



A11.7 Client server databases and concurrent access

Databases and big data · A level · OCR H446 1.3.2, AQA 7517 4.10.5,
Eduqas A500QS 2.5 · about 60 min

Name _____

Class _____

Date _____

What this lesson is about

Many clients, one database: the lost update, and the four ways to prevent it: record locks, serialisation, timestamp ordering and commitment ordering.

- Client server databases
- The lost update
- 1. Record locks
- 2. Serialisation
- 3. Timestamp ordering
- 4. Commitment ordering

Questions

6 marks in all

- Two clients read a score of 10. One writes 12, then the other writes 13. What is this problem called? [1 mark]
 - ☐ A A lost update
 - ☐ B A deadlock
 - ☐ C A deletion anomaly
 - ☐ D An orphan record
- Transaction A has locked record X and waits for Y; transaction B has locked Y and waits for X. What is this? [1 mark]
 - ☐ A Deadlock
 - ☐ B Serialisation
 - ☐ C A lost update
 - ☐ D Timestamp ordering
- In timestamp ordering, transaction 3 tries to write a record whose read timestamp is 5. What happens? [1 mark]
 - ☐ A Transaction 3 is aborted, because a younger transaction has already read the record
 - ☐ B The write goes ahead and the write timestamp becomes 3
 - ☐ C Transaction 3 waits for transaction 5 to finish
 - ☐ D Transaction 5 is aborted

4. What does this print?

[1 mark]

```
read_ts, write_ts, aborted = {}, {}, set()
schedule = [(1, "read", "x"), (2, "write", "x"), (1, "write", "x"), (3, "read", "x")]
for ts, op, rec in schedule:
    if ts in aborted:
        result = "skipped"
    elif op == "read":
        result = "abort" if ts < write_ts.get(rec, 0) else "ok"
        if result == "ok":
            read_ts[rec] = max(read_ts.get(rec, 0), ts)
    else:
        late = ts < read_ts.get(rec, 0) or ts < write_ts.get(rec, 0)
        result = "abort" if late else "ok"
        if result == "ok":
            write_ts[rec] = ts
    if result == "abort":
        aborted.add(ts)
    print(f"T{ts} {op}: {result}")
```

5. Which are advantages of a client server database over separate copies of the data on each computer? [1 mark]

Tick every answer that is true.

- ☐ A Every client sees the same, up-to-date data
- ☐ B Integrity rules and access rights are enforced in one place
- ☐ C The server can never be a single point of failure
- ☐ D Concurrent updates can never cause problems

6. How does serialisation prevent lost updates?

[1 mark]

- ☐ A It makes transactions act as if they ran one after another, never overlapping
- ☐ B It stores every update as a new timestamped fact
- ☐ C It converts the data to JSON before saving
- ☐ D It deletes transactions that run at the same time

The task: timestamp ordering

`schedule` is a list of operations in the order they reach the DBMS. Each is `(ts, op, record)`: the transaction's timestamp (a whole number from 1 to 5), `"read"` or `"write"`, and the record's name. Apply timestamp ordering exactly as described above. Every record's read and write timestamps start at 0. For each operation, in order, print `T<ts> <op> <record>: <result>`, where result is: - `skipped` if transaction `ts` has already been aborted (do not restart it; its later operations are skipped); - `abort` if the rules refuse the operation (the transaction is now aborted); - `ok` otherwise, updating the record's timestamp. Then print `committed:` followed by the transactions that were never aborted, and `aborted:` followed by those that were, each as `T<ts>` in increasing order, separated by a comma and a space: for example `aborted: T1, T2, T4`. Fourteen lines in all. The robot does not move.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

# (transaction timestamp, operation, record)
schedule = [
    (1, "read", "balls"),
    (2, "read", "balls"),
    (2, "write", "balls"),
    (1, "write", "balls"),
    (3, "write", "score"),
    (1, "read", "score"),
    (2, "read", "score"),
    (3, "read", "balls"),
    (2, "write", "score"),
    (3, "write", "balls"),
    (5, "write", "led"),
    (4, "write", "led"),
]
```

Plan your program here, then type it in and press Run.



Do it on the robot

www.bugbotlab.com/learn/a11-7-client-server-and-concurrency/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Draw the timeline for the deadlock between A and B, with the time each lock is taken and requested.
2. Run the lost update with locks, but let B lock first. What is the final score?
3. In the task schedule, T4 is aborted only because T5 wrote `led` first. Does that write actually conflict with anything T4 has done? (Look up the Thomas write rule.)

