CS486/586 Introduction to Databases

Spring 2022 Quarter

Assignment 5

Constraints, Indexes, Operator Implementation

Due: Wednesday May 18th at 0900 Pacific Time

You may do this assignment individually or you may work in a group of up to five people total. You should only talk to the instructor, the TA and your group about this assignment. You may also post questions to the class Slack channel.

Please turn in your completed assignments as a .pdf or .txt file. Put your last name, first name, the assignment number in that order in the first line of your assignment. List last name and first name for each additional member of your group, if you have one, on the second line of your assignment. (If you are working with a group, turn in one assignment paper.)

#### **Part I: Checking Constraints**

Sometimes it is necessary or desirable for an application to check constraints on a database rather than the database itself. Sometimes this is because the DBMS cannot enforce a particular constraint. Another example of this is implementing a “soft constraint” through an application, where we do not want to forbid a violation being written, but we want to know if one occurs.

To implement a soft constraint C, write a query QC that returns any violations of C. Then, if QC returns an empty result, we know the current database satisfies the constraint.

Questions 1-8 below involve creating queries to check different soft constraints on the spy database. *For each query, run it against the spy database and report the query and number of results returned.* Say whether or not the constraint is satisfied.

**Question 1** (10 points): Give a query to check whether name is a key for the Mission table.

SELECT name, COUNT(\*)

FROM mission

GROUP BY name

HAVING COUNT(\*) > 1;

name | count

-----------------+-------

Northern Fences | 2

(1 row)

SELECT name

FROM "mission" m1

WHERE (

SELECT COUNT(\*)::int

FROM "mission" m2

WHERE m1."name" = m2."name"

) > 1;

2 rows returned, thus ‘name’ is not a key for the ‘mission’ table because we had a duplicate row returned.

Hint: You can check whether a table currently satisfies a key by writing a query that looks for any value for the key column(s) that occurs more than once.

**Question 2** (10 points): Give a query to check whether {salary, city} is a key for the Agent table.

Write a query to check if the specific set (salary, city) has any duplicates

SELECT (salary, city), COUNT(\*)

FROM agent

GROUP BY (salary, city)

HAVING COUNT(\*) > 1;

(37 rows)

SELECT salary, city

FROM "agent" a1

WHERE (

SELECT COUNT(\*)::int

FROM "agent" a2

WHERE a1.salary = a2.salary AND a1.city = a2.city

) > 1;

84 rows returned, thus {salary, city} are not a key for the ‘agent’ table as non-unique rows were returned.

Hint: You can check a 1..\* cardinality condition with a query that looks for rows of one entity type that fail to match up any rows of the other entity type.

**Question 3** (10 points): Give a query to check whether each agent speaks at least one language.

SELECT agent\_id, COUNT(\*)

FROM agent

NATURAL JOIN languagerel

GROUP BY agent\_id

HAVING COUNT(\*) < 1;

0 rows returned

SELECT agent\_id

FROM agent

WHERE agent\_id NOT IN (

SELECT agent\_id

FROM languagerel

);

0 rows returned, thus all agents speak at least one language since no rows were returned to show agents who spoke less than 1 language.

**Question 4** (10 points): Give a query to check whether each agent is on at least one team.

SELECT A.agent\_id

FROM agent A

LEFT JOIN teamrel T ON A.agent\_id = T.agent\_id

WHERE T.team\_id IS NULL;

Returns 368 rows of agents without team\_id’s. There are 662 total agents, and 294 of them are on teams (SELECT COUNT(DISTINCT teamrel.agent\_id) FROM teamrel;)

Hint: You can write test queries for cardinality constraints with other values, too.

**Question 5** (10 points): Give a query to check whether every agent speaks at least two languages.

SELECT COUNT(DISTINCT agent.agent\_id)

FROM agent

JOIN languagerel ON agent.agent\_id = languagerel.agent\_id

JOIN language ON languagerel.lang\_id = language.lang\_id

WHERE agent.agent\_id = languagerel.agent\_id

GROUP BY agent.agent\_id

HAVING COUNT(DISTINCT language.lang\_id) >= 2;

541 rows returned, thus not every agent speaks at least two languages, only 121 agents speak 2 or more languages since there are 662 agents total (SELECT agent\_id FROM agent;)

SELECT A.agent.id, COUNT(\*)

FROM agent A

LEFT JOIN languagerel l ON A.agent\_id = l.agent\_id

GROUP BY A.agent\_id

HAVING COUNT(\*) < 2;

Returns 121 rows of single language speakers

**Question 6** (10 points): Give a query to check whether every agent has six or fewer skills.

SELECT agent\_id, COUNT(\*)

FROM skillrel

GROUP BY agent\_id

HAVING COUNT(\*) > 6;

0 rows returned

SELECT COUNT(\*)

FROM (

SELECT agent\_id

FROM skillrel

GROUP BY agent\_id

HAVING COUNT(\*) > 6

) AS t1

JOIN agent ON agent.agent\_id = t1.agent\_id;

1 row returned, the rows value for ‘count’ was 0, thus all agents have 6 or less skills, no agents have 7 or more skills.

**Question 7** (10 points): Give a query that checks whether the maximum salary in each city is no more than 150% the minimum salary for that city.

