CSE Midterm - Spring 2017 *Solutions*

March 28, 2017

Question	Points Possible	Points Earned
A.1	10	
A.2	10	
A.3	10	
A	30	
B.1	10	
B.2	25	
B.3	10	
B.4	5	
B	50	
C	20	
Total	100	

Extended Relational Algebra Operator Reference

Select	$\sigma_c(R)$	c : The selection condition
Extended Project	$\pi_{e_1,e_2,\ldots}(R)$	e_i : The column or expression to project
Product	$R_1 \times R_2$	
Join	$R_1 \bowtie_c R_2$	c : the join condition
Distinct	$\delta(R)$	
Group	$\gamma_{gb_1,gb_2,\ldots,\texttt{AGG}(e_1),\ldots}(R)$	gb_i : group by columns, e_i : expression
Set Difference	$R_1 - R_2$	
Union	$R_1 \cup R_2$	
Sort	τ_A	A one or more attributes to sort on

Relational Algebra Equivalences

Rule	**Notes**
$\sigma_{C_1 \wedge C_2}(R) \equiv \sigma_{C_1}(\sigma_{C_2}(R))$	
$\sigma_{C_1 \vee C_2}(R) \equiv \sigma_{C_1}(R) \cup \sigma_{C_2}(R)$	Note, this is only true for set, not bag union
$\sigma_C(R \times S) \equiv R \bowtie_C S$	
$\sigma_C(R \times S) \equiv \sigma_C(R) \times S$	If C references only R's attributes, also works for joins
$\pi_A(\pi_{A \cup B}(R)) \equiv \pi_A(R)$	
$\sigma_C(\pi_A(R)) \equiv \pi_A(\sigma_C(R))$	If A contains all of the attributes referenced by C
$\pi_{A \cup B}(R \times S) \equiv \pi_A(R) \times \pi_B(S)$	Where A (resp., B) contains attributes in R (resp., S)
$R \times (S \times T) \equiv (R \times S) \times T$	Also works for joins
$R \times S \equiv S \times R$	Also works for joins
$R \cup (S \cup T) \equiv (R \cup S) \cup T$	Also works for intersection and bag-union
$R \cup S \equiv S \cup R$	Also works for intersections and bag-union
$\sigma_C(R \cup S) \equiv \sigma_C(R) \cup \sigma_C(S)$	Also works for intersections and bag-union
$\pi_A(R \cup S) \equiv \pi_A(R) \cup \pi_A(S)$	Also works for intersections and bag-union
$\sigma_C(\gamma_{A,AGG}(R)) \equiv \gamma_{A,AGG}(\sigma_C(R))$	If A contains all of the attributes referenced by C

Question A: Relational Algebra
(30 points)

Consider the following database schema, listing information for a supply-chain company. The `CITY` table lists information about cities with the company's warehouses, while the `ROUTES` table lists established routes between warehouses. Listed routes form a directed-acyclic graph (i.e., following edges will always lead you to a leaf). For each query listed, provide a relational algebra tree.

```
CREATE TABLE WAREHOUSE( id int, name string, capacity int );
CREATE TABLE ROUTES( origin_warehouse int, dest_warehouse int, volume float );
```

1. **(10 pt)** Find the total shipping volume of all routes originating in a warehouse with a smaller capacity than the destination.

$$\gamma_{SUM(volume)}(\sigma_{origin.capacity<dest.capacity}((R \bowtie_{origin} W) \bowtie_{dest} W))$$

2. **(10 pt)** Find the name of every warehouse that receives shipments exclusively from warehouses with a smaller capacity.

$$A := \delta(\pi_{origin.id}(\sigma_{origin.capacity \geq dest.capacity}((R \bowtie_{origin} W) \bowtie_{dest} W)))$$
$$B := (\pi_{id}(W) - A)$$
$$\pi_{name}(B \bowtie_{id} W)$$

   Partial credit was awarded for answers that returned the names of warehouses that received any shipments from a warehouse with a smaller capacity (as opposed to **exclusively** from such warehouses).

3. **(10 pt)** A spanning tree for a directed acyclic graph is a subset of the edges such that every node has exactly one parent. Although many graph algorithms can not be expressed in RA, this one can. Compute a spanning tree for `ROUTES`, with source as the parent.

