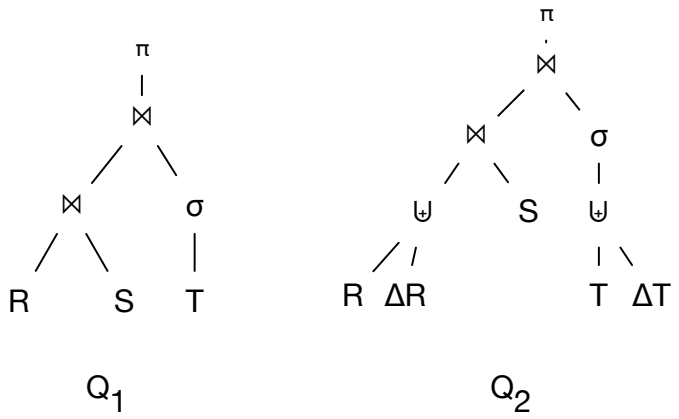
CSE 562 Final *Solutions*

May 14, 2015

Question	Points Possible	Points Earned
A	20	
B	20	
C	20	
D	20	
E	20	
Total	100	

Question A: Relational Algebra
(20 points)

Consider the following two Bag Relational Algebra expressions ($\uplus$ denotes Bag Union): Q_1 and Q_2. The two queries are identical, except that Q_2 contains additional rows in the relations R and T, represented by the relations ΔR and ΔT respectively.



Part 1 (15 pts): Assume that the result for Q_1 is materialized and available, suggesting that incremental view maintenance may be ideal for this query. Recall that in incremental view maintenance, our goal is to construct a delta query $\Delta Q_{1\to 2}$ such that $Q_2 \equiv Q_1 \uplus \Delta Q_{1\to 2}$, allowing us to re-use the already available Q_1. Using the set of relational algebra equivalencies given on the reference page, rewrite Q_2 into the form $Q_1 \uplus \Delta Q_{1\to 2}$ and clearly indicate which part corresponds to $\Delta Q_{1\to 2}$

Distributivity of union over join:

$$(R \uplus \Delta R) \bowtie S \equiv (R \bowtie S) \uplus (\Delta R \bowtie S)$$

Let's refer to the above query as Q_{RS}. Union commutes through selection:

$$\sigma(T \uplus \Delta T) \equiv (\sigma T) \uplus (\sigma \Delta T)$$

Apply distributivity again:

$$Q_{RS} \bowtie ((\sigma T) \uplus (\sigma \Delta T)) \equiv (Q_{RS} \bowtie \sigma T) \uplus (Q_{RS} \bowtie \sigma \Delta T)$$

Distributivity once more:

$$(R \bowtie S) \uplus (\Delta R \bowtie S) \bowtie \sigma T \equiv ((R \bowtie S) \bowtie \sigma T) \uplus ((\Delta R \bowtie S) \bowtie \sigma T)$$

And similarly for ΔT, resulting in:

$$\pi(((R \bowtie S) \bowtie \sigma T) \uplus ((\Delta R \bowtie S) \bowtie \sigma T) \uplus ((R \bowtie S) \bowtie \sigma \Delta T) \uplus ((\Delta R \bowtie S) \bowtie \sigma \Delta T))$$

Union commutes through projection:

$$(\pi((R \bowtie S) \bowtie \sigma T)) \uplus \pi(((\Delta R \bowtie S) \bowtie \sigma T) \uplus ((R \bowtie S) \bowtie \sigma \Delta T) \uplus ((\Delta R \bowtie S) \bowtie \sigma \Delta T))$$

Which means that $\Delta Q_{1\to 2}$ is:

$$\pi(((\Delta R \bowtie S) \bowtie \sigma T) \uplus ((R \bowtie S) \bowtie \sigma \Delta T) \uplus ((\Delta R \bowtie S) \bowtie \sigma \Delta T))$$

Part 2 (5 pts): Recall that the performance benefit of this 'delta-form' query is a result of the delta query being much cheaper to evaluate than the original query. That is, constructing the output of Q_2 has a cost of $O(|Q_1| + \texttt{cost}(\Delta Q_{1\to 2}))$: A linear scan over the pre-materialized results Q_1 and an evaluation of the delta query.

