Prof. Li-Yan Yuan CMPUT 391: Database Management Systems

Solutions to Assignment 1

Due: 18:00, Feb. 10, 2014, at the 391 Drop Box

1. Present a real-life example (Not using ABCD, etc) to show differences between BCNF and 4NF.

   Solution: Consider the following table

   ```
   real_estate(realtor_id,listing_property,customer_name)
   ```

   used to store the information for a real estate company with one MVD constraint

   $$\text{realtor\_id} \twoheadrightarrow \text{listing\_property} \mid \text{customer\_name}.$$

   It is not difficult to see that `real_estate` is in BCNF but not in 4NF.

2. Consider a database consisting of the following tables with obvious meanings:

   ```
   employee( employee_name, street, city )
   works( employee_name, company_name, salary )
   company( company_name, city )
   ```

   Write triggers, according to Oracle style, to enforce the constraint that no employee is allowed to work for a company NOT located in the city where the employee lives. One needs to (1) give a list of events that must be monitored for imposing the constraint, and (2) present one trigger for any event in (1). [20]

   Solution:

   [1] The following events must be monitored for imposing the constraint:

   INSERT OR UPDATE on all three tables.

   [2] A trigger for monitoring the INSERT or UPDATE on employee is given below.

   ```
   create trigger employee_location
   before insert or update on employee
   for each row
   declare  dummy integer;
   begin
     select count(*) into dummy
     from works, company
     where works.company_name = company.company_name and
             :new.employee_name = works.employee_name and
             :new.city <> company.city;
       if ( dummy > 0 )
       then  raise_application_error(-20502, 'no employee shall live in the
             same city as the location of the company ');
       end if;
   end;
   ```

3. Consider a relation schema $R = ABCDE$, functional dependencies

$$A \rightarrow C$$
$$B \rightarrow C$$
$$C \rightarrow D$$
$$DE \rightarrow A$$

and a decomposition $D = \{AC, AD, BE, AB\}$ of R.

Prove or disprove that D is a join lossless decomposition of R with respect the given set of FDs. A proof can be done by finding an appropriate decomposition tree, and a counter-example is good enough for a disproof.[10]

Solution: We are going to disprove the claim using a counter-example. Consider a table r as follows:

A	B	C	D	E
a1	b	c	d	e1
a2	b	c	d	e2

Then r satisfies the given set of FDs, and further we have

$$\Pi_{AB}(r) \bowtie \Pi_{AC}(r) \bowtie \Pi_{AD}(r) \bowtie \Pi_{BE} =$$

A	B
a1	b
a2	b

$\bowtie$

A	C
a1	c
a2	c

$\bowtie$

A	D
a1	d
a2	d

$\bowtie$

B	E
b	e1
b	e2

$=$

A	B	C	D
a1	b	c	d
a2	b	c	d

$\bowtie$

B	E
b	e1
b	e2

$=$

A	B	C	D	E
a1	b	c	d	e1
a1	b	c	d	e2
a2	b	c	d	e1
a2	b	c	d	e2

$\neq r.$

Therefore, the decomposition is NOT join lossless.

Solution2: This can also be done by using the Tabular test as follows. The initial table is:

A	B	C	D	E
a_1	b_{12}	a_3	b_{14}	b_{15}
a_1	b_{22}	b_{23}	a_4	b_{25}
b_{31}	a_2	b_{33}	b_{34}	a_5
a_1	a_2	b_{43}	b_{44}	b_{45}

Apply $A \rightarrow C$:

A	B	C	D	E
a_1	b_{12}	a_3	b_{14}	b_{15}
a_1	b_{22}	a_3	a_4	b_{25}
b_{31}	a_2	b_{33}	b_{34}	a_5
a_1	a_2	a_3	b_{44}	b_{45}

Apply $B \rightarrow C$:

A	B	C	D	E
a_1	b_{12}	a_3	b_{14}	b_{15}
a_1	b_{22}	a_3	a_4	b_{25}
b_{31}	a_2	a_3	b_{34}	a_5
a_1	a_2	a_3	b_{44}	b_{45}

Apply $C \rightarrow D$:

A	B	C	D	E
a_1	b_{12}	a_3	a_4	b_{15}
a_1	b_{22}	a_3	a_4	b_{25}
b_{31}	a_2	a_3	a_4	a_5
a_1	a_2	a_3	a_4	b_{45}

Apply $DE \rightarrow A$:

A	B	C	D	E
a_1	b_{12}	a_3	a_4	b_{15}
a_1	b_{22}	a_3	a_4	b_{25}
b_{31}	a_2	a_3	a_4	a_5
a_1	a_2	a_3	a_4	b_{45}

Since no row is all a's, the decomposition is not a lossless join.

