CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (SPRING 2024)
PROF. JIGNESH PATEL

Homework #4 (by Amy Cheng, Ritu Pathak, Shivang Dalal) – Solutions

Due: **Saturday April 6, 2024 @ 11:59pm**

IMPORTANT:

- Enter all of your answers into **Gradescope by 11:59pm on Saturday April 6, 2024**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.

For your information:

- Graded out of **100** points; **4** questions total
- Rough time estimate: ≈ 2 - 4 hours (0.5 - 1 hours for each question)

Revision : 2024/04/04 13:56

Question	Points	Score
Query Execution, Planning, and Optimization	24	
Serializability and 2PL	26	
Hierarchical Locking	24	
Optimistic Concurrency Control	26	
Total:	100	

Question 1: Query Execution, Planning, and Optimization . . . . . [24 points]

Graded by:

(a) **[3 points]** The zone map optimization is more effective in speeding up OLAP queries as opposed to OLTP queries.

■ **True** ☐ False

Solution: Recall that the zone map optimization pre-computes aggregations for each tuple attribute in a page. Since OLAP DBMSs (which store columns of data contiguously) are typically the DBMS of choice when running aggregate queries, zone maps are more useful for speeding up OLAP queries.

(b) **[3 points]** For OLAP queries, which often involve complex operations on vast datasets, intra-query parallelism is typically not preferred to optimize performance.

☐ True ■ **False**

Solution: False. OLAP queries, characterized by their complex operations on large volumes of data, can greatly benefit from intra-query parallelism. By executing the operations of a single query in parallel, it helps in significantly decreasing the latency, thus optimizing the performance of these types of queries.

(c) **[3 points]** The process per DBMS worker approach provides better fault isolation than the thread per DBMS worker approach.

■ **True** ☐ False

Solution: True. In the process per DBMS worker approach, each worker runs in its process, meaning that if one process fails, it doesn't directly affect the others. Whereas in a thread per DBMS worker approach, all threads share the same process address space, so a fault in one thread could potentially impact others.

(d) **[3 points]** In OLAP workload, the vectorized model's performance improvements come mainly from the reduction in the number of disk I/O operations.

☐ True ■ **False**

Solution: False. While the Vectorized Model can reduce some I/O operations due to its batch processing, its primary advantage is from reducing CPU overhead, optimizing cache utilization, and leveraging SIMD instructions.

(e) **[3 points]** An index scan is always better (fewer I/O operations, faster run-time) than a sequential scan if the query contains an `ORDER BY` clause matching the index key.

☐ True ■ **False**

Solution: An index scan may require multiple I/O operations per result tuple (to look up the row in the index, then retrieve additional attributes in the heap) whereas a sequential scan will just go through the heap. Moreover, a sequential heap scan may benefit more from pre-fetching pages. PostgreSQL's optimizer defaults to picking sequential scans over index scans if it thinks that you're asking for more than (very approximately) 10% of all rows in the table. Note that based on the number of expected returned tuples,

Question 1 continues. . .

clustered indexes will perform very well. Unclustered indexes however will not be as helpful.

(f) **[3 points]** The query optimizer in a database management system always guarantees the generation of an optimal execution plan by exhaustively evaluating all possible plans to ensure the lowest cost for query execution.
☐ True ■ **False**

Solution: No, it is usually not necessary to estimate the cost of every plan for a query via a cost model. In this case, the time it would take to enumerate every plan and then filter out the plans to pick the most optimal one would introduce too high of an overhead compared to the query time itself. Usually, DBMSs will use rule-based optimizations (or heuristics) first, transforming the plan into a more simple one.

(g) **[3 points]** Predicate pushdown involves moving filter conditions closer to the node where the data is stored, while projection pushdown involves transferring only the necessary columns of the data.
■ **True** ☐ False

Solution: True. Predicate pushdown is an optimization technique that moves the predicate (i.e., the WHERE clause conditions) to execute on the data node where the relevant data is stored. This filters out irrelevant data at the source, reducing data transfer. Similarly, projection pushdown ensures that only required columns are retrieved, further optimizing data movement and query performance.

(h) **[3 points]** The vectorized query processing model (which uses SIMD instructions to parallelize operations) is an example of intra-query parallelism.
■ **True** ☐ False

Solution: This is true. Intra-query parallelism is when the DBMS executes the operations of a single query in parallel. Therefore, using SIMD instructions within a vectorized processing model to execute a query would be an example of intra-query parallelism.

Question 2: Serializability and 2PL............................[26 points]

(a) True/False Questions:

i. **[3 points]** Strong strict Two-Phase Locking (2PL) prevents the occurrence of cascading aborts and inherently avoids deadlocks without the need for additional prevention or detection techniques.

