CS 186/286 Fall 2017 Final Exam

- Do not turn this page until instructed to start the exam.
- You should receive 1 *answer sheet* and a 25-page *exam packet*.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- You have *3 hours* to complete the final.
- The midterm has *8 questions*, each with multiple parts.
- For each question, place only your *final answer* on the answer sheet; do not show work.
- For multiple choice questions, please fill in the bubble or box completely, **do not mark the box with an X or checkmark**.
- Use the blank spaces in your exam for scratch paper.
- You are allowed **two** 8.5" × 11" double-sided pages of notes.
- No electronic devices are allowed.

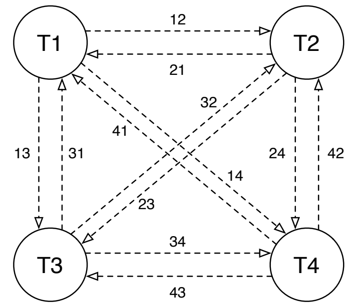
1 Concurrency

Consider the following schedule of reads and writes to pages.

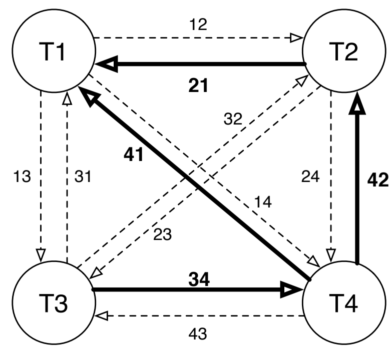
	1	2	3	4	5	6	7	8	9	10
T_1	R(A)						W(A)	R(C)		
T_2		R(A)							R(C)	
T_3				R(C)	R(B)					W(B)
T_4			R(C)			W(C)				

Conflicts

1. (2 points) What edges are included in the **dependency graph** (*not* waits-for graph) for the above schedule? A reference graph is provided for your convenience. The dotted edges represent all *possible* edges; each edge is labeled with the transaction IDs of the nodes it connects, in order. E.g. the edge from $T_4 \rightarrow T_1$ is labeled 41. Mark the answer sheet with the labels of the edges that are in the dependency graph of the schedule above.



Solution:



Rubric: All-or-nothing.

21, 34, 41, 42. 21 comes from the RW conflict at time steps 2 and 7. I comes from the RW conflict at time steps 4 and 6. J comes from the WR conflict at time steps 6 and 8. K comes from the WR conflict at time steps 6 and 9.

The "backwards" answer is also acceptable: 12, 43, 14, 24.

2. (1 point) Which of the following serial schedules are conflict equivalent to the schedule above?

   A. T_4, T_2, T_1, T_3

B. T_3, T_2, T_4, T_1

C. T_3, T_4, T_2, T_1

D. T_1, T_2, T_4, T_3

E. T_1, T_4, T_2, T_3

F. None of the above

Solution: C. The graph is acyclic, so it is conflict serializable. Topologically sorting the above graph gives this ordering.

3. (2.5 points) Mark all statements that are true.

A. In Strict 2PL, we can give up locks after aborting but before rollback is complete.

B. Some conflict serializable schedules cannot be produced when using 2PL.

C. Schedules that are conflict serializable will not produce a cyclic dependency graph.

D. Both Strict 2PL and 2PL enforce conflict serializability.

E. All schedules that are conflict serializable are view serializable.

Solution: Only A is false. For A, you must wait until rollback is complete before giving up locks. For B, 2PL enforces conflict serializability but may not allow all conflict serializable schedules (e.g. W1(X), R2(X), W1(Y), R2(Y) is impossible under 2PL). For C, a schedule is conflict serializable if and only if the dependency graph is acyclic, so conflict serializable implies an acyclic dependency graph. For D, because you cannot get any new locks after releasing a lock in both strict and non-strict 2PL, the dependency graph for the resulting schedule can never be cyclic. For E, view serializability is a generalization of conflict serializability, hence all conflict serializable schedules are view serializable.

Dealing with Deadlock

For the following sequence of lock requests, assume T_1 is older than T_2, and T_2 is older than T_3. The oldest transaction has the highest priority. No locks are released in the time frame shown. Note that not all requests may actually be issued—in some cases the transaction is still blocked waiting for a prior request.

Consider the following schedule of lock requests:

	1	2	3	4	5	6	7	8	9
T1	X(D)		S(A)			S(C)			
T2		X(A)		S(B)				S(D)	
T3					S(B)		X(B)		S(C)

4. (1 point) The above schedule could not successfully occur whether one uses deadlock avoidance or not. Suppose we are not using any deadlock avoidance algorithms. List the time steps in the schedule above in which *the lock request could not have been made*: i.e. timesteps when the requesting transaction would have been blocked waiting for a previous lock request.

