CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2023)
PROF. ANDY PAVLO AND JIGNESH PATEL

# Homework #4 (by Aolei Zhou) – Solutions

Due: **Sunday November 12, 2023 @ 11:59pm**

**IMPORTANT:**

- Enter all of your answers into **Gradescope by 11:59pm on Sunday November 12, 2023**.
- **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: $\approx$ 2 - 4 hours (0.5 - 1 hours for each question)

*Revision* : 2023/12/06 10:00

| 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]** For a query that involves a range condition, zone maps can help in determining if a block contains any relevant data.

■ **True** □ False

**Solution:** True. Given a range condition, if the range doesn't overlap with the minimum and maximum values recorded in the zone map for a block, that block can be skipped.

(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]** For a table with a well-maintained B-Tree index, an index scan will always be faster than a sequential scan because of fewer I/O operations and faster run-time.

□ True ■ **False**

**Solution:** False. 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.

(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]** In the context of query optimization, where accurate statistics are crucial for plan choices, sampling requires examining every row in a table to compute these vital statistics.

□ True ■ **False**

**Solution:** False. Within query optimization, sampling is an efficient technique utilized precisely because it avoids the need to examine every row in a table. Instead, it estimates statistics by checking a representative subset of rows, saving considerable time and resources.

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

(a) True/False Questions:

i. **[2 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. **[2 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 regular (i.e., not strong strict) 2PL does not guarantee a conflict serializable schedule.

□ True ■ **False**

**Solution:** False. 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, even in the event of cascading aborts.

□ 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]** 2PL is a pessimistic concurrency control protocol.

■ **True** □ False

**Solution:** True. 2PL is considered a pessimistic concurrency control protocol because it uses locks to prevent conflicts before they can occur. By ensuring transactions follow a strict locking protocol, 2PL aims to prevent any possible conflicts, in contrast to optimistic protocols which allow conflicts but then resolve them.

vi. **[2 points]** Every conflict-serializable schedule is always conflict-equivalent to at least one view-serializable schedule.

■ **True** □ False

**Solution:** True. Every conflict-serializable schedule is conflict-equivalent to a serial schedule. Since all serial schedules are inherently view-serializable, the statement holds true.

(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(C) | | | R(A) | | | | | |
| $T_2$ | | | | | | | W(A) | R(D) | W(B) | |
| $T_3$ | W(B) | | | | | W(C) | | | | R(D) |
| $T_4$ | | | W(B) | W(A) | | | | | | |

Table 1: A schedule with 4 transactions

i. **[2 points]** Is this schedule serial?

□ Yes ■ **No**

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

ii. **[2 points]** Is this schedule conflict serializable?

□ Yes ■ **No**

**Solution:** This schedule isn't conflict serializable because there is a cycle in its data dependency graph. Moreover, this schedule is not conflict equivalent (every pair of conflicting operations is ordered in the same way) to any serial schedule of transaction execution.

iii. **[2 points]** Is this schedule view serializable?

■ **Yes** □ No

**Solution:** To determine if the schedule is view serializable, we need to check it against three criteria for view equivalence for original schedule $S_1$ and serial schedule $S_2$:

1. If $T_1$ reads the initial value of $A$ in $S_1$, then $T_1$ must also read the initial value of $A$ in $S_2$.
2. If $T_1$ reads a value of $A$ written by $T_2$ in $S_1$, then $T_1$ must also read that value of $A$ written by $T_2$ in $S_2$.
3. If $T_1$ writes the final value of $A$ in $S_1$, then $T_1$ must also write the final value of $A$ in $S_2$.

From our analysis, we found a possible serial schedule that is view-equivalent to the given schedule: $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$. This serial schedule fulfills the criteria for view equivalence:

- $T_4$ writes $B$ and $A$.
- $T_1$ reads the initial value of $C$ and the value of $A$ written by $T_4$.
- $T_3$ writes $B$, writes the final version of $C$, and reads the initial value of $D$.
- $T_2$ writes the final version of $A$, reads the initial version of $D$, and writes the final version of $B$.