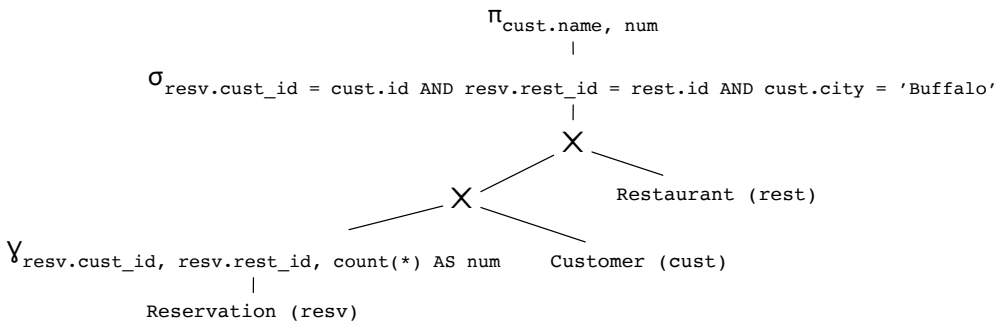
   To construct a spanning tree from any DAG, you need to pick one (arbitrary) parent for each node. In this case, a simple approach is to use MIN or MAX, although many relational databases also support FIRST, LAST, and NTH aggregates.

$$\gamma_{dest_warehouse, MIN(origin_warehouse)}(ROUTES)$$

   Answers fell into 4 categories. Several students proposed the exact right answer as above (or something relatively close). Several students proposed solutions based on extending RA with some form of recursion. Though RA does not support recursion, this solution earned 8 points. Anyone who recognized and clearly described the need for some sort of aggregate received 5 points.

Question B: Query Optimization
(50 points)

Questions in this section ask you optimize the following query:



1. **(10 pt)** As we discussed in class, the first stage of optimizing a query takes place entirely in relational-algebra land. The optimizer picks out a set of candidate expressions that could, under some assumptions, generate an optimal plan. Using only the rewrite rules listed at the front of this exam, show how of any one of the potentially optimal expressions is derived.

$$\pi_{name,num}(\sigma_{cust \wedge rest \wedge city}((\gamma(Reservation) \times Customer) \times Restaurant)) \quad (1)$$
$$\pi_{name,num}(\sigma_{rest}((\sigma_{cust \wedge city}(\gamma(Reservation) \times Customer)) \times Restaurant)) \quad (2)$$
$$\pi_{name,num}((\sigma_{cust \wedge city}(\gamma(Reservation) \times Customer)) \bowtie_{rest} Restaurant) \quad (3)$$
$$\pi_{name,num}((\sigma_{cust}(\gamma(Reservation) \times \sigma_{city}(Customer))) \bowtie_{rest} Restaurant) \quad (4)$$
$$\pi_{name,num}((\gamma(Reservation) \bowtie_{cust} \sigma_{city}(Customer)) \bowtie_{rest} Restaurant) \quad (5)$$

A few other optimizations were also possible. Several students applied projection pushdown and/or join reordering at this stage, both of which can produce more efficient plans under certain conditions. Reasons for points being taken off include:

- Not converting $\sigma(R \times S)$s into joins: -1 pt
- Pulling up aggregates (Not a correct optimization in general, and not one of the allowed rewrites): -1pt
- Non-recursive optimization (e.g., only applying rewrites to the top-level): -5 pt
- Only answer, no justification: -6 pt

2. **(25 pt)** The database has gathered the statistics below,

Table	# of Rows	Primary Index
Reservation	20,000,000	$\langle cust\_id, rest\_id \rangle$
Customer	1,000,000	$\langle id \rangle$
Restaurant	10,000	$\langle id \rangle$

Field	# of Distinct Values
resv.rest_id	10,000
resv.cust_id	1,000,000
cust.id	1,000,000
cust.city	20
cust.name	800,000
rest.id	10,000

There are no secondary indexes built and no other information is available to the optimizer. Assume that enough memory is available for slightly more than 5,000 tuples (say 5,100 tuples), independent of relation. Draw the optimal query evaluation plan according to these statistics. Be sure to indicate which algorithm is used for which (combination of) operator(s)!

A number of plans were possible. Of all of the possible join orders, $(Reservation \bowtie Customer) \bowtie Restaurant$ produces the fewest intermediate tuples. Because there is no index on $Customer.city$, a full table scan is required for this table. We were looking for three algorithm choices: one for the aggregate on reservation and two for the two joins:

(a) **Aggregate of Reservation**: The optimal plan is one in which the optimizer identifies that Reservation already has a primary index on the two group-by columns, and is thus already sorted in the right order. Thus, the aggregate may be computed in a single scan.