Solution: Rubric: All-or-Nothing. Answer: 6 and 9. Transaction 1 is blocked at time step 3 because it is not granted the shared lock on A, since Transaction 2 already has an exclusive lock on it, so it cannot make the request at time step 6. Transaction 3 is blocked at time step 7 because it is not granted its lock upgrade, since there is another transaction that also has a shared lock on B.

5. (1 point) Assuming we do not use deadlock avoidance, what is the first time step when deadlock will occur?

Solution: 8. When T2 requests this lock, it is waiting on T1, which is blocked, so the transactions enter deadlock.

6. (1 point) Mark the transactions involved in the first deadlock.

Solution: T1 and T2. The cycle in the waits-for graph only goes between these two transactions. All or nothing for credit.

7. (1 point) If we were using the wound-wait deadlock avoidance algorithm, what would happen at time step 3?

   A. The transaction requesting the lock would wait

   B. The transaction requesting the lock would abort

   C. The transaction that holds the lock would abort

Solution: C. T1 is requesting the lock T2 holds and it has higher priority, so T2 would abort.

2 Recovery

1. (0.5 points) Which buffer management policy would allow us to maximize efficiency in terms of speed?

   A. STEAL, NO FORCE

   B. STEAL, FORCE

   C. NO STEAL, FORCE

   D. NO STEAL, NO FORCE

   Solution: A. STEAL allows us to re-use the frames in the buffer pool more efficiently, and NO FORCE avoids unnecessary disk flushes. Together, these improve the efficiency of the buffer manager at the cost of requiring more complex recover algorithms.

   This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

2. (0.5 points) True or False? In ARIES recovery, after the analysis phase, the recLSN of each page in the dirty page table must be larger than the pageLSN of the corresponding page.

   Solution: False. The page could have been updated and flushed from the buffer pool between the last checkpoint and time of crash. The flushed page would have a pageLSN from its *most recent update*, which is after the recLSN in the checkpoint.

   This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

3. (2 points) Select all the correct statements that are true about ARIES.

   A. CLRs are never undone during ARIES recovery.

   B. We remove a page from the dirty page table each time we write a CLR to the log.

   C. The transaction table and dirty page table in the End Checkpoint record are up-to-date as of the time when the End Checkpoint record was logged.

   D. ARIES never causes a database disk page to be overwritten by a page with the same PageLSN.

   Solution: A is true. CLRs are re-done but never undone. B is false. CLRs do not require us to flush a page in the dirty page table to disk. In fact, at no point in ARIES are we forced to flush a page in the dirty page table to disk. C is false. The transaction table and dirty page table in the End Checkpoint record are up-to-date as of the time where the *Begin Checkpoint* record was logged. D is true. A page on disk is only ever overwritten with a more recent version of the page, and if the page is more recent, it has a larger PageLSN.

   This question was graded as 4 independent true/false questions, each worth 0.5 points. That is, you got 0.5 points for every correct choice that you selected and 0.5 points for every incorrect choice that you did *not* select.

4. (0.5 points) True or False: Regardless of the number of times we run recovery, each UPDATE in the log will have at most one matching CLR record in the log.

Solution: True. ARIES never issues multiple CLRs for the same UPDATE twice. Instead, the redo phase redoes CLR log entries.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

5. (1 point) Why would the undoNextLSN field in a CLR be marked null?

   A. If the transaction committed.

   B. If this CLR corresponds to the lowest UPDATE LSN of the transaction

   C. If this CLR corresponds to the highest UPDATE LSN of the transaction

   D. If this CLR was the highest LSN flushed at the time of crash.

   E. If this CLR corresponds to a page that was flushed before the checkpoint.

Solution: The undoNextLSN field of a CLR is the LSN of the next record that is to be undone. When there are no records left to be undone, the field is left null. Thus, B is correct and all other answers are not.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

Now consider the following log records which are recovered after a crash.

LSN	**Record**	**PrevLSN**
⋮	⋮	⋮
40	UPDATE: T_3 writes P_4	null
50	UPDATE: T_2 writes P_3	null
60	Begin Checkpoint	-
70	End Checkpoint	-
80	UPDATE: T_1 writes P_1	null
90	UPDATE: T_3 writes P_5	40
100	UPDATE: T_3 writes P_2	90
110	UPDATE: T_2 writes P_1	50
120	T_3 Commit	100
130	T_3 End	120
140	UPDATE: T_1 writes P_5	80
150	T_1 Abort	140
160	CLR: Undo write to P_5 (T_1, LSN 140), undoNextLSN: 80	150
	CRASH!	