Therefore, this serial schedule is view-equivalent to the original one, so it is proved to be view-serializable. Importantly, compared to conflict serializability, view serializability allows all conflict serializable schedules along with "blind writes", ensuring database consistency after all transactions are committed.

iv. **[6 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.

| | | |
|---|---|---|
| ■ $T_1 \rightarrow T_2$ | □ $T_2 \rightarrow T_3$ | □ $T_2 \rightarrow T_4$ |
| □ $T_2 \rightarrow T_1$ | ■ $T_3 \rightarrow T_2$ | ■ $T_4 \rightarrow T_2$ |
| ■ $T_1 \rightarrow T_3$ | □ $T_1 \rightarrow T_4$ | ■ $T_3 \rightarrow T_4$ |
| □ $T_3 \rightarrow T_1$ | ■ $T_4 \rightarrow T_1$ | □ $T_4 \rightarrow T_3$ |

**Solution:** The answer is:

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

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

v. **[2 points]** Is this schedule possible under regular 2PL?

□ Yes

■ **No**

**Solution:** This schedule is not possible under 2PL because it is not 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]** Update the type of all musical artists in table A with the name = 'Lunar Echoes' to 'Band'

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

**Solution:** The correct answer choice is IX(D), X(A). This is because we potentially need to modify records in table A where the artist's name is 'Lunar Echoes', and this choice accesses the intended exclusive parent lock to get the exclusive lock on those specific rows in table A.

- X(D) is incorrect because it gains an exclusive lock on the entire database D when it only needs to modify specific rows in table A.
- SIX(D), X(A) is incorrect because it gains a shared intention lock on the database D when it has no need to read the contents of D.
- IX(D), IX(A) is incorrect because it does not gain the exclusive lock on the specific rows in table A that match the condition (artist name is 'Lunar Echoes'), which it needs to modify the type of those artists.

(b) **[4 points]** Find the release record whose year equals the most recent year in all releases.

- ☐ S(D)
- ☐ IS(D), IS(R)
- ☐ IX(D), X(R)
- ■ IS(D), S(R)

**Solution:** The correct choice is `IS(D), S(R)`. The query needs to scan all records in `R` to find the release with the most recent year and scan all records again to find the release records that satisfy the equation condition. So, it requires a shared lock on `R`.

- `S(D)` is incorrect because it doesn't acquire the intended parent locks necessary to obtain the `S(R)` lock for reading the table `R`.
- `IS(D), IS(R)` is incorrect because it only states the intention to read-access the table R but doesn't actually acquire the lock to perform the read.
- `IX(D), X(R)` is incorrect because it accesses an intended exclusive lock, which is not needed for a read-only query.

(c) **[4 points]** Modify the $30^{th}$ record on `A12`.

☐ `IX(D), IX(A), IX(A12), IX(A12.30)`
■ `IX(D), IX(A), IX(A12), X(A12.30)`
☐ `SIX(D), SIX(A), SIX(A12), X(A12.30)`
☐ `IS(D), IS(A), IS(A12), X(A12.30)`

**Solution:** The correct choice is `IX(D), IX(A), IX(A12), X(A12.30)`. 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.

- `IX(D), IX(A), IX(A12), IX(A12.30)` is incorrect because it only gets the intention exclusive lock for the record.
- `SIX(D), SIX(A), SIX(A12), IX(A12.30)` 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 locks.
- `IS(D), IS(A), IS(A12), X(A12.30)` is incorrect because the DBMS intends to write to a tuple, so it should not grab the shared intention parent locks.

(d) **[4 points]** Scan all records in `R` and modify the $3^{rd}$ record on `R4`

■ `IX(D), SIX(R), IX(R4), X(R4.3)`
☐ `IS(D), S(R), IX(R4.3)`
☐ `S(D), IX(R), X(R4.3)`
☐ `IS(D), SIX(R), X(R4)`

**Solution:** The correct choice is `IX(D), SIX(R), IX(R4), X(R4.3)`. This choice is correct because it accesses all intended locks and the exclusive lock `X(R4.3)`. It also gains a shared intention lock on `R`, so it can read and modify records in `R`.

- `IS(D), S(R), IX(R4.3)` is incorrect because it accesses an intended shared parent lock for both `D` and `R`, when we plan to modify a record in `R`.
- `S(D), IX(R), X(R4.3)` is incorrect because we do plan on reading and modifying

in relation R, but we are only gaining a shared lock on the database level, which isn't sufficient.

- IS(D), SIX(R), X(R4) is incorrect because while it gains a shared intention lock for R and the database, it tries to exclusively lock the whole page R4 instead of just the specific record.

(e) **[4 points]** Scan all records in R and modify the $23^{rd}$ record on R7.

