Sai University

Data Base Management Systems

Mid-Semester Examination (Make-Up) with Solutions,

Semester A, 2023–2024

Date	: 25 October, 2023	Marks	: 20
Duration	: 90 minutes	Weightage	: 20%

1. Consider a university database with relations

   - $student(sid, name)$ — student ID and name for each student
   - $course(cid, title)$ — course ID and title for each course
   - $passed(sid, cid)$ — which courses each student has cleared already, in terms of student and course IDs
   - $core(cid)$ — IDs of core courses

   Write relational algebra expressions to compute each of the following relations:

   (a) IDs of students who have not yet cleared *any* course *(2 marks)*

   Solution $\pi_{sid}(student)$ is the set of all student IDs and $\pi_{sid}(passed)$ is the set of all student IDs who have passed at least one course. Hence, the required expression is $\pi_{sid}(student) \setminus \pi_{sid}(passed)$, where $X \setminus Y$ denotes set difference between sets X and Y.

   (b) IDs of students who have not yet cleared all core courses *(2 marks)*

   Solution

   - $\pi_{sid}(student) \times core$ is the set of all possible pairs (sid, cid) for students and core courses. Let $r \leftarrow \pi_{sid}(student) \times core$.
   - $passed \bowtie core$ is the set of pairs (sid, cid) where student sid has passed core course cid. Let $s \leftarrow passed \bowtie core$.
   - If a student with ID sid has not cleared a core course with ID cid, then (sid, cid) will appear in $r \setminus s$. So the desired query is $\pi_{sid}(r \setminus s)$.

   (c) IDs of core courses that are still pending for at least one student *(2 marks)*

   Define r and s as above. Any core course cid that is still pending for a student sid will appear as (sid, cid) in $r \setminus s$. So the desired query is $\pi_{cid}(r \setminus s)$.

2. For the ongoing cricket World Cup, a database is being maintained that includes the following two tables:

   Players(PlayerID, Name, Country)
   Runs(MatchNumber, PlayerID, Runs)

   Each country has a list of players registered for the World Cup and *PlayerID* is a unique ID assigned to each registered player. The competition consists of 48 matches and each match is identified by its *MatchNumber*, from 1 to 48.

The *Players* table lists all the registered players. The *Runs* table records the runs scored in all the matches played so far, and is updated after every match. For each match, the table records runs for those players who actually batted. There is no entry for a player who did not play in the match or who played but did not get a chance to bat.

Whenever a player comes to bat, the screens at the stadium show the runs that the player has scored so far in this World Cup. To compute this quantity, we need "join" the two tables above to create a table of the form

Aggregate(PlayerID, Name, TotalRunsTillNow)

that records the runs scored by *all* registered players, including those who have yet to bat in any match.

(a) Explain why a natural join is not adequate for this purpose. *(2 marks)*

Solution We have to join the two tables on *PlayerID*. If a player has not scored any runs so far — either he has not played any match or he has not batted in the matches he has played — that player's ID will not be present in the relation *Runs*, and hence will not appear in the joined relation *Aggregate*. Our requirement is that such a player should appear in *Aggregate* with *TotalRunsTillNow* set to zero, so a natural join will not suffice.

(b) What kind of join should we use to ensure that *every* player is included in the table *Aggregate*? *(2 marks)*

Solution We need an outer join — specifically, a natural left outer join — to ensure that all plaers are reported in the final table *Aggregate*.

(c) Write an SQL query to compute the table *Aggregate*. *(4 marks)*

We need to group and aggregate the runs in *Runs* by *PlayerID* and then compute a natural left outer join with the table *Players*. Here are two versions of the query.

(i)
```
with CurrentRuns(PlayerID,TotalRunsTillNow) as
    (select PlayerID, sum(Runs)
     from Runs
     group by PlayerID)
select PlayerID, Name, TotalRunsTillNow from
   Players natural left outer join CurrentRuns
```

(ii)
```
select PlayerID, Name, TotalRunsTillNow from
   Players natural left outer join
   (select PlayerID, sum(Runs)
    from Runs
    group by PlayerID)
    as CurrentRuns(PlayerID,TotalRunsTillNow)
```

To fully solve the problem, we should create a view `Aggregate` to store the output of the query.

3. A bank's database contains a table *Accounts(CustomerID, AccountNo, BranchID)*, where *CustomerID* is a unique ID for each customer, *AccountNo* is the bank account number and *BranchID* is a unique ID for each branch of the bank.

   Account numbers are unique across the bank and each account number is attached to one branch of the bank. The bank permits joint accounts, with more than one customer associated with an account. A customer may have many accounts in the bank, but is restricted to at most one account in each branch, whether it is single or joint.

   (a) What functional dependencies can you infer from these constraints? *(2 marks)*

   Solution Since account numbers are unique across the bank, from the account number we can uniquely identify the branch that it belongs to. Since no customer has more than one account in each branch, from the customer ID and the branch ID, we can uniquely identify the customer's account number in that branch. Hence we have two functional dependencies.

   - *AccountNo* $\rightarrow$ *BranchID*
   - *CustomerID, BranchID* $\rightarrow$ *AccountNo*

   (b) Compute a BCNF decomposition of the table *Accounts*. *(2 marks)*

   Solution We have a functional dependency *AccountNo* $\rightarrow$ *BranchID* where the left hand side, *AccountNo*, is not a superkey for *Accounts*, since there can be joint accounts with more than one customer ID mapped to an account number. Hence this relation is not in BCNF. The BCNF decomposition gives two relations

   - *R(AccountNo, BranchID)*
   - *S(CustomerID, AccountNo)*

   (c) Explain whether your BCNF decomposition is dependency preserving. *(2 marks)*

   Solution The functional dependency *CustomerID, BranchID* $\rightarrow$ *AccountNo* cannot be checked locally in either relation R or relation S resulting from the BCNF decomposition. Hence this decomposition is not dependency preserving.