The following dirty page table and transaction table are read from the checkpoint:

Transaction Table

Transaction	**Status**	**lastLSN**
T_2	Running	50
T_3	Running	40

Dirty Page Table

Page	**recLSN**
P_4	40

6. (1 point) Assume that at the end of the Analysis phase, we convert all Running status to Aborting in the transaction table. For each transaction, give its status in the transaction table at the end of the analysis phase. List its status as "A" if the transaction is Aborting and "C" if the transaction is Committing. If a transaction is **NOT** in the transaction table, list its status as "0".

Solution: The transaction table after the analysis phase is shown below. T_1 and T_2 were running at the time of the crash, so they are marked as Aborting in the transaction table. T_3 committed and ended before the crash, so it is absent from the transaction table.

Transaction	**Status**	**lastLSN**
T_1	Aborting	160
T_2	Aborting	110

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

7. (2.5 points) For each page referenced in the log, give its recLSN in the dirty page table at the end of the analysis phase. If a page is **NOT** in the dirty page table, list its recLSN as "0". Assume the buffer manager does not flush any pages during recovery.

Solution: The dirty page table after the analysis phase is show below.

Page	**recLSN**
P_1	80
P_2	100
P_3	0
P_4	40
P_5	90

This question was graded as 5 independent true/false questions, each worth 0.5 points. That is, you got 0.5 points for every correct choice that you selected and 0.5 points for every incorrect choice that you did *not* select.

8. (1 point) Mark the LSNs of the records that will actually be redone in the redo phase (i.e. that will require us to update a database page).

Solution: The redo phase of ARIES begins at the smallest recLSN in the dirty page table. In this example, that is 40, the recLSN of P_4. We then walk forward through the log and redo all the UPDATEs except for those that satisfy one of the three conditions under which we do not redo an UPDATE.

The UPDATE with LSN 50 is not redone becasue P_3 is not in the dirty page table. All other UPDATEs are redone. These updates have LSNs 40, 80, 90, 100, 110, 130, and 150.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

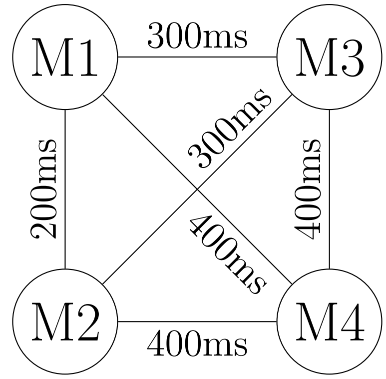
9. (1 point) Mark the LSNs of the UPDATE records that we write CLRs for during the undo phase.

Solution: The undo phase of ARIES undoes all of the UPDATEs made by the aborting transactions. In this example, T_1 and T_2 are aborting, so we must undo the UPDATEs with LSNs 50, 80, 110, and 140. The CLR with LSN 160 undoes the UPDATE with LSN 140 and is already in the log, so we introduce three new CLRs to undo the UPDATEs with LSNS 50, 80, and 110.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

3 Two-Phase Commit

Our database runs on 4 machines and uses Two-Phase Commit. Machine 1 is the Coordinator, while Machines 2, 3, and 4 are Participants.



Suppose our machines are connected such that the time it takes to send a message from Machine i to Machine j is $100 \cdot \max(i, j)$ milliseconds (see graph). Assume these communication latencies are symmetric: it takes the same amount of time to send from i to j as it takes to send from j to i. For example, sending a message between Machine 2 and Machine 4 takes 400 milliseconds in either direction.

Assume that the transaction will commit (i.e. all subordinates vote yes), and that everything is instantaneous except for the time spent sending messages between two machines.

1. (0.25 points) What is the first message Machine 1 sends?

   A. VOTE YES   **B. PREPARE**   C. COMMIT   D. None of these

   Solution: The coordinator (i.e. Machine 1) begins the first round of two-phase commit by sending a PREPARE message to all of the subordinates.

   This question was graded all or nothing. The correct answer received 0.25 points; every other answer received 0 points.

2. (0.25 points) What is the second message Machine 1 sends?

   A. VOTE YES   B. PREPARE   **C. COMMIT**   D. None of these

   Solution: The coordinator (i.e. Machine 1) begins the second phase of two-phae commit by sending a COMMIT message to all of the subordinates. Note that the problem states that all subordinates vote yes in the first round which is why the coordinator sends a COMMIT message (instead of an ABORT message) to start the second round.

   This question was graded all or nothing. The correct answer received 0.25 points; every other answer received 0 points.

3. (1 point) How much time passes from when Machine 1 sends its first message to when Machine 1 sends its second message?

   Solution: This is just the maximum time for a round-trip between Machine 1 and Machines 2–4, since Machine 1 sends its second message (COMMIT) once its first message (PREPARE) reaches each of the other machines and each of the machines respond back with a VOTE YES message.