☐ True ■ **False**

Solution: False. While strict 2PL prevents cascading aborts by holding all the locks until a transaction reaches its commit point, ensuring that other transactions do not see the intermediate, uncommitted data, it does not inherently resolve deadlocks. Deadlocks, which are situations where two or more transactions prevent each other from progressing by holding locks that the other needs, still require specific detection or prevention mechanisms in strict 2PL.

ii. **[3 points]** For a schedule following strong strict 2PL, the dependency graph is guaranteed to be acyclic.

■ **True** ☐ False

Solution: True. Using regular 2PL guarantees a conflict-serializable schedule, and a schedule provided by strong strict 2PL has an even stronger guarantee which means it also avoids the formation of cycles in the dependency graph.

iii. **[2 points]** Using 2PL guaruntees a conflict-serializable schedule.

■ **True** ☐ False

Solution: True. Using regular 2PL guarantees a conflict-serializable schedule because it generates schedules whose precedence graph is acyclic. This is because this ordering of acquiring resources (via the locks) ensures that there is a order of acquisition precedence.

iv. **[2 points]** Conflict-serializable schedules prevent unrepeatable reads and dirty reads.

☐ True ■ **False**

Solution: False. Conflict-serializability ensures that the schedule is conflict equivalent to a serial schedule, thus protecting against unrepeatable reads. However, dirty reads can still occur in conflict-serializable schedules, especially when considering cascading aborts. If a transaction reads data written by another transaction which later gets aborted, then the first transaction has effectively read "dirty" data.

v. **[2 points]** A schedule that is view-serializable is also conflict-serializable.

☐ True ■ **False**

Solution: False. There exists view-serializable schedules that are not conflict-serializable.

(b) Serializability:

Consider the schedule of 4 transactions in Table 1. R(·) and W(·) stand for 'Read' and 'Write', respectively, and time increases from left to right. (This is in contrast to the diagrams in class, where time proceeded downward.)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
T_1		R(B)		W(C)	W(A)					
T_2	W(E)			R(E)			W(C)	R(D)	W(A)	
T_3						R(A)	W(B)			R(B)
T_4			R(D)	R(B)		W(D)				

Table 1: A schedule with 4 transactions

i. **[3 points]** Is this schedule serial?
☐ Yes ■ **No**

Solution: This schedule isn't serial because this schedule interleaves the actions of different transactions.

ii. **[3 points]** Is this schedule conflict-serializable?
■ **Yes** ☐ No

Solution: This schedule is conflict-serializable because there are no cycles in its data dependency graph. Moreover, this schedule is conflict equivalent (every pair of conflicting operations is ordered in the same way) to a serial schedule of transaction execution.

iii. **[5 points]** Compute the conflict dependency graph for the schedule in Table 1, selecting all edges (and the object that caused the dependency) that appear in the graph.

■ $T_1 \to T_2$
☐ $T_2 \to T_1$
■ $T_1 \to T_3$
☐ $T_3 \to T_1$
☐ $T_2 \to T_3$
■ $T_3 \to T_2$
☐ $T_1 \to T_4$
☐ $T_4 \to T_1$
☐ $T_2 \to T_4$
■ $T_4 \to T_2$
☐ $T_3 \to T_4$
■ $T_4 \to T_3$

Solution: The answer is:

- $T_1 \to T_2(A, C), T_1 \to T_3(A, B)$
- $T_3 \to T_2(A)$
- $T_4 \to T_2(D), T_4 \to T_3(B)$

For A, there is a W-W conflict from T_1 to T_2, a W-R conflict from T_1 to T_3, and a R-W conflict from T_3 to T_2;
For B, there is a R-W conflict from T_1 to T_3 and a R-W conflict from T_4 to T_3;
For C, there is a W-W conflict from T_1 to T_2;
For D, there is a W-R conflict from T_4 to T_2.
For E, there are no conflicts.

iv. **[3 points]** Is this schedule possible under regular 2PL?
■ **Yes**
☐ No

Solution: This schedule is possible under 2PL because it is conflict-serializable, and 2PL is guaranteed to produce conflict serializable schedules.

Question 3: Hierarchical Locking .... [24 points]

Consider a database D consisting of two tables A (which stores information about musical artists) and R (which stores information about the artists' releases). Specifically:

- `R(rid, name, artist_credit, language, status, genre, year, number_sold)`
- `A(id, name, type, area, gender, begin_date_year)`

Table R spans 1000 pages, which we denote R1 to R1000. Table A spans 50 pages, which we denote A1 to A50. Each page contains 100 records. We use the notation R3.20 to denote the twentieth record on the third page of table R. There are no indexes on these tables.

