COMP9311 Assignment 2

Shu Hao Ivan Teong (z3386180)

May 3, 2015

Abstract

Database systems assignment investigating SQL queries, functional dependencies, decomposition and MRU buffer replacement policy.

Question 1 - Airline Flights SQL queries

For each pilot, list their employee ID, name, and the number of routes he can pilot:

```
SELECT employees.eid AS employee_id, employees.ename AS pilot_name,
count(distinct flights.flno) AS number_of_routes_he_can_pilot
FROM flights, aircraft, certified, employees
WHERE flights.distance <= aircraft.cruisingrange AND
aircraft.aid = certified.aircraft
AND certified.employee = employees.eid
GROUP BY employees.eid, employees.ename;
```

Find the name and salary of every non-pilot whose salary is more than the average salary for pilots:

```
SELECT employees.ename AS non_pilot_name,
employees.salary AS salary_more_than_pilots_average_salary
FROM employees, certified
WHERE employees.eid NOT IN (SELECT certified.employee
FROM certified) AND
employees.salary >
(SELECT AVG(pilotsalary.salary)
FROM
(SELECT DISTINCT employees.ename, employees.salary
FROM employees, certified
WHERE employees.eid = certified.employee
GROUP BY employees.ename, employees.salary) AS pilotsalary)
GROUP BY
employees.ename, employees.salary;
```

Find the names of employees who are certified only for aircrafts with cruising range longer than 1000 miles:

```
SELECT DISTINCT employees.ename AS employee_names
FROM employees, certified, aircraft
WHERE employees.eid = certified.employee AND
certified.aircraft = aircraft.aid AND
aircraft.cruisingrange > 1000;
```

Question 2

Q2a)(i) Canonical cover of the set R1:

We only look at the functional dependencies involving R1, which is:

$$AB \rightarrow C, AC \rightarrow B, BC \rightarrow A$$

To compute the canonical cover of the set R1, we have to check whether the RHS is all singleton attributes, which they are. Then we check whether if there is any extraneous attributes on the LHS for elimination, but there isn't any for this set of dependencies. Finally, we eliminate any redundant functional dependencies but there isn't any. Hence, the canonical cover turns out to be the set of dependencies from F projected on R1.

The canonical cover of the set of dependencies from F projected on R1 is

$$\{AB \rightarrow C, AC \rightarrow B, BC \rightarrow A\}$$

Q2a)(i) Canonical cover of the set R2:

Before we can compute the canonical cover of the set R2, we need to use functional dependencies that involve R2. To do that, we use the axioms of transitivity and augmentation to derive the functional dependencies that we need.

Based on the axiom of transitivity,

$$if\ AC \rightarrow B\ and\ B \rightarrow D,\ then\ AC \rightarrow D$$

Based on the axiom of augmentation,

$$if\ AC \rightarrow D,\ then\ ACA \rightarrow AD,\ which\ results\ in\ AC \rightarrow AD\ since\ ACA = AC$$

Based on the axiom of transitivity,

$$if\ AC \rightarrow AD\ and\ AD \rightarrow E,\ then\ AC \rightarrow E$$

Since there is only 1 functional dependency that can be derived from F which involves R2 and it contains a singleton RHS attribute, and there is no extraneous attributes on LHS or redundant functional dependencies, the canonical cover of the set of dependencies derived from F projected on R2 is

$$\{AC \rightarrow E\}$$

Q2a)(ii) Decomposition of set R1 into a collection of BCNF relations:

For subset R1 = ABC, it is in BCNF since AB, BC and AC are all superkeys. We know that they are all superkeys because the LHS of the functional dependencies of F which involve R1 can determine all its attributes:

$$AB \rightarrow C, AC \rightarrow B, BC \rightarrow A$$

$We\ know\ that\ AB \rightarrow C\ and\ based\ on\ the\ axiom\ of\ augmentation, we\ also\ know\ that:$

$$AB \rightarrow A$$
$$\big(if\ A \rightarrow A, then\ AB \rightarrow A\ for\ any\ B\big)$$

$$AB \rightarrow B$$
$$\big(if\ B \rightarrow B, then\ AB \rightarrow B\ for\ any\ A\big)$$

$We\ know\ that\ AC \rightarrow B\ and\ based\ on\ the\ axiom\ of\ augmentation, we\ also\ know\ that:$

$$AC \rightarrow A$$
$$\big(if\ A \rightarrow A, then\ AC \rightarrow A\ for\ any\ C\big)$$

$$AC \rightarrow C$$
$$\big(if\ C \rightarrow C, then\ AC \rightarrow C\ for\ any\ A\big)$$

$We\ know\ that\ BC \rightarrow A\ and\ based\ on\ the\ axiom\ of\ augmentation, we\ also\ know\ that:$

$$BC \rightarrow B$$
$$\big(if\ B \rightarrow B, then\ BC \rightarrow B\ for\ any\ C\big)$$

$$BC \rightarrow C$$
$$\big(if\ C \rightarrow C, then\ BC \rightarrow C\ for\ any\ B\big)$$

Q2a)(ii) Decomposition of set R2 into a collection of BCNF relations:

For subset R2 = ACEH, it is not in BCNF since the attribute H cannot be found in the functional dependencies of F. Therefore, we decompose ACEH into:

$$\{ACE\}$$

$$\{ACH\}$$

For ACE, it is in BCNF format because AC is a superkey that can determine all attributes.