   The time for a round-trip between Machine 1 and Machine i is $2 \cdot 100 \cdot i$, and the largest of these is with Machine 4: $2 \cdot 100 \cdot 4 = 800$ms.

   This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

4. (0.25 points) What is the first message Machine 2 sends?

   A. VOTE YES B. PREPARE C. COMMIT D. None of these

   Solution: A subordinate (e.g. Machine 2) responds to a PREPARE message from the coordinator with a VOTE YES or VOTE NO. The problem states that all subordinates vote yes, so Machine 2 sends a VOTE YES message.

   This question was graded all or nothing. The correct answer received 0.25 points; every other answer received 0 points.

5. (0.25 points) What is the second message Machine 2 sends?

   A. VOTE YES B. PREPARE C. COMMIT **D. None of these**

   Solution: The second message Machine 2 (a participant) sends is an ACK.

   This question was graded all or nothing. The correct answer received 0.25 points; every other answer received 0 points.

6. (1 point) How much time passes from when Machine 2 sends its first message to when Machine 2 sends its second message?

   Solution: Let's have Machine 1 send the PREPARE message at time 0.

   This PREPARE message will reach Machine 2 at time 200ms, so Machine 2 sends its first message (VOTE YES) at time 200ms.

   Machine 1 sends the COMMIT message at time 800ms (when it hears back from Machine 4), and this message takes 200ms to reach Machine 2. Machine 2 therefore receives the COMMIT message at time 1000ms, and sends its second message (ACK).

   The time passed is therefore 1000ms - 200ms = 800ms.

   The correct answer received 1 point. 0.5 points of partial credit was given to the answer 600ms (failing to realize that Machines 3 and 4 have not received PREPARE at the time that Machine 2 sends a VOTE YES, but correctly observing that the coordinator waits for all votes). Every other answer received 0 points.

7. (0.5 points) True or False. A transaction is considered committed even if over half of the participants do not acknowledge the commit.

   Solution: True. A transaction must commit once the COMMIT message is sent out, even if all the participants promptly crash repeatedly and do not respond with ACKs as a result.

   This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

Now suppose that our implementation of 2-Phase Commit has an off-by-one bug where the Coordinator receives, but does not use, Machine 4's vote. That is, Machine 4's vote does not affect whether or not the transaction commits or aborts.

8. (0.5 points) True or False. A transaction that should normally commit may be aborted instead.

> **Solution:** False. If the transaction would normally commit, then Machines 2 and 3 must have been functional and voted yes, so the transaction would still have to commit with this bug.
>
> This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

9. (0.5 points) True or False. A transaction that should normally abort may be committed instead.

> **Solution:** True. If Machine 4 votes no and other participants vote yes, then the transaction should be aborted but commits anyways.
>
> This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

10. (0.5 points) True or False. A transaction that should normally commit may be committed properly.

> **Solution:** True. If all machines vote yes, then the transaction commits even if we ignore Machine 4's vote.
>
> This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

11. (0.5 points) True or False. A transaction that should normally abort may be aborted properly.

> **Solution:** True. If Machine 2 or 3 votes no, then the transaction aborts in both cases.
>
> This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

4 Joins

4.1 Young Justice... again

For the following questions in this section, assume that we are streaming our query output to a terminal. (Do not consider the cost of writing the final output)

```
CREATE TABLE JusticeLeague (
    member_id INTEGER PRIMARY KEY,
    code_name CHAR(33),
    birth_planet CHAR(20),
    power_level INTEGER
);

CREATE TABLE Teaches (
    member_id INTEGER PRIMARY KEY REFERENCES JusticeLeague,
    teacher_id INTEGER REFERENCES JusticeLeague,
    since DATE
);
```

We assume that

- JusticeLeague has [J] = 100 pages
- JusticeLeague has $|J| = 1000$ members
- Teaches has $[T] = 200$ pages
- Teaches has $|T| = 2000$ members
- Buffer size = 12 pages
- An Alternative-1 B+ tree for Teaches indexed on `T.member_id` has the height 2. *In this problem, we interpret height as the number of I/Os it takes to travel from the root to the leaf page.*

1. (2 points) What is the I/O cost of using Index Nested Loop Join to perform a natural join of Teaches and JusticeLeague? Consider table JusticeLeague as the outer relation. Note that we want the actual cost in I/O requests, not a Selinger-style estimate. Treat each I/O request the same, regardless of whether you think the request will hit in the buffer pool. *Your answer must be a single integer.*

   Solution: $[J] + |J|$*(cost of finding T)

   100 + 1000 * 2 = 2100 I/O

2. (2 points) Assume we introduce an Alternative 2, clustered B+ tree for JusticeLeague indexed on `member_id` with height 2. What is the I/O cost of using Index Nested Loop Join to perform a natural join of Teaches and JusticeLeague? This time consider table Teaches as the outer relation. *Your answer must be a single integer.*

   Solution: $[T] + |T|$*(cost of finding J)

   200 + 2000 * 3 = 6200 I/O

3. (1 point) If the previous question's B+ tree for JusticeLeague had used an unclustered index, would the I/O cost have been different? (Answer only in Yes or No)

Solution: No

4.2 General Join Algorithm Questions

4. (2 points) In this question, we explore the strengths and weaknesses of various join algorithms. Assume throughout that we are joining two tables R and S, and we have B buffers of memory for our join. Do not assume any additional information about the tables that is not explicitly provided.