4. Consider $R = ABCDEGHK$ and the following set F of functional dependencies:

$$E \rightarrow DK$$
$$C \rightarrow K$$
$$EK \rightarrow BC$$
$$HK \rightarrow A$$

(a) Find a join loss-less, dependency preserving and 3NF decomposition of R.

(b) Indicate whether your database schema is in BCNF with respect to F. Explain. [20]

Solution:

(a) We first find a minimal cover of the FDs, as shown below.

Right reduced	Left Reduced:	Minimal Cover
$E \rightarrow D$	$E \rightarrow D$	$E \rightarrow D$
$E \rightarrow K$	$E \rightarrow K$	$C \rightarrow K$
$C \rightarrow K$	$C \rightarrow K$	$E \rightarrow B$
$EK \rightarrow B$	$E \rightarrow B$	$E \rightarrow C$
$EK \rightarrow C$	$E \rightarrow C$	$HK \rightarrow A$
$HK \rightarrow A$	$HK \rightarrow A$	

Then construct a database $D' = \{EDBC, CK, HKA\}$.

Now, we need to check if D' contains any candidate key.

Since no FD in the minimal cover above contains E, G, or H in its right side, any candidate key shall contain these three attributes. Further, it is not difficult to check that EGH is indeed a candidate key. Therefore, EGH is the only candidate key of R, and shall be added to D'.

Hence, $D = \{EDBC, CK, HKA, EGH\}$ is a join loss-less, dependency preserving and 3NF decomposition of R.

(b) D is in BCNF since all the non-trivial FDs $X \rightarrow A$ in held in any relation $R_i \in D$, X is a key of R_i.

5. Consider the following two algorithms for computing $R \bowtie S$:

   (a) index,

   (b) merge-sort join.

   Assume that (1) both tables are stored in B-tree, (2) the main memory can accommodate 502 pages, (3) R is stored in 20,000 pages with 20 tuples per page, and S is stored in 50,000 pages with 5 tuples per page, (4) there exists a non-clustered index on the join attribute of both R and S; and (5) each index search takes **1.6** disk accesses on average. Note that since the index on the join attribute of each table is non-clustered, neither R nor S is sorted on the join attribute.

   For each algorithm, estimate the number of page accesses, excluding outputting the joined result table, with an optimized configuration. By an optimized configuration we mean the arrangement of all the relevant factors such that the estimated number of disk accesses is less than that of other arrangements using the same algorithm. Note that at most 8-ways merge-sort is allowed. (That is, it is not allowed to merge more than 8 sub-sorted files in one merge operation.) [20]

   Solution

   (a) Index

   Case 1: 500 pages for R, 1 page for S and 1 for output.

   We need to load all 20,000 pages, in 40 steps, and for each tuple in R, one index search on S must be performed.

   Therefore, the total number of pages loaded is 20,000 + 20,000 * 20 *1.6 = 660,000 (pages).

   Case 2: 500 pages for S, 1 page for R and 1 for output.

   We need to load all 50,000 pages, in 100 steps, and for each tuple in S, one index search on R must be performed.

   Therefore, the total number of pages loaded is 50,000 + 50,000 * 5 *1.6 = 450,000 (pages).

   Thus loading S in memory and having an index in R (Case 2) is more efficient.

   (b) Merge Sort

   Merge Sort of S: Total Number of pages of S : 50,000.

   With the replacement selection, we shall be able to obtain sorted subfiles (runs) with the average length of 1000 pages, with the memory of 500 pages. Therefore, it takes one loading and one writing of S to obtain 50 runs.

   Since at most 8 ways of merge sort is allowed, it takes two levels of merge-sort to sort S. Further, each level of merge-sort takes one loading and one writing of S.

   Therefore, the cost of sorting S is three loadings and three writings of S, which is 300,000 pages.

   Merge Sort of R:

   Total Number of pages of R : 20,000.

   In average, the number of runs generated is 20,000/1,000 = 20. Therefore, the two level of multiple merge-sort is needed.

   The cost of sorting R is thus also three loadings and three writings of R which is 120,000 pages.

Final joining: The cost of the merge join of both sorted S and R is 20,000 + 50,000 = 70,000. So, the total cost of joining R and S using the merge sort join is 300,000 + 120,000 + 70,000 = 490,000.