Consider what happens when we define two new queries Q_1', Q_2' by replacing the projection π in Q_1, Q_2 (respectively) with the following group-by aggregate:

$$\gamma_{R.A,S.B,T.C,\texttt{SUM}(T.D),\texttt{MAX}(T.E)}$$

It is not possible to rewrite Q_2' into a form $Q_1' \uplus \Delta Q_{1\to 2}'$. However, it's still possible to rewrite the query to obtain a similar performance benefit. Provide a relational algebra query (as a plan or an RA expression) that explicitly uses the relation Q_1' to reduce the cost of evaluating Q_2' in the same way as above.

Label the following expression $\Delta Q_{1\to 2}'$

$$((\Delta R \bowtie S) \bowtie \sigma T) \uplus ((R \bowtie S) \bowtie \sigma \Delta T) \uplus ((\Delta R \bowtie S) \bowtie \sigma \Delta T)$$

As they are almost identical, $\texttt{cost}(\Delta Q_{1\to 2}') \approx \texttt{cost}(\Delta Q_{1\to 2})$, and the following query is efficient:

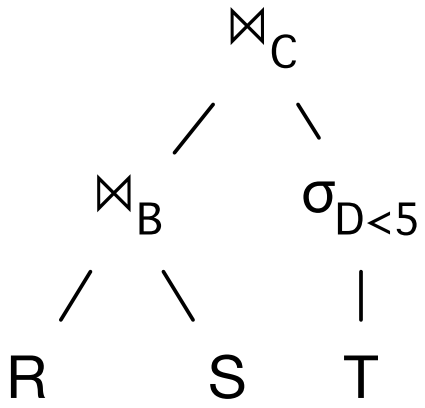
$$\gamma_{R.A,S.B,T.C,\texttt{SUM}(T.D),\texttt{MAX}(T.E)}(Q_1' \uplus \Delta Q_{1\to 2}'))$$

Question B: Cost-Based Optimization
(20 points)

Relation R has schema (A, B), and contains 100 rows.
Relation S has schema (B, C), and contains 200 rows.
Relation T has schema (C, D), and contains 400 rows.

Assume a uniform distribution for all values.
Attribute A is a primary key for A and has 100 distinct values.
Attribute B has 50 distinct values (in both R and S).
Attribute C has 20 distinct values (in both S and T).
Attribute D has values in the range $0 \leq D < 10$ in T.

Now consider the following query plan:



Part 1 (5 pts): How many tuples are emitted by $\bowtie_B$, above.
The RF for an equijoin on B is $\frac{1}{50}$, out of a total of 100×200 tuples, or 400 tuples in total.
Part 2 (5 pts): How many tuples are emitted by $\sigma_{D<5}$, above.
The predicate covers half of the range of values in D. Thus, the reduction factor is $\frac{1}{2}$, or 200 tuples.
Part 2 (10 pts): How many tuples are emitted by $\bowtie_C$, above.
The RF for an equijoin on C is $\frac{1}{20}$, so the overall RF is:

$$\frac{1}{50} \times \frac{1}{2} \times \frac{1}{20}$$

$$\#tuples = \frac{1}{50} \times \frac{1}{2} \times \frac{1}{20} \times 100 \times 200 \times 400 = 4000$$

Question C: Transactions
(20 points)

Each part of this question provides a sequence of read, write, and commit *arrivals*. For each sequence of arriving events, show the final schedule that would result under each of the following concurrency protocols. Do not replay aborted transactions.

If an operation would block (without being aborted), it and all subsequent operations in the same transaction are delayed until the blocking locks are released.

Assume the following: The Timestamp CC protocol uses transaction ID numbers as timestamps. Validation-Based CC assigns timestamps in the order in which commits occur. Periodic Deadlock Detection prefers to kill younger transactions.

For lock-based protocols, show all lock acquires and releases. Indicate all transactions that are aborted by the concurrency protocol.