   (a) (1 point) We are performing an equijoin. $[R] < B$, $[S] > B^3$. Which join algorithm will provide the correct answer with the fewest I/Os:

      A. Block Nested Loops Join

      B. Grace Hash Join

      C. Sort-Merge Join

   (b) (1 point) We are performing an equijoin. $B < [R] < (B-1)^2$, and $B < [S] < (B-1)^2$. Both R and S have only 2 distinct values in the join key column. Which join algorithm will provide the correct answer with the fewest I/Os:

      A. Block Nested Loops Join

      B. Grace Hash Join

      C. Sort-Merge Join

5. (2 points) Consider the following "Index On Demand" join algorithm. Given an equijoin on tables R and S (with $[R] < [S]$), but no indexes on the join columns initially, the algorithm will (a) bulk-load an Alternative-1 B+-tree on R, (b) perform an Index Nested Loops Join on $S \bowtie R$ using the index, and (c) delete the index on R. Mark all of the following that you believe to be true for joining R and S, assuming $B < [R] < [S] < (B-1)^2$:

   A. This algorithm generates more I/Os than Grace Hash Join (without recursive partitioning).

   B. This algorithm generates more I/Os than a 2-pass Sort-Merge join.

   C. This algorithm generates more *random* I/Os than Block Nested Loops Join.

   D. If we omit step (c), this algorithm might be of interest for its benefits on subsequent queries, even though it could be sub-optimal for the current query.

5 Query Optimization

For the following questions, assume the following:

- The System R assumptions about uniformity and independence from lecture hold
- We use System R defaults when selectivity estimation is not possible
- Primary key IDs are sequential in each table, starting from 1, and there are no gaps in the ID sequence in any table
- Our system implements only implements Grace Hash Join. It allocates 5 pages of memory for the join operator.
- Assume all indices are alternative 3 indices.
- Assume the optimizer statistics are fully accurate w.r.t. the current content of the database.

Table Schema	Records	Pages	Indices

`CREATE TABLE User (`
`uid INTEGER PRIMARY KEY,`
`fullname VARCHAR(32),`
`country VARCHAR(4))`
`)`
	1,000	100	None

`CREATE TABLE Item (`
`sku INTEGER PRIMARY KEY,`
`itemname VARCHAR(32),`
`price NUMERIC`
`)`
	10,000	1,000	• Index 0: Unclustered Index on sku.
			• Index 1: Clustered Index price Low 0, High 100

`CREATE TABLE Order (`
`uid INTEGER REFERENCES User,`
`sku INTEGER REFERENCES Item,`
`quantity INTEGER)`
`)`
	100,000	5,000	• Index 2: Clustered Index on uid

`CREATE TABLE Tax (`
`sku INTEGER PRIMARY KEY`
`REFERENCES Item,`
`taxrate NUMERIC`
`)`
	10,000	1,000	None

1. (1 point) After applying the System R optimizer, which query has a *lower* estimated final cost for the optimal query plan.

A.
```
SELECT *
FROM Item
WHERE Item.itemname = 'WidgetA'
```

B.
```
SELECT *
FROM Item
WHERE Item.price < 3
```

> **Solution:** B. Since there is no index on `itemname`, the optimizer will estimate a selectivity of $\frac{1}{10}$, compared to a selectivity of $\frac{3}{100}$ for Item.price.
>
> This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

2. (1 point) After applying the System R optimizer, which query has a *lower* estimated final cost for the optimal query plan.

A. `SELECT *`
`FROM Item`
`ORDER BY Item.price`

B. `SELECT *`
`FROM Tax`
`ORDER BY Tax.taxrate`

Solution: A. `Item` and `Tax` have the same number of pages, but there is an eligible clustered index on `Item.price`. Thus, we can perform an index scan on `Item` to return tuples ordered by `Item.price`, but we have to perform an external sort on `Tax` in order to return tuples ordered by `Tax.taxrate`.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

3. (0.5 points) (True or False) The System R optimizer will apply a projection to the Tax table that preserves only the `taxrate` attribute before performing a join for the following query:

```
SELECT taxrate
FROM Item, Tax
WHERE Tax.sku = Item.sku
```

Solution: False. `sku` is also needed for the join.

