CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2026)
PROF. JIGNESH PATEL

# Homework #5 (by Drew)
Due: **Sunday Nov 22, 2026 @ 11:59pm**

**IMPORTANT:**

- Enter all of your answers into **Gradescope by 11:59pm on Sunday Nov 22, 2026**.
- **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)
- Each part is all or nothing. There is no partial credit.

*Revision* : 2026/11/18 19:27

| Question | Points | Score |
|---|---|---|
| Serializability and 2PL | 30 | |
| Hierarchical Locking | 30 | |
| Optimistic Concurrency Control | 25 | |
| Multi-Version Concurrency Control | 15 | |
| Total: | 100 | |

## Question 1: Serialization and 2PL.....[30 points]

(a) True/False Questions:

i. **[2 points]** There are conflict-serializable schedules that are not view-serializable.
☐ True ☐ False

ii. **[2 points]** Using regular (i.e., not strong strict) 2PL does not guarantee a view serializable schedule.
☐ True ☐ False

iii. **[2 points]** 2PL is a pessimistic concurrency control protocol.
☐ True ☐ False

iv. **[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

v. **[2 points]** Conflict-serializable schedules prevent unrepeatable reads and dirty reads.
☐ True ☐ False

(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.)

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ |
|---|---|---|---|---|---|---|---|---|
| $T_1$ | | W(A) | | W(B) | | | | |
| $T_2$ | W(A) | | R(A) | | | | | |
| $T_3$ | | | | | R(B) | W(C) | | |
| $T_4$ | | | | | | | R(C) | W(D) |

Table 1: A schedule with 4 transactions

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

ii. **[8 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.

☐ $T_1 \rightarrow T_2$
☐ $T_2 \rightarrow T_1$
☐ $T_1 \rightarrow T_3$
☐ $T_3 \rightarrow T_1$
☐ $T_2 \rightarrow T_3$
☐ $T_3 \rightarrow T_2$
☐ $T_1 \rightarrow T_4$
☐ $T_4 \rightarrow T_1$
☐ $T_2 \rightarrow T_4$
☐ $T_4 \rightarrow T_2$
☐ $T_3 \rightarrow T_4$
☐ $T_4 \rightarrow T_3$

iii. **[3 points]** Is this schedule conflict serializable?
☐ Yes ☐ No

iv. **[3 points]** Is this schedule view serializable?
☐ Yes ☐ No

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

## Question 2: Hierarchical Locking . . . . . [30 points]

Consider a database `D` consisting of two tables `C` (which stores information about companies) and `P` (which stores information about products). Specifically:

- `C(cid, name, type, headquarters, ceo_gender, founded_year)`
- `P(pid, name, company_id, category, status, market, release_year, units_sold)`

Table `P` spans 2500 pages, which we denote `P1` to `P2500`. Table `C` spans 150 pages, which we denote `C1` to `C150`. Each page contains 160 records. We use the notation `P3.20` to denote the twentieth record on the third page of table `P`. 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 (`P` and `C`), page-level (e.g., `P10`), and record-level (e.g., `P10.42`). We use the notation `IS(D)` to mean a shared database-level intention lock, and `X(C2.20-C3.80)` 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 `C`.

For each of the following operations below, which given sequence of lock requests **maximize the potential for concurrency** while guaranteeing correctness?

(a) **[5 points]** Find the product record that has the largest `units_sold` where `release_year = 2001`.
- ☐ `IX(D), X(P)`
- ☐ `S(D)`
- ☐ `IS(D), IS(P)`
- ☐ `IS(D), S(P)`

(b) **[4 points]** Update the `type` of all companies in table C with the `name = 'MegaCorp'` to `'VerticallyIntegrated'`
- ☐ `IX(D), X(C)`
- ☐ `X(D)`
- ☐ `SIX(D), X(C)`
- ☐ `IX(D), IX(C)`

(c) **[5 points]** Increment the units_sold for the $159^{th}$ record on `P1440`.
- ☐ `IX(D), IX(P), IX(P1440), IX(P1440.159)`
- ☐ `IX(D), IX(P), IX(P1440), X(P1440.159)`
- ☐ `SIX(D), SIX(P), SIX(P1440), X(P1440.159)`
- ☐ `IS(D), IS(P), IS(P1440), X(P1440.159)`

(d) **[5 points]** Delete records in `C` if type = 'PrivateEquity'.
- ☐ `SIX(D), SIX(C)`
- ☐ `IX(D), IX(C)`
- ☐ `IX(D), X(C)`
- ☐ `SIX(D), X(C)`

Question 2 continues. . .