We know that $AC \rightarrow E$ *from the canonical cover of* $R2$ *that we found earlier.*

Based on the axiom of augmentation, we also know that these are true:

$$AC \rightarrow A$$
$$\big(if\ A \rightarrow A, then\ AC \rightarrow A\ for\ any\ C\big)$$

$$AC \rightarrow C$$
$$\big(if\ C \rightarrow C, then\ AC \rightarrow C\ for\ any\ A\big)$$

For ACH, it is not in BCNF but cannot be decomposed further as there is no H attribute within the functional dependencies of F. Hence, R2=ACEH is decomposed into only ACE which is in BCNF. We ignore ACH as it is not in BCNF, since we are supposed to decompose R2 into a collection of BCNF relations.

Q2b) Consideration of the decompositions of R:

Dependency-preservation:

The decomposition is not dependency-preserving because one of the functional dependencies from F is not dependency-preserving:

$$E \rightarrow G$$

Based on the axiom of transitivity,

$$if\ AD \rightarrow E\ and\ E \rightarrow G,\ then\ AD \rightarrow G$$

We now look at all the functional dependencies including this derived one. We will know whether a functional dependency is dependency-preserving by checking whether the attributes on the LHS and the RHS collectively are contained in either of the decomposed relation sub-schemas. In this case, the functional dependency

$$E \rightarrow G$$

is not collectively contained in either of the relation sub-schemas. E-closure is also not able to determine anything else, as this functional dependency is the only one with E on the LHS. Hence, this renders the whole decomposition not dependency-preserving. Although the rest are, but as long as one is not dependency-preserving, the overall decomposition is considered as not dependency-preserving.

Loseless-join:

Let relation sub-schemas be R1 = (ABCDE) and R2 = (ADG), a = distinguished variable and b = undistinguished variable.

We create a matrix with the same number of columns as the relation, and put distinguished variables in the columns for each relation sub-schema where the attribute exists. Then we add distinguished variables based on the rules for adding distinguished variables, changing "b" to "a" where we add them until we get a row of purely "a"s in either R1 or R2 while looping through the functional dependencies. A full row of "a"s of distinguished variables will mean that the decomposition is lossless-join.

Before adding distinguished variables:

	A	B	C	D	E	G
R1	a	a	a	a	a	b
R2	a	b	b	a	b	a

After adding distinguished variables (went through all the functional dependencies once, meaning 1 loop):

	A	B	C	D	E	G
R1	a	a	a	a	a	a
R2	a	b	b	a	a	a

The decomposition is considered lossless-join because when we add distinguished variable "a" to the 2 relation sub-schemas as mentioned above, R1 will eventually end up with a full row of distinguished variables. Hence, we know that the decomposition is lossless-join.

Question 3: MRU buffer replacement policy

In MRU (Most Recently Used) buffer replacement, the idea is to replace the page slot (or block) in memory which is used (referred) very recently. The assumption is that if the page slot (block) is used very recently, it might not be used again in the near future.

A scenario that leads to the worst-case performance of the MRU buffer replacement policy is when we alternate between a small set of pages as demonstrated in the scenario below.

Consider the scenario where there is this sequence of page slot (or block) accesses (A, B, C, D, D+1, D, D+1, D, C). Assume that the limit of page slots (blocks) for the buffer pool is 4. When all its 4 frames contain pages/blocks (no free frames), then a block needs to be removed through the `release_block` function and then replaced with the new block. (The `release_block` function indicates that the block is no longer in use and is a good candidate for removal.)

The Most Recently Used (MRU) buffer replacement algorithm/policy will kick out block (D) to make room for block (D+1), then kick out block (D+1) to make room for block (D), and so on according to the stated sequence.

For example, after kicking out block (D) to make room for block (D+1), the next block in the sequence is block (D) again but since it is not already in the buffer pool, there is a cache miss and there will be a need to read from the hard disk to a free frame, and if there is no free frame, a block needs to be removed. If the block is already in the buffer pool, then there is no need to read it again and we can use the copy there unless it is write-locked.

It will have a cache miss on almost every block look-up for this sequence of files, since the a block that has been previously opened needs to be accessed again but the MRU algorithm has already trashed it from its memory.

Most Recently Used will see (N) cache misses, making it (N / (D+1))-competitive. Since we can make (N) as large as we want, this can be arbitrarily large – we might call the Most Recently Used algorithm (infinity)-competitive.