This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

4. (0.5 points) (True or False) For the following query, one can determine the cardinality of the join result (i.e the number of tuples in the result) exactly:

```
SELECT taxrate
FROM Item, Tax
WHERE Tax.sku = Item.sku
```

Solution: True. The foreign key relationship enforces a constraint between the two tables. Note that `Tax.sku` is a primary key and hence is never null.

This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

5. (0.5 points) (True or False) In general, even if an eligible index exists, a sequential scan can be selected rather than an index scan.

Solution: True. For example, a full table scan can often be less expensive than an unclustered index scan with a low selectivity predicate.

This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

6. (1.5 points) Consider the following query:

```
SELECT *
FROM User, Order
WHERE User.uid = Order.uid AND
      User.country = 'USA'
```

During Pass 2 of the System R Optimizer, what is the estimated cost of executing the query using a Grace hash join? Be sure to include the cost of the heap scans and any other upstream operators.

> **Solution:** First, notice that we apply the predicate `User.country = 'USA'` to `User` before doing any joins. This incurs a cost of 100 pages. Since there isn't an index on this attribute, the selectivity estimate is $\frac{1}{10}$. This means that optimizer estimates that after applying the predicate there are 100 records and 10 pages. The `Order` table is also scanned and this incurs a cost of 5000 pages.
>
> $10 < 5 * 4$, so Grace hash can be performed in 2 passes–leading to the formula $(100 + 5000) + 2 * (10 + 5000) = 15120$.

7. (1 point) Consider the following query:

```
SELECT *
FROM User, Order, Tax, Item
WHERE User.uid = Order.uid AND
          Item.sku = Order.sku AND
          Tax.sku = Item.sku AND
          Tax.sku = Order.sku
```

Without making assumptions about costs and selectivities, which of the following join orders could **NEVER** be considered by the System R optimizer to answer this query? *Mark all that apply*

A. (((User ⋈ Order) ⋈ Item) ⋈ Tax)

B. (User ⋈ Order) ⋈ (Item ⋈ Tax)

C. (((User ⋈ Item) ⋈ Order) ⋈ Tax)

D. (((User ⋈ Order) ⋈ Tax) ⋈ Item)

E. (((Tax ⋈ Item) ⋈ Order) ⋈ User)

F. None of the above

> **Solution:** Rubric: all-or-nothing. B is not a left-deep query plan, so it is never considered by the System R optimizer. C's leftmost join is between `User` and `Item` but this is a cross join. The System R optimizer defers cross joins to the top of the query plan, so the System R optimizer would not consider C. All other query plans may be considered by the System R optimizer.

8. (0.5 points) (True or False) Ignoring order by statements, group by statements, and nested queries, the System R optimizer is guaranteed to return the left-deep query plan with the lowest IO cost with respect to the estimated costs.

> **Solution:** True. The System R optimizer is not guaranteed to return the best overall query plan, but it *is* guaranteed to return the best left-deep query plan with respect to the estimated costs. Note that the estimated costs may be erroneous, so the query plan returned by the System R optimizer may not actually be the best left-deep plan.

6 SQL

Consider a table `Friend` representing a social network. Each record of the Friend table consists of two integers corresponding to a friendship relationship between two users (identified by the id number). Assume that no person is friends with themselves.

```
CREATE TABLE Friend(
    id1 INTEGER,
    id2 INTEGER,
    PRIMARY KEY (id1,id2));
```

1. (1.5 points) Some of the friendship relationships in this table are not reciprocated, i.e., `(id1=a,id2=b)` exists in the table but `(id1=b,id2=a)` does not exist. Which of the following SQL queries identifies all tuples where the reciprocal relationship does not exist in the table. *Mark all that apply.*

   A.
   ```
   SELECT *
   FROM Friend f1
   WHERE NOT EXISTS (SELECT *
                     FROM Friend f2
                     WHERE f1.id1 = f2.id2 AND f1.id2 = f2.id1)
   ```

   B.
   ```
   SELECT t.id1, t.id2
   FROM (
       (SELECT id1, id2
        FROM Friend f1)
       UNION ALL
       (SELECT f2.id2 as id1, f2.id1 as id2
        FROM Friend f2)
   ) as t
   GROUP BY t.id1, t.id2
   HAVING count(*)  <  2
   ```

   C.
   ```
   SELECT t.id1, t.id2
   FROM (
       SELECT f2.id2 as id1, f2.id1 as id2
       FROM Friend f2
   ) as t FULL OUTER JOIN Friend
   ON t.id1 = Friend.id1 AND t.id2 = Friend.id2 AND Friend.id1 <> NULL
   ```

   D. None of the above

   Solution: A, B.