- Strict 2PL with Periodic Deadlock Detection
- Validation-Based Optimistic CC
- Timestamp CC without Versioning
- Timestamp CC with Versioning

Part 1 (10 pts):

```
1-8:  T1: R(X), T2: W(X), T1: W(Y), T2: R(Y), T3: R(Z), T1: W(Z), T3: W(X), T1: COMMIT,
9-12: T2: W(Y), T3: W(Z), T2: COMMIT, T3: COMMIT
```

S = Acquire Shared, X = Acquire Exclusive, U = Unlock
2PL: 1:S(X), 1:R(X), 2:X(X), 1:X(Y), 1:W(Y), 3: S(Z), 3: R(Z), 1:X(Z); DEADLOCK; 3: ABORT, 1:W(Z), 1: COMMIT, 1:U(X, Y, Z), 2: W(X), 2:S(Y), 2:R(Y), 2:X(Y), 2:W(Y), 2:COMMIT, 2:U(X,Y)
OCC: ops 1-7 as arrived, 1: COMMIT, 2:W(Y), 3:W(Z), 2:COMMIT, 3:ABORT (invalid read on Y)
TSCC: ops 1-3 as arrived, 2: ABORT (invalid read on Y), 3:R(Z), 1: ABORT (invalid write on Z), 3:W(X), 3:W(Z), 3: COMMIT
MVTSCC: ops 1-5 as arrived, 1: ABORT (invalid write on Z), 3:W(X), 2:W(Y), 3:W(Z), 2: COMMIT, 3: COMMIT

Part 2 (10 pts):

```
1-8:  T2: W(Y), T1: W(X), T2: W(X), T3: W(X), T1: R(Y), T2: R(X), T1: COMMIT, T2: COMMIT,
9-10: T3: W(Y), T3: COMMIT
```

2PL: 2:X(Y), 2:W(Y), 1:X(X), 1:W(X), 2:X(X), 3:X(X), 1:S(Y); DEADLOCK; 2:ABORT, 2:U(Y); 1: R(Y), 1: COMMIT, 1:U(X,Y), 3:W(X), 3:X(Y), 3:W(Y), 3: COMMIT, 3: U(X,Y)
OCC: ops 1-6 as arrived, 1: ABORT (Invalid read on Y), 2: ABORT (Invalid read on X), 3:W(Y), 3: COMMIT
TSCC: ops 1-4 as arrived, 1: ABORT (Invalid read on Y), 2: ABORT (Invalid read on X), 3:W(Y), 3: COMMIT
MVTSCC: ops 1-4 as arrived, 1: R(Y), 2: R(Y), 1: COMMIT, 2: COMMIT, 3:W(Y), 3: COMMIT

Question D: ARIES & Logging
(20 points)

A database recovers after a crash with the following log file, with `#: TN-W(PX)` denoting log entry `#` recording a write to page `PX` from Transaction `TN`.

```
1: T1-W(P1), 2: T1-W(P4), 3: T1-W(P2), 4: T4-W(P4), 5: T2-W(P4), 6: T1-W(P5), 7: T4-W(P1),
8: T1-COMMIT, 9: T4-W(P4)
```

Meta-data for data pages safely flushed to disk is as follows:

Page	1	2	3	4	5
Last Write	7	3	0	2	0

The log begins from the most recently completed checkpoint. Data pages 1-5 loaded from disk were last written to at timestamps: 7, 3, 0, 2, and 0 respectively. The checkpointed transaction table has only a single transaction T0 with no prior writes.

Part 1 (15 pts): Go through the ARIES recovery protocol step-by-step, clearly indicating every modification to a data page, the transaction table, and the last-written marker on each data page.
Analysis begins with the checkpoint at timestamp 0.

1. Add T1 to transaction table with Last Write of LSN1
2. Update T1 Last Write to LSN2
3. Update T1 Last Write to LSN3
4. Add T4 to transaction table with Last Write of LSN4
5. Add T2 to transaction table with Last Write of LSN5
6. Update T1 Last Write to LSN6
7. Update T4 Last Write to LSN7
8. Remove T1 from transaction table.
9. Update T4 Last Write to LSN9

The earliest timestamp is 0. Redo begins at timestamp 1.