■ IX(D), SIX(R), IX(R7), X(R7.23)
□ IS(D), IS(R), IS(R7), X(R7.23)
□ SIX(D), SIX(R), IX(R7), X(R7.23)
□ IS(D), SIX(R), X(R7.23)

**Solution:** The correct choice is IX(D), SIX(R), IX(R7), X(R7.23). This choice is correct because it accesses all intended exclusive locks and the exclusive lock X(R7.23). It also gains a shared lock on R, so it can scan all the records in R.

- IS(D), IS(R), IS(R7), X(R7.23) is incorrect because we should not gain intended shared locks on —R— if we plan to modify a record in R.
- SIX(D), SIX(R), IX(R7), X(R7.23) is incorrect because we do not plan on reading the database, only relation R, so we should grab only an intention exclusive lock at the database level.
- IS(D), SIX(R), X(R7.23) is incorrect because we do not have the prerequisites to gain a SIX lock on R with the intended shared lock on database D. We needed an intended exclusive lock on D instead.

(f) **[4 points]** Two users are trying to access data. User A is scanning all the records in R to read, while User B is trying to modify the $23^{rd}$ record in A1. Which of the following sets of locks are most suitable for this scenario?

□ User A: SIX(D), S(R), User B: SIX(D), IX(A), X(A1.23)
□ User A: S(D), User B: X(D)
■ **User A: IS(D), S(R), User B: IX(D), IX(A), X(A1.23)**
□ User A: IS(D), S(R), User B: SIX(D), IX(A), X(A1.23)

**Solution:** The correct choice is User A: IS(D), S(R) and User B: IX(D), IX(A), X(A1.23). This choice is correct because:

- User A only intends to read all records in R. Thus, he acquires an intention shared lock on the database D and a shared lock on the table R.
- User B intends to modify a specific record in table A. He acquires an intention exclusive lock on the database D, an intention exclusive lock on the table A, and then an exclusive lock on the specific record A1.23.

Other choices are incorrect because:

- `User A: SIX(D), S(R)` and `User B: SIX(D), IX(A), X(A1.23)` is incorrect because User A doesn't need to acquire shared intention exclusive locks for a mere read operation, and neither does User B for a specific record modification.
- `User A: S(D)` and `User B: X(D)` is problematic as it locks the entire database either in shared mode or exclusive mode, preventing concurrent operations.
- `User A: IS(D), S(R)` and `User B: SIX(D), IX(A), X(A1.23)` is incorrect because while User A's locks are appropriate for his read operation, User B does not need a shared intention exclusive lock on the database for modifying a specific record.

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

Consider the following set of transactions accessing a database with object *A, B, C, D*. The questions below assume that the transaction manager is using **optimistic concurrency control** (OCC). Assume that 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.

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

You can assume that the DBMS is using the serial validation protocol discussed in class where only one transaction can be in the validation phase at a time, and each transaction is doing backward validation (i.e. Each transaction, when validating, checks whether it intersects its read/write sets with any transactions that have already committed. You may assume there are no other transactions in addition to the ones shown below. )

| time | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| 1 | | READ(A) | |
| 2 | | WRITE(A) | |
| 3 | READ(A) | | |
| 4 | WRITE(A) | | |
| 5 | | | READ(C) |
| 6 | READ(B) | | |
| 7 | READ(E) | | |
| 8 | VALIDATE? | | |
| 9 | | WRITE(B) | |
| 10 | WRITE? | | |
| 11 | | WRITE(C) | |
| 12 | | VALIDATE? | |
| 13 | | | WRITE(C) |
| 14 | | WRITE? | |
| 15 | | | READ(D) |
| 16 | | | WRITE(D) |
| 17 | | | VALIDATE? |
| 18 | | | 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 = 3, will $T_1$ read $A$ written by $T_2$?
□ Yes ■ **No**

**Solution:** No. 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 $T_2$'s write phase can see the value of $A$ written by $T_2$, provided $T_2$ commits successfully. Hence, $T_1$ at time 3 will not read the original $A$ that isn't written by $T_2$.

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

**Solution:** $T_1$ does not need to abort because no other transactions have been committed yet.

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

**Solution:** $T_2$ will abort because $T_2$'s read set intersects with with $T_1$'s write set. $T_2$ saw initial value of $A$ when time = 1, but it should read the value written by $T_1$. So $T_2$ should abort in its validation phase.

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

**Solution:** $T_3$ won't abort because $T_3$'s read and write set doesn't intersect with $T_1$'s read and write 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

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