Suppose the database supports shared and exclusive hierarchical intention locks (S, X, IS, IX and SIX) at four levels of granularity: database-level (D), table-level (R and A), page-level (e.g., R10), and record-level (e.g., R10.42). We use the notation IS(D) to mean a shared database-level intention lock, and X(A2.20-A3.80) to mean a set of exclusive locks on the records from the 20th record on the second page to the 80th record on the third page of table A.

For each of the following operations below, what sequence of lock requests should be generated to **maximize the potential for concurrency** while guaranteeing correctness?

(a) **[4 points]** Fetch the records of all musical artists in A with `type = 'Orchestra'`.

☐ SIX(D), S(A)
☐ IX(D), S(A)
■ IS(D), S(A)
☐ S(D)

Solution: The correct answer choice is IS(D), S(A). We need to scan records in table A to find records where `type = 'Orchestra'`. This choice is correct because it accesses the intended shared parent lock to get the shared lock on table A.

- SIX(D), S(A) is incorrect because it gains a shared+intention-exclusive lock on the database D when it only needs to read from A and has no intention of modifying any records.
- IX(D), S(A) is incorrect because it gains an intention-exclusive lock when it has no intention of modifying.
- S(D) is incorrect because it gains a shared lock on the entire database D when it only needs to fetch rows in table A.

(b) **[4 points]** Update the genre for all release records with `language = 'English'` to `'Musical theatre'`.

■ IX(D), X(R)
☐ SIX(D), X(R)
☐ IX(D), IX(R)
☐ IX(D), SIX(R)

Solution: The correct answer choice is `IX(D), X(R)`. This choice is correct because it accesses all intended locks and the exclusive lock on `R`, since we potentially need to modify all records in `R`.

- `SIX(D), X(R)` is incorrect because it gains a shared intention parent lock for `D`, when it only needs to read from `R`.
- `IX(D), IX(R)` is incorrect because it does not gain an exclusive lock for the records of `R` it needs to delete.
- `IX(D), SIX(R)` is incorrect because it does not gain an exclusive lock for the records of `R` it needs to delete.

(c) **[4 points]** Modify the 7^{th} record on `R80`.

☐ `IS(D), IS(R), IS(R80), X(R80.7)`
☐ `IX(D), IX(R), IX(R80), IX(R80.7)`
☐ `SIX(D), IX(R), IX(R80), X(R80.7)`
■ `IX(D), IX(R), IX(R80), X(R80.7)`

Solution: The correct choice is `IX(D), IX(R), IX(R80), X(R80.7)`. This choice is correct because it accesses all intended exclusive locks for all parent levels necessary, and then accesses the exclusive lock for the particular record.

- `IS(D), IS(R), IS(R80), X(R80.7)` is incorrect because the DBMS intends to write to a tuple, so it should not grab the shared intention parent locks.
- `IX(D), IX(R), IX(R80), IX(R80.7)` is incorrect because it only gets the intention exclusive lock for the record.
- `SIX(D), IX(R), IX(R80), X(R80.7)` is incorrect because the DBMS only intends to write to a tuple, not read any data. Therefore it should not grab the shared-exclusive intention lock.

(d) **[4 points]** Increment the `number_sold` for the 6^{th} record on `R999`.

☐ `IX(D), IX(R), SIX(R999), X(R999.6)`
☐ `IS(D), IS(R), IS(R999), S(R999.6)`
☐ `IX(D), IX(R), S(R999), X(R999.6)`
■ `IX(D), IX(R), IX(R999), X(R999.6)`

Solution: The correct choice is `IX(D), IX(R), IX(R999), X(R999.6)`. This choice is correct because it accesses all intention locks and the exclusive lock on `R999.6`, so it can perform both a read and write while holding this exclusive lock.

- `IX(D), IX(R), SIX(R999), X(R999.6)` is incorrect because it gains a shared intention lock for `R999` when it only needs to read `R999.6`, thus limiting potential for concurrency.

- `IS(D), IS(R), IS(R999), S(R999.6)` is incorrect because we plan on modifying `R999.6`, but we are only gaining a shared lock on `R999.6`, which isn't sufficient.
- `IX(D), IX(R), S(R999), X(R999.6)` is incorrect because while it gains a intention-exclusive locks for `R` and the database, it then gains a shared lock on `R999` which is not sufficient to gain the exclusive lock on `R999.6`.

(e) **[4 points]** Scan all records on pages `A10` to `A50` and modify the 20^{th} record on `A14`.

☐ `IX(D), S(A), X(A14)`
☐ `SIX(D), IX(A), IX(A14), X(A14.20)`
☐ `IX(D), IX(A), IX(A10-A50), IX(A14), X(A14.20)`
■ `IX(D), SIX(A), IX(A14), X(A14.20)`

Solution: The correct choice is `IX(D), SIX(A), IX(A14), X(A14.20)`. This choice is correct because it accesses all intended locks and the exclusive lock `X(A14.20)`. It also gains a shared lock on `A`, so it can read pages `A10` to `A50`.