SELECT A.city, MIN(A.salary), MAX(A.salary)

FROM agent A

GROUP BY city

HAVING (MIN(A.salary)\*1.5) < MAX(A.salary);

26 rows returned, meaning that there are 26 cities that have max salaries that are greater than 150% of the minimum salary

SELECT city, MIN(salary), MAX(salary)

FROM agent

GROUP BY city

HAVING MAX(salary) <= 1.5 \* MIN(salary);

20 rows returned, thus 20 cities out of the 46 total are within the 150% range from min salary to max salary.

**Question 8** (10 points): Give a query that checks that every team has at least three skills represented among its agents.

SELECT T.team\_id, COUNT(T)

FROM teamrel T LEFT JOIN skillrel R ON T.agent\_id = R.agent\_id

GROUP BY team\_id

HAVING COUNT(T)<3;

1 row returned (team\_id: 42, count: 2), this team only has 2 skills represented meaning that not all teams have at least three skills

SELECT team\_id, COUNT(skill\_id)

FROM teamrel

JOIN agent ON agent.agent\_id = teamrel.agent\_id

JOIN skillrel ON skillrel.agent\_id = agent.agent\_id

GROUP BY team\_id

HAVING COUNT(DISTINCT skill\_id) > 2;

41 rows returned, thus 41 of the 44 teams have at least 3 skills amongst its agents.

**Part II: Evaluating Queries with Just Indexes**

It is sometimes possible to evaluate a particular query using only indexes, without accessing the actual data records.

Consider a database with two tables:

Pokemon(charName, attack, stamina, pType)

Captured(charName, player, difficulty)

Assume three indexes, where the leaf entries have the form   
[search-key value, RID].

<stamina> on Pokemon

<pType> on Pokemon

<difficulty, charName> on Captured

These indexes are unclustered, meaning we can’t count on the records being stored in-order by their search key.

For Questions 5-12, say which queries can be evaluated with just data contained in these indexes. If the query can, describe how. If the query can't, explain why.

Each question is worth 10 points.

**Question 9**

SELECT AVERAGE(difficulty)

FROM Captured

GROUP BY player;

This query can’t be evaluated because player is not an attribute present in the index for Captured.

**Question 10**

SELECT MIN(difficulty)

FROM Captured;

This query can be evaluated because an index exists on Captured that has all difficulty entries. Because the indexes are unclustered, it’s necessary to search the whole index for the minimum instead of going to the smallest value.

**Question 11**

SELECT charName, MAX(difficulty)

FROM Captured

GROUP BY charName;

This can be done, the index on Captured has sufficient attributes that can be grouped by charName. Again, the unclustered index must be traversed completely to find the max.

**Question 12**

SELECT pType, COUNT(charName)

FROM Pokemon

GROUP BY pType;

This query cannot be evaluated, charName is only available in the Captured table and can not be put in the aggregate function COUNT when grouping by Pokemon attributes.

**Question 13**

SELECT COUNT(\*)

FROM Pokemon

WHERE pType = 'Water' AND stamina = 85;

This can be done, the index on Pokemon has sufficient data where the system can form AND and OR conditions across several index scans. Use each index with the appropriate query clause and then AND together the index results to identify the result rows.

**Question 14** (think carefully about this one)

SELECT charName, AVERAGE(difficulty)

FROM Captured

GROUP BY charName

HAVING COUNT(DISTINCT player) > 1;

This can not be done, since player is not an attribute on the index of Captured. It can not be evaluated by the index alone.

**Question 15**

SELECT pType, difficulty

FROM Pokemon, Captured

WHERE stamina = 85

AND Pokemon.charName = Captured.charName;

This can be done, we can obtain the data from the index with just stamina then compare the charName between the Captured and Pokemon to get the resulting rows.

**Question 16**

SELECT COUNT(DISTINCT stamina)

FROM Pokemon

WHERE pType = 'Water';

This can be done, as pType and stamina are both attributes on the index of Pokemon.

#### **Part III: Operator Implementation**

This question is about implementing outer join. Assume we have the tables Pokemon and Captured from the previous part, and we want to compute

Pokemon LEFT OUTER JOIN Captured ON charName

**Question 17** (10 points): Consider computing this join using a nested-loops algorithm with Pokemon in the outer loop and Captured in the inner loop. Explain how to adapt the nested-loops-join algorithm to produce the additional rows needed for the outer join.

FOR each tuple P in Pokemon DO

FOR each tuple C in Captured DO

IF C.charName matches P.charName join to make tuple T THEN

Output T;

END;

END;

For this to output Captured as C that don’t match up to Pokemon as P, we need to make a temporary table that would keep track if any matchings happened. The table would hold the name of C.charName, and as the inner nested loop iterates through C it marks 0 if unmatch to a P.charName, or 1 for a match. Once the outer loop completes, any C.names with 0’s will be output as NULL.

**Question 18** (15 points): Now suppose that Pokemon is in the inner loop, and Captured the outer loop. Explain how to adapt nested-loop joins to produce the additional rows. Be sure to describe any additional information that the algorithm must maintain.

For each tuple in C Captured DO

For each tuple in P Pokemon DO

IF C.charName matches P.charName

join to make tuple T Then output T;

ELSE output tuple P;

END;

END;

In this version, the algorithm needs to additionally store information about the tuples from the relation pokemon which do not have matching equivalents in the relation captured.

Note: Don’t confuse INNER and OUTER join options with what are sometimes called the *inner* and *outer* inputs to a join algorithm.