   Answer **A** correctly uses a subquery to determine whether for each friendship tuple there exists at least one inverse relationship.

   Answer **B** takes a union all of friends table and a friends table where the attributes are in reverse order. It counts the number of elements in each group. This relies on the fact that (id1,id2) is a primary key.

   Answer **C** is incorrect because the statement Friend.id1 <> NULL should not be applied in the join condition.

2. (1 point) Consider the same table, but without the primary key constraint. Continue to assume that no person is friends with themselves.

```
CREATE TABLE Friend(
   id1 INTEGER,
   id2 INTEGER);
```

This table can contain multiple copies of the same friendship relationship, e.g., two tuples with `(id1=a,id2=b)`. Therefore, each distinct `(id1=a,id2=b)` pair will have a count of the number of times it occurs in the relation. We want to write a query that finds all distinct pairs where the count of `(id1=a,id2=b)` is not equal to the count of `(id1=b,id2=a)`. *Mark all that apply.*

A.
```
SELECT id1, id2, count(*)
FROM Friend
GROUP BY id1, id2

EXCEPT

SELECT id2, id1, count(*)
FROM Friend
GROUP BY id1, id2
```

B.
```
SELECT id1, id2, count(*)
FROM Friend
GROUP BY id1, id2

EXCEPT ALL

SELECT id2, id1, count(*)
FROM Friend
GROUP BY id1, id2
```

C.
```
(SELECT id1, id2, count(*)
FROM Friend
GROUP BY id1, id2

EXCEPT

SELECT id2, id1, count(*)
FROM Friend
GROUP BY id1, id2)

UNION ALL

(SELECT id2, id1, count(*)
 FROM Friend
 GROUP BY id1, id2

 EXCEPT

 SELECT id1, id2, count(*)
 FROM Friend
 GROUP BY id1, id2)
```

D. None of the above

Solution: A and B are correct see `http://sqlfiddle.com/#!17/f3cf9/1`. C is incorrect because it can contain tuples that don't exist in the database: `http://sqlfiddle.com/#!17/f3cf9/3`

3. (1.5 points) Assume the following `Friend` table with the primary key constraint.

```
CREATE TABLE Friend(
    id1 INTEGER,
    id2 INTEGER,
    PRIMARY KEY (id1,id2));
```

A "triangle" of friends is defined as Person a is friends with Person b, Person b is friends with Person c, and Person c is friends with Person a. Which of the following queries lists of all of the distinct triangles output in alphabetical order—i.e., if a, b, c is in the result then c, a, b should not be in the result. *Mark all that apply.*

A.
```
SELECT f1.id1, f2.id1, f3.id1
FROM Friend f1, Friend f2, Friend
    f3
WHERE f1.id2 = f2.id1 AND
      f2.id2 = f3.id1 AND
      f3.id2 = f1.id1 AND
```

```
       f1.id1 < f2.id1 AND
       f2.id1 > f3.id1
```

B.
```
SELECT f1.id1, f2.id1, f3.id1
FROM Friend f1, Friend f2, Friend
    f3
WHERE f1.id2 = f2.id1 AND
      f2.id2 = f3.id1 AND
      f3.id2 = f1.id1
```

C.
```
SELECT f1.id1, f2.id1, f3.id1
FROM Friend f1, Friend f2, Friend
    f3
WHERE f1.id2 = f2.id1 AND
      f2.id2 = f3.id1 AND
      f3.id2 = f1.id1 AND
      f1.id1 < f2.id1 AND
      f2.id1 < f3.id1
```

D. None of the above

Solution: Only C is correct. See `http://sqlfiddle.com/#!17/4e78f/1/0`.

7 Replication

1. (3 points) Which of the following are reasons to replicate data:

   A. To increase the odds that some node will be able to respond to any request at any given time

   B. To allow multiple nodes to split up the work for a large volume of requests.

   C. To reduce latency by allowing clients to fetch data from a nearby server

   D. To handle ever-increasing database sizes.

   E. To hide transient performance problems.

   F. To recover from failures.

   Solution: Rubric: each answer treated as independent T/F question. A. is Availability. B is Workload Scaling. C is Locality. D is false: that's what partitioning is for. E is a version of Availability: latency within timeout. F is a version of Availability as well.

2. (2 points) Consider a single-master system that performs 2PC upon transaction commit. Which of the following problems could occur in a single-master system, but are prevented by 2PC?

   A. The standby node may have an older value for some key than the master.

   B. Transactions could commit while the standby node has failed.

   C. The standby node may have processed an update that conflicts with the master node.

   D. The master node may fail and block progress.

   Solution: Rubric: each answer treated as independent T/F question. A: True. this is likely to be the case before 2PC commit processing. B. True. The standby could get arbitrarily out of date. C. False. The standby does not process updates in Single-Master. D. While this could happen, 2PC does not help.

