1 Question 1

1.1 Premise

Consider the following schema and data:

course (c_id, dept_id, dept, evaluations, inst, office, sect, time_slot)

<table>
<thead>
<tr>
<th>c_id</th>
<th>dept_id</th>
<th>dept</th>
<th>evaluations</th>
<th>inst</th>
<th>office</th>
<th>sect</th>
<th>time_slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>1</td>
<td>CS</td>
<td>HW, Midterm, Final</td>
<td>Eddie Kohler</td>
<td>345</td>
<td>A</td>
<td>8am-10am</td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>CS</td>
<td>HW, Midterm, Final</td>
<td>Eddie Kohler</td>
<td>345</td>
<td>B</td>
<td>1pm-3pm</td>
</tr>
<tr>
<td>165</td>
<td>2</td>
<td>MATH</td>
<td>Project, Final</td>
<td>Stratos Idreos</td>
<td>346</td>
<td>A</td>
<td>10am-11am</td>
</tr>
<tr>
<td>165</td>
<td>2</td>
<td>MATH</td>
<td>Project, Final</td>
<td>Stratos Idreos</td>
<td>346</td>
<td>B</td>
<td>7pm-8pm</td>
</tr>
<tr>
<td>265</td>
<td>2</td>
<td>MATH</td>
<td>HW</td>
<td>Andy Pavlo</td>
<td>346</td>
<td>B</td>
<td>1pm-2pm</td>
</tr>
<tr>
<td>455</td>
<td>3</td>
<td>PHYS</td>
<td>Midterm, Final</td>
<td>Nesime Tatbul</td>
<td>347</td>
<td>B</td>
<td>6pm-7pm</td>
</tr>
</tbody>
</table>

Throughout this problem, we will take this schema through four different normal forms to gain a sense of what benefits each one brings. Note that it will be useful to you to create populated tables using the data above for each schema you create, but it is not required. Lastly, for each conversion, be sure to document and explain your steps; not too much, just enough for us to understand your thought process.

1.2 1.1 First Normal Form

A schema is in the first normal form when: - The attributes are atomic, and - The attribute names are unique.

We already have the latter, so we’ll focus on the former. Once you’ve identified the non-atomic attributes, you can either split them into multiple attributes or into a separate relation entirely.

Convert this schema into the first normal form. You should have three relations by the end of your conversion.

1.3 1.2 Second Normal Form

A schema is in the second normal form when: - The schema is in first normal form (done!) - There are no partial dependencies.

In other words, say there’s a candidate key $A$. Then, every other attribute in the relation should depend on all of $A$, not a subset of it. If there is an attribute such that only a part of $A$ functionally determines it, there is a partial dependency. Once you’ve identified a partial dependency, you can normalize it by splitting the relation in two: one for attributes that are partially dependent, and one for those that aren’t.

Convert this schema into the second normal form. You should have four relations by the end of your conversion.

1.4 1.3 Third Normal Form

A schema is in the third normal form when: - The schema is in second normal form (done!) - There are no transitive dependencies.
A transitive dependency is a dependency between two sets of attributes where both sets are still dependent on the primary key. Formally, if \( A \rightarrow B \) and \( B \rightarrow C \) and NOT \( B \rightarrow A \), then \( A \rightarrow C \) is a transitive dependency. Once you’ve identified a transitive dependency, you can separate the dependent attributes out into another relation.

Convert this schema into the third normal form. You should have five relations by the end of your conversion.

(Hint: in these instances of our relations, and conceivably for all potential instances, instructors only teach one course).

1.5 1.4 BCNF (Third-and-a-half Normal Form)

A schema is in BCNF when: - The schema is in third normal form (done!) - For all dependencies, either: - \( a \rightarrow b \) is a trivial dependency (\( a \subseteq b \)), or - \( a \) is a superkey.

Convert this schema into BCNF. You should have six relations by the end of your conversion.