(e) **[5 points]** Scan all records between `P30` and `P260` and modify the $5^{th}$ record on `P75`
- ☐ `IS(D), S(P), IX(P75.5)`
- ☐ `IX(D), IX(P), IX(P30-P260), IX(P75), X(P75.5)`
- ☐ `IX(D), SIX(P), IX(P75), X(P75.5)`
- ☐ `IS(D), SIX(P), X(P75)`

(f) **[6 points]** Two users are trying to access data. User C is scanning all the records in `P` to read, while User B is trying to modify the $7^{th}$ record in `C16`. Which of the following sets of locks are most suitable for this scenario?
- ☐ `User C: SIX(D), S(P), User B: SIX(D), IX(C), IX(C16), X(C16.7)`
- ☐ `User C: S(D), User B: X(D)`
- ☐ `User C: IS(D), S(P), User B: SIX(D), IX(C), IX(C16), X(C16.7)`
- ☐ `User C: IS(D), S(P), User B: IX(D), IX(C), IX(C16), X(C16.7)`

## Question 3: Optimistic Concurrency Control . . . . . [25 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 | | READ(B) | |
| 3 | WRITE(B) | | |
| 4 | VALIDATE? | | |
| 5 | WRITE? | | |
| 6 | | WRITE(A) | |
| 7 | | | READ(C) |
| 8 | | | READ(A) |
| 9 | | READ(D) | |
| 10 | | VALIDATE? | |
| 11 | | WRITE? | |
| 12 | | | WRITE(D) |
| 13 | | | VALIDATE? |
| 14 | | | WRITE? |

Figure 1: An execution schedule

(a) **[3 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.

(b) **[3 points]** Will $T_1$ abort?
- ☐ Yes
- ☐ No

(c) **[3 points]** Will $T_2$ abort?
- ☐ Yes
- ☐ No

(d) **[3 points]** Will $T_3$ abort?
- ☐ Yes
- ☐ No

(e) **[3 points]** When time = 8, will $T_3$ read $A$ written by $T_2$?
☐ Yes ☐ No

(f) **[2 points]** In optimistic concurrency control (OCC), a transaction does not acquire locks on every object that it reads during its normal execution phase.
☐ True ☐ False

(g) **[2 points]** OCC works best when concurrent transactions access disjoint subsets of data in a database.
☐ True ☐ False

(h) **[2 points]** If contention is high, OCC may perform poorly because transactions can do substantial work before discovering that they must abort.
☐ True ☐ False

(i) **[2 points]** Transactions can suffer from *unrepeatable reads* in OCC.
☐ True ☐ False

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

## Question 4: Multi-Version Concurrency Control . . . . . [15 points]

Mark either **True** or **False** for each of the following statements.

Assume that the DBMS in each scenario supports multi-versioning of tuples.

(a) Consider a MVCC DBMS using **delta** version storage.

i. **[3 points]** If the DBMS uses *newest-to-oldest* (N2O) version chain ordering, all reclaimable versions will be found in the delta storage area.
☐ True ☐ False

ii. **[3 points]** When a transaction modifies an existing tuple's primary key attributes, the DBMS must delete the tuple and then re-insert it (with its new primary key) regardless of the version chain ordering.
☐ True ☐ False

(b) **[3 points]** Background vacuuming requires separate cleaning threads to be running alongside any worker threads for ongoing transactions.
☐ True ☐ False

(c) **[3 points]** In a MVCC DBMS, writers never wait to acquire exclusive locks on tuples and never get aborted.
☐ True ☐ False

(d) **[3 points]** Consider a MVCC DBMS using **append-only** version storage. The DBMS can use *cooperative garbage collection* with oldest-to-newest (O2N) version chain ordering only if it also uses logical pointers for secondary indexes.
☐ True ☐ False