RDBMS and SQL — 29 November 2023
Final Exam, Aug–Nov 2023 — Marks: 50, Weightage: 50%

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 *(3 marks)*

   (b) IDs of students who have not yet cleared all core courses *(3 marks)*

   (c) IDs of core courses that are still pending for at least one student *(3 marks)*

2. During the 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 the World Cup. To compute this quantity, we need to "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. *(3 marks)*

   (b) What kind of join should we use to ensure that *every* player is included in the table *Aggregate*? *(3 marks)*

   (c) Write an SQL query to compute the table *Aggregate*. *(5 marks)*

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 non-trivial functional dependencies can you infer from these constraints? *(2 marks)*

   (b) Recall that a table is in BCNF if for every non-trivial functional dependency $\alpha \rightarrow \beta$, α is a superkey. Compute a BCNF decomposition of the table *Accounts*. *(2 marks)*

   (c) Recall that a decomposition is dependency preserving if all functional dependencies can be checked without having to join any tables in the decomposition. Explain whether your BCNF decomposition is dependency preserving. *(2 marks)*

4. Let relations $r_1(A, B, C)$ and $r_2(C, D, E)$ have the following properties: r_1 has 4×10^6 tuples, r_2 has 3×10^6 tuples, 50 tuples of r_1 fit in one block, and 20 tuples of r_2 fit in one block. Neither r_1 nor r_2 is sorted with respect to any of its attributes.

   (a) Estimate the number of block accesses required for computing $r_1 \bowtie r_2$ using nested-loop join. Explain your answer, including your choice of outer and inner relations. *(2 marks)*

   (b) Estimate the number of block accesses required for computing $r_1 \bowtie r_2$ using block nested-loop join in the following situations. Explain your answers, including your choice of outer and inner relations.

      (i) The memory can hold approximately 10^3 blocks. *(3 marks)*

      (ii) The memory can hold approximately 10^5 blocks. *(3 marks)*

5. Consider the following tables from the university database discussed in class.

   - *Student(ID, name, dept_name, tot_cred)*
   - *Takes(ID, course_id, sec_id, semester, year, grade)*
   - *Course(course_id, title, credits)*

   We want to generate a table *(ID, name, course_id, title)* containing details of courses from 2023 that students from the Physics department have enrolled in.

   Our SQL query for this requirement is initially translated into the following relational algebra expression.

$$\Pi_{ID,name,course\_id,title}(\sigma_{dept\_name\ =\ Physics \wedge year\ =\ 2023}(Student \bowtie (Takes \bowtie Course)))$$

   (a) Draw the expression tree representing this expression. *(3 marks)*

   (b) Modify the expression tree to generate an equivalent tree that is more efficient to evaluate. Justify your transformation and explain why it is more efficient. *(5 marks)*

6. Consider the following schedules of concurrent read and write operations.

Schedule A

T_a	T_b	T_c	T_d	T_e
				$w(x_2)$
			$w(x_1)$	
$w(x_3)$				
				$r(x_1)$
	$r(x_1)$			
$r(x_2)$				
	$w(x_4)$			
		$r(x_4)$		
			$r(x_3)$	
		$r(x_3)$		
$r(x_4)$				
				$r(x_4)$

Schedule B

T_a	T_b	T_c	T_d	T_e
				$w(x_2)$
			$w(x_1)$	
$r(x_1)$				
				$r(x_1)$
	$r(x_1)$			
$r(x_2)$				
	$w(x_4)$			
		$r(x_4)$		
			$r(x_3)$	
		$r(x_3)$		
$r(x_4)$				
				$r(x_4)$

   (a) One of these schedules is conflict-serializable and the other is not. Explain which is which. *(4 marks)*

   (b) For the schedule that is conflict-serializable, list out all possible serial schedules that are consistent with this schedule. *(4 marks)*