(b) **Join with Customer**: The optimal plan is one in which the optimizer identifies that Reservation and Customer both have primary indexes that start with customer id, making it possible to perform sort/merge join (except without needing to sort). Partial credit was also given for 2-pass hash or index-nested loop join. Block-nested loop and nested loop joins are entirely inappropriate here, as they have N^2 performance on an equality-join. 1-pass Hash is entirely inappropriate here, as it requires more memory than is available.

(c) **Join with Restaurant**: The optimal plan (depending on your assumptions about the cost of an index lookup) was either an Index-Nested-Loop or a 2-pass hash join. An index already exists on restaurant, making INLJ feasible. Conversely, we can expect 10^6 tuples from the earlier pipeline stages, and we will need to do exactly that many index lookups. Depending on your assumptions about how much this costs, it may be safer to do a 2-pass hash join. Sort-merge join is inappropriate as it would require sorting 10^6 tuples. As before, block-nested loop, nested loop, and 1-pass hash are all inappropriate here.

- 10 points for recognizing that the Reservation and Customer tables are sorted and using SMJ
- 10 points for using a valid join algorithm for $R \bowtie Restaurant$
- 5 Points for using a constant time γ

3. **(10 pt)** What is the plan's total IO cost in terms of *total number of* ***tuples*** *read or written*?

(a) All tuples in Reservation are required ($2 \cdot 10^7$ tuples read)

(b) To estimate $|\gamma resv|$ we would normally use the domains of the two group-by columns:

$$|\gamma resv|_1 = |rest\_id| \cdot |cust\_id| = 10^4 \cdot 10^6 = 10^10$$

However, there is also a hard upper bound based in the input to the group-by: $|\gamma resv| = |resv| = 2 \cdot 10^7$. The latter is lower, so clearly not all of the groups will be populated. However, the aggregate will not need to perform any IOs.

(c) All tuples in Cust are required (10^6 tuples read)

(d) We can estimate selectivity on the customer table is $\frac{1}{20}$ based on the number of tuples in it.

(e) Sort-free SMJ does not require new IOs.

(f) The final join, implemented as a 2-pass hash join requires us to write out all of the tuples on both sides, then read them back in in groups. For the right-hand side, this is the entire Restaurant table (10^4 tuples written, 10^4 tuples read). For the left hand side, we need to estimate the number of tuples resulting from the join. Naively, we can do this by using the lower number of distinct values in the join attributes. Both tables have 10^6 values of the customer id attribute, so regardless of which direction we go in, we'll get $2 \cdot 10^7$ tuples in the output (this number both written, and then read). Since the statistics indicate that cust.id is unique, we can be a little more clever and observe that the selection predicate is going to filter out all but 5% of these unique values, leading to $1 \cdot 10^6$ tuples in the intermediate result.

- Full credit was given as long as the answer correctly analyzes the evaluation plan in part 2.
- Partial credit was given if the analysis was almost correct

4. **(5 pt)** As suggested by the value statistics, `rest.id` is a key (each value is unique). Considering this, and the fact that no attribute of restaurants is used in the query, why is it unsafe to simply drop the join with restaurants?

   If rest.id is a foreign key, then it's safe to drop the join with restaurants. If not, then there would be some rest.id not in the restaurant table.

Question C: Algorithms
(20 points)

Using Big-O notation, what is the working set size of each of the following algorithms/operators in terms of number of tuples. Be sure to explicitly indicate any shorthands, assumptions, or unspecified parameters that you use.

#	Algorithm	Rel. Alg.	Working set size
1	Projection (Map)	$\pi_A R$	$O(1)$
2	Selection (Filter)	$\sigma_\phi R$	$O(1)$
3	Union	$R \cup S$	$O(1)$
4	Nested Loop Join	$R \bowtie_\phi S$	$O(1)$
5	Block-Nested Loop Join	$R \bowtie_\phi S$	$O(\|B\|)$
6	1-Pass Hash Join	$R \bowtie_A S$	$O(\|R\|)$ or $O(\|S\|)$
7	Sort-Merge Join (input sorted on A)	$R \bowtie_A S$	$O(1)$
8	2-Pass Sort	$\tau_A R$	$O(\|B\|)$
9	Group-By Aggregate (unsorted input)	$\gamma_{A,SUM(B)}$	$O(\|A\|)$ or $O(\|H\|)$
10	Group-By Aggregate (input sorted on A)	$\gamma_{A,SUM(B)}$	O(1)

Notes above use:

- $|B|$: Block Size
- $|H|$: Hash Bucket Size
- $|R|, |S|$: Size of R, S
- $|A|$: Size of the domain of attribute A