3. (2 points) Which of the following drawbacks are true of 2PC-controlled replication?

   A. If it is not strict 2PC, you could get cascading aborts.

   B. A failed participant node can cause live nodes to abort.

   C. A failed coordinator node can freeze up the entire system indefinitely.

   D. Message latency for 2PC across wide-area networks can be high, and lead to slow transaction performance.

   Solution: Rubric: each answer treated as independent T/F question. A: False: this makes no sense. Don't confuse 2PL and 2PC! B. TRUE, due to unanimous it's true. C. TRUE. Eventually the coordinator will time out waiting for the failed node and abort transactions. D. TRUE. This is the case that Paxos Commit came to solve. E. This is TRUE – e.g. the slides point out that Google spanner takes 100ms to turn around a simple transaction.

4. (0.5 points) True or False: In multi-master replication, it is possible for the system to get "split brain": two nodes may disagree about the value associated with some key.

Solution: True.

5. (0.5 points) True or False: In the absence of 2PC, multi-master replication is a high-bandwidth approach because many nodes can service writes to the same key in parallel, without sending each other messages.

Solution: True.

8 B+ Trees

8.1 True/False

1. (3 points) Mark all statements that are true.

   A. A B+ tree is always balanced-height

   B. The keys on an internal node are always in sorted order

   C. A B+ tree built using bulk loading is always more bushy than a B+ trees built from standard insert operations

   D. Alternative 3 is always better than alternative 2

   E. The number of records in the table must be known before performing bulk loading

   F. A leaf node can be a root node

Solution: A is true because B+ trees are dynamic. B is true by definition. C is false with a low fill factor. D is false because it can cause variable sized data entries. D is also false if index is built on primary key, in this case they are the same. E is not a requirement for bulk loading. F this is not true because the index could be on a different key. G true when height = 1.

8.2 Best and Worst Case IOs

Consider an alternative 2 B+ tree index of height $h = 3$ and order $d = 4$. In this problem, the height of a tree corresponds to the number of I/Os to traverse to a leaf. Assume that the index key is the primary key of the table. Note that index pages include both leaf and non-leaf pages. The maximum number of leaf pages that can exist in this index is $(2d + 1)^3 = 729$. The minimum number of leaf pages that can exist in this index is $(d + 1)^3 = 125$.

2. (0.5 points) In the **worst case**, what is the **largest** number of index pages that will be **read** in an equality search?

   A. 0 B. 1 **C. 3** D. 4 E. 729 F. 731 G. 732 H. none of the above

Solution: 3. It will read one page for each height level to get to the correct leaf page, where it can determine if the specified record exists or not.

3. (0.5 points) In the **best case**, what is **fewest** number of index pages that will be **read** in an equality search?

   A. 0 B. 1 **C. 3** D. 4 E. 125 F. 127 G. 128 H. none of the above

Solution: 3. Same reasoning as Q2.

4. (0.5 points) In the **worst case**, what is the **largest** number of index pages that you will **write to** during an insert? Include both existing index pages that are written to and index pages that are created as part of the write in your answer.

A. 2
B. 3
C. 4
D. 5
E. 6
F. 7
G. 8
H. none of the above

Solution: 7. If the tree is maximally full, it will split at every level upon insertion which will require it to write two index pages at every height. It will also have to write a new root page. Therefore the max number of index pages written on insert is $2 * h + 1 = 6 + 1 = 7$.

5. (0.5 points) In the **best case**, what is the **fewest** number of index pages that you will **write to** during an insert? Include both existing index pages that are written to and index pages that are created as part of the write in your answer.

A. 0
B. 1
C. 2
D. 3
E. 4
F. 5
G. 6
H. none of the above

Solution: 1. You can only write one leaf page on insert if there is enough space to hold the record.

6. (0.5 points) In the **worst case**, what is the **largest** number of index pages that will be **read** during a range search that returns exactly 7 records?

   A. 2
   B. 3
   C. 4
   D. 5
   E. 6
   F. 7
   G. 8
   H. none of the above

Solution: 5. It will take 3 reads to get to the correct leaf page. The records can be spread over three pages, so it can take two additional reads to return all 7 records if the leaf pages only contain the minimal 4 records each.

7. (0.5 points) In the **best case**, what is the **fewest** number of index pages that will be **read** during a range search that returns exactly 7 records?

   A. 2
   B. 3
   C. 4
   D. 5
   E. 6
   F. 7
   G. 8
   H. none of the above

Solution: 3. It takes 3 reads to get to the correct leaf page. If the records are packed, they can all fit on one page.