- `IX(D), S(A), X(A14)` is incorrect because it fails to gain a intention-exclusive lock for `A`.
- `SIX(D), IX(A), IX(A14), X(A14.20)` is incorrect because it gains a shared intention lock for the database `D` when it only needs to read from `A`.
- `IX(D), IX(A), IX(A10-A50), IX(A14), X(A14.20)` is incorrect because it fails to gain shared locks to read from `A10-A15`.

(f) **[4 points]** Delete records in `A` if `type='Band'`.

☐ `SIX(D), SIX(A)`
■ `IX(D), X(A)`
☐ `IX(D), IX(A)`
☐ `SIX(D), X(A)`

Solution: The correct choice is `IX(D), X(A)`. This choice is correct because it accesses all intended locks and the exclusive lock on `A`, since we potentially need to modify all records in `A`.

- `SIX(D), SIX(A)` is incorrect because it gains a shared intention parent lock for both `D` and `A`, when it does not need to read any contents, and it does not gain the exclusive lock for the records of `A` it needs to delete.
- `IX(D), IX(A)` is incorrect because it does not gain an exclusive lock for the records of `A` it needs to delete.
- `SIX(D), X(A)` is incorrect because it gains a shared lock for the database, when it does not need to read any contents.

Question 4: Optimistic Concurrency Control .... [26 points]

Consider the following set of transactions accessing a database with object *A, B, C, D*. You should make the following assumptions:

- The transaction manager is using **optimistic concurrency control** (OCC).
- A transaction begins its read phase with its first operation and switches from the READ phase immediately into the VALIDATION phase after its last operation executes.
- The DBMS is using the serial validation protocol discussed in class where only one transaction can be in the validation phase at a time.
- Each transaction is doing **forward validation** (i.e. Each transaction, when validating, checks whether it intersects its read/write sets with any active transactions that have not committed yet).
- There are no other transactions in addition to the ones shown below.

Note: VALIDATION may or may not succeed for each transaction. If validation fails, the transaction will get immediately aborted.

time	T_1	T_2	T_3
1	READ(A)		
2			READ(B)
3			WRITE(B)
4		READ(A)	
5		READ(D)	
6	READ(B)		WRITE(C)
7			READ(D)
8			VALIDATE?
9	READ(C)		
10			WRITE?
11	VALIDATE?		
12	WRITE?		
13		WRITE(D)	
14		WRITE(C)	
15		VALIDATE?	
16		WRITE?	

Figure 1: An execution schedule

(a) **[4 points]** When is each transaction's timestamp assigned in the transaction process?
- ☐ The start of the write phase.
- ☐ Timestamps are not necessary for OCC.
- ■ **The start of the validation phase.**
- ☐ The start of the read phase.

Solution: Each transaction's timestamp is assigned at the beginning of the validation phase.

(b) **[4 points]** When time = 9, will T_1 read C written by T_3?
☐ Yes ■ **No**

Solution: In OCC, each transaction maintains a private workspace that is invisible to other transactions until its write phase is completed. Only transactions that start after T_3's write phase can see the value of A written by T_3, provided T_3 commits successfully. Hence, T_1 at time = 9 will read the original C, that isn't written by T_3.

(c) **[4 points]** Will T1 abort?
☐ Yes
■ **No**

Solution: T_1 does not do any writes, so T_1's write set does not intersect T_2's read set.

(d) **[4 points]** Will T2 abort?
☐ Yes
■ **No**

Solution: No other transactions will be committed in the future, so there are no conflicts.

(e) **[4 points]** Will T3 abort?
■ **Yes**
☐ No

Solution: T_3's write set intersects T_1's read set.

(f) **[2 points]** OCC works best when concurrent transactions access the same subset of data in a database.
☐ True ■ **False**

Solution: OCC is good to use when the number of conflicts is low.

(g) **[2 points]** Transactions can suffer from *phantom reads* in OCC.
■ **True** ☐ False

Solution: Considering the following sequence of events for table A and transactions T_1 and T_2:

- T_1 reads all tuples of A.
- T_2 inserts a new tuple into A.
- T_2 enters validation phase. Note T_2's write set contains the new tuple whereas T_2's read set only contains the initial existing tuples. So T_2's write set does not intersect T_1's read set, and T_2 validation succeeds.

Question 4 continues. . .

- T_2 completes the write phase.
- T_1 reads all tuples of A again, now including the tuple added by T_2.

(h) **[2 points]** Aborts due to OCC are wasteful because they happen after a transaction has already finished executing.

■ **True** □ False

Solution: During the read phase, a transaction will execute operations on its private workspace. It is only after the read phase and validation phase that a transaction may abort.