1. Replay Write to P1; Update P1 Last Write to LSN1
2. Replay Write to P4; Update P4 Last Write to LSN2
3. Replay Write to P2; Update P2 Last Write to LSN3
4. Replay Write to P4; Update P4 Last Write to LSN4
5. Replay Write to P4; Update P4 Last Write to LSN5
6. Replay Write to P5; Update P5 Last Write to LSN6
7. Replay Write to P1; Update P1 Last Write to LSN7

8. no-op
9. Replay Write to P4; Update P4 Last Write to LSN9

Final Transaction Table:

Transaction	Last Write
T0	n/a
T2	LSN5
T4	LSN9

Undo begins by removing T0 from the transaction table (Read only transaction). Then

- Append T2-BEGIN ABORT; Appended operation has LSN10
- Append T4-BEGIN ABORT; Appended operation has LSN11
- Append CLR for LSN9; Appended operation has LSN12. Update T4 Last Write to LSN7.
- Append CLR for LSN7; Appended operation has LSN13. Update T4 Last Write to LSN4
- Append CLR for LSN5; Appended operation has LSN14. Update T2 Last Write to n/a.
- Append T2-END ABORT; Appended operation has LSN15. Remove T2 from transaction table.
- Append CLR for LSN4; Appended operation has LSN16. Update T4 Last Write to n/a.
- Append T4-END ABORT; Appended operation has LSN17.

Part 2 (5 pts): Assume that the database crashes immediately after the REDO phase of ARIES, and that the buffer pool has not had a chance to flush any of the dirty pages to disk before the crash. Describe the recovery process from this second crash. You may answer this part in terms of your solution to Part 1.
Nothing on-disk has changed. The recovery process is identical to Part 1.

Question E: Big Data
(20 points)

Answer True/False:

1. A star schema consists of a fact table and numerous dimension tables. [[TRUE]] / false
2. An advantage of column-wise storage is a reduced cost to access wide tables. [[TRUE]] / false
3. In a Semi-Join, the amount of data sent by the node evaluating the join is linear in the number of distinct values of the join attribute. [[TRUE]] / false
4. In a Bloom-Join, the amount of data sent by the node evaluating the join is linear in the number of distinct values of the join attribute. [[TRUE]] / false
5. For a **selection** operator implemented in parallel, doubling the size of the source relation and also doubling the number of processors results in **no** change to the the processing time. [[TRUE]] / false
6. For a **projection** operator implemented in parallel, doubling the size of the source relation and also doubling the number of processors results in **no** change to the the processing time. [[TRUE]] / false
7. For a **sort** operator implemented in parallel, doubling the size of the source relation and also doubling the number of processors results in **no** change to the the processing time. true / [[FALSE]]
8. For a **join** operator implemented in parallel, doubling the size of both source relations and also doubling the number of processors results in **no** change to the the processing time. true / [[FALSE]]
9. For a **group-by aggregate** operator implemented in parallel, doubling the size of both source relations and also doubling the number of processors results in **no** change to the the processing time. true / [[FALSE]]
10. Range (as opposed to hash) partitioning is guaranteed to produce uniformly distributed partition buckets. true / [[FALSE]]

Grading Details

Part A.1 (Grader: Oliver): Full credit was given if the correct result appeared at the end of the solution. Partial credit was given as follows:

- Solutions that treated union as *commutative* with join rather than *distributive* over join were given 6 points.
- Partial solutions that brought the Δ terms up but didn't pull them out of the top level join were given 10 points.
- 1-2 extra points were awarded at the grader's discretion.

Part A.2 (Grader: Oliver): Full credit was given if the result clearly identified the possibility of pushing algebraic aggregates up through the query tree rather than re-computing them from scratch. Partial credit was awarded for recognizing that such a push could happen. A 1-3 point penalty was applied for illegible solutions.

Relational Algebra Operator Reference

Selection	$\sigma_c(R)$	c : The selection condition
Projection	$\pi_{e_1,e_2,\ldots}(R)$	e_i : The column or expression to project
Cartesian Product	$R_1 \times R_2$	
Join	$R_1 \bowtie_c R_2$	c : the join condition
Aggregate	$\pi_{gb_1,gb_2,\ldots,\texttt{SUM}(e_1),\ldots}(R)$	gb_i : group by columns, e_i : expression
Set Difference	$R_1 - R_2$	
Union	$R_1 \cup R_2$	

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 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