2 Question 2

Consider the following schema \( S \) and constraints \( F \):

\[\begin{align*}
\text{users} & \quad (\text{user_id}, \text{name}, \text{dept_id}, \text{dept_name}) \\
\text{books} & \quad (\text{book_id}, \text{isbn}, \text{title}, \text{author}, \text{publisher})
\end{align*}\]

\[
\begin{align*}
\text{user_id} \rightarrow & \text{name} \\
\text{user_id} \rightarrow & \text{dept_id} \\
\text{user_id} \rightarrow & \text{dept_name} \\
\text{dept_id} \rightarrow & \text{dept_name} \\
\text{book_id} \rightarrow & \text{isbn} \\
\text{isbn} \rightarrow & \text{title} \\
\text{isbn} \rightarrow & \text{publisher} \\
\text{isbn} \rightarrow & \text{author}
\end{align*}
\]

Complete the following exercises:

1. What are the additional functional dependencies that can be generated by Armstrong’s Transivity Rule?
2. Compute the canonical cover of this set of constraints, \( F_C \). (Hint: use the algorithm described in class).
3. Normalize the given schema to BCNF. Was your normalization lossy? Was your normalization dependency preserving?

3 Question 3

Use Armstrong’s axioms (reflexivity, augmentation, and transitivity ONLY) to prove the following rules:

1. If \( a \rightarrow b \), then \( a \rightarrow ab \) (extensivity).
2. If \( a \rightarrow b \) and \( a \rightarrow c \), then \( a \rightarrow bc \) (union rule).
3. If \( a \rightarrow b \) and \( bc \rightarrow d \), then \( ac \rightarrow d \) (pseudotransitivity).
4 Question 4

You’ve learned how to create table-level checks during table creation using key words such as NOT NULL, UNIQUE, or CHECK. Now, we’ll learn how to create arbitrary assertions over multiple tables that your database should check.

Note that global assertions can be expensive for your database system to maintain; they need to be checked every time a change is made. Moreover, not every SQL implementation has assertions, nor do they all do it the same way. Don’t worry about performance or syntax-correctness too much for this homework, but keep it in mind for future database design decisions.

An assertion essentially takes a subquery and checks whether it evaluates to some result. The following makes sure that there are exactly 0 users:

```
CREATE ASSERTION assertion_name CHECK (0 = (SELECT COUNT(*) FROM users))
```

Consider the following schema:

| users (user_id, name, dept_id, dept_name) |
| books (book_id, isbn, title, author, publisher) |
| reads (user_id, book_id) |

Now, create assertions to check the following:

1. Make sure that there is never a book by “JK Rowling” that is also published by “Penguin” (Hint: use NOT IN).
2. Make sure that the number of users in a department never exceeds 100.
3. Make sure that nobody is reading more than 5 books.

5 Question 5

As of now, we’ve mostly been focusing on OLTP workloads. OLTP (OnLine Transactional Processing) workloads are commonly used for many systems because they are very good at retrieving data. Unfortunately, OLTP databases are a poor choice for asking questions about the data, like calculating statistics over the data. In that case, we use OLAP (OnLine Analytical Processing). You don’t need a perfect understanding between the two for this question. You only need to understand that we sometimes (not always!) structure data differently when the situation calls for it. We encourage you to read more about the differences here.

One schema design used in OLAP workloads is the Star Schema. In this style, we represent facts as a centralized table (for example, in this question: when a worker bee collects honey, this is recorded in the fact_work table). Dimensions about these facts are represented in related tables. Consider the following star schema for Barry’s bees.

**ER Diagram:**

**Star Schema:**

```
fact_work(worker_id, queen_id, honey_id, hive_id, quantity_milliliters, hours_worked)
dim_worker(worker_id, worker_name)
dim_queen(queen_id, queen_name)
dim_hive(hive_id, location)
dim_honey(honey_id, type)
```

**Normalized Schema:**
workers(worker_id, worker_name, hive_id, queen_id)
queens(queen_id, queen_name)
hives(hive_id, location)
honey(honey_id, type)
work(work_id, worker_id, honey_id, quantity_milliliters, hours_worked)

1. Does the star schema meet 1NF? Why or why not?
2. Does the star schema meet 2NF? Why or why not?
3. Does the star schema meet 3NF? Why or why not?
4. Does the star schema meet BCNF? Why or why not?
5. Suppose we want to know how many workers live in each hive. Can we write an SQL query to find this for the star schema? How about the normalized schema?
6. Suppose we want to know how many quantity_milliliters of honey were produced for each queen, along with the queen bee’s name. Write a SQL query to find this for both the star schema and the normalized schema.
7. In what ways is this star schema “better” than a normalized schema?

______________________________________________________________

Feedback

As this is a new course, we appreciate any feedback you have for us! If you enjoyed this assignment, hated this assignment, or have other thoughts to share, please do so here!