15)

4 Task 4

In order to import the files into the database a program written in JavaScript for NodeJS was used. The way we inserted the information into the database was by reading a certain amount of lines off of the current file, curating the data into its separate tables within the code and then using a bulk insert query. In the un-constrained database the order of insertion of the tables would not be an issue. On the other hand in the constrained database the order of the data must be inserted starting with subreddits, followed by posts ending with comments. The reason this has to be done is to satisfy the foreign key constraints.

Originally our program was designed to import data line by line. In the unconstrained database this was not a big concern since it took around 30 minutes to complete the import. On the other hand, for the same file in the constrained database it took over 4 hours. This was unreasonable, thus we decided to re-code the application. Now, when using the bulk insert type of queries the un-constrained database and the constrained database do not seem to have a measurable time difference when importing data since all the tests we did with the same file and constraints resulted in somewhat different times (15000ms-26000ms). We suspect this was due to external performance hits and not whether the database was constrained or not. This could also be repeated for the second file, with an average of 1110000 ms/18 minutes.

We believe importing all the data and turning constraints afterwards could be a reasonable solution so long as there is confidence that the data to be imported will not break the constrains that are to be added afterwards, in short, the data should be validated before being imported.

5 Task 5

1. In our database the author could be found within the comments table meaning that there was no joining needed. To find another author's comments simply replace the name in the quotes.

```
SELECT COUNT(author) FROM comments WHERE author = 'tony28';
```

2. Due to our database structure, we had to join all the tables in order to reach the subreddit information for all the comments then we filtered out by the property of the subreddit name.

```
SELECT COUNT(comments.id) FROM comments JOIN posts ON comments.post_id = posts.id JOIN
subreddits ON posts.subr_id = subreddits.id WHERE subreddits.name = 'memes';
```

3. In this query we attempted to find all the possible combinations where "lol" would be used as a single word (not within another word) making us having to use many wildcards for the query.

```
SELECT COUNT(comments.id) FROM comments WHERE body LIKE '% lol' OR 'lol %' OR '% lol %'
OR '% lol.' OR '% lol.%' OR 'lol %' OR 'lol?%' OR 'lol!%' 'lol,%';
```

4. We used a nested query to find all the authors in a post, then pass that into another query that would join all the tables together and select only the name column from the subreddits. Since some values were repeated we used the "DISTINCT" keyword to filter out doubles.

```
SELECT DISTINCT subreddits.name FROM subreddits JOIN posts ON subreddits.id =
posts.subr_id JOIN comments ON posts.id = comments.post_id WHERE author IN (SELECT
author FROM comments WHERE post_id = 't3_2u29i');
```

5. For this query we created a temporary table we could use to calculate the score for each author in the database. After this we used min and max on the created view to display the minimum and maximum scores.

```
CREATE VIEW scoreTable AS SELECT author,sum(score) AS addedScores FROM comments GROUP BY
 author;
SELECT max(addedScores),min(addedScores) FROM scoreTable;
```

6. We made many attempts in order to perform the requested query but none of them seemed to give fruitful results. Unfortunately this query did not work well either, since it did not display anything, but it is the one we believe is the most correct. We believe the issue appears when joining the tables and attempting to perform a "GROUP BY".

```
SELECT subreddits.name,max(comments.score),min(comments.score) FROM comments JOIN posts
 ON comments.post_id = posts.id JOIN subreddits ON posts.subr_id = subreddits.id GROUP
BY subreddits.name;
```

7. For the following query we had to use "GROUP BY" as well as join. Since we cannot provide a query we will provide an explanation on what could be done. Firstly, for a specified user we would have to find all the posts that user has commented on. This should not be difficult since it was done in query 1. Then out of those posts we would need all the users whose link_id (post id) would be the same as the ID of the posts we have just found. These are all the users that our fetched used could have interacted with.

8. Unfortunately for the last query we did not find a way to perform it.

The queries that worked "best" for us were the queries that simply worked. We did not know how to re-write the query in a different way to increase performance. In fact, we believe that for "ease" of writing the queries we might have done performance heavy solutions (such as using temporary views).