1 Which pair of read write operations is a conflict for Multi Version Protocols?

Only read-write because if the reader should come after the writer in order of the read version but comes in earlier in time, the writer has to be aborted.

2 Does the notion of strict 2V2PL and strong 2V2PL make sense? Justify!

Strict 2V2PL: 2V2PL is effectively strict 2V2PL by default. Before commit the commit locks on all write locks needs to be acquired. Therefore for all transactions that write all locks (including all write locks) are held until commit and then released.

Strong 2V2PL: This makes sense as it makes read-only transactions behave differently than “normal” 2V2PL. I don’t think is has any practical advantages though.

3 You were tasked to implement a garbage collection for the ROMV protocol. State a sufficient condition for deciding whether a version is eligible for garbage collection.

Its timestamp is older than the timestamp of the oldest active read-only transaction and it is not the most recent committed version.

4 Why doesn’t it need to implement versioning from scratch? What is the original reason that versioning is exist?

It was implemented because in case there is any unexpected event happens to this version, it can restore or roll back to the last version.

5 Is it possible to use MVCC for recovery purposes? Why?

Since MVCC techniques are based on storing older versions of data items, we maintain the history of the db items. While MVCC techniques require more storage, this “disadvantage” can be used to address recoverability issue.
6 Explain the need for multi version concurrency control algorithm.

Distinct transactions could be given distinct versions of the same data item to read or to over-
write. Maintaining multiple versions can help making concurrency control more flexible. Also
beneficial for long read transactions that execute concurrently with writes.

7 Explain which operations in MVCC pose a conflict, and why (or
why not respectively)?

rr: no conflict, everyone reads the same item
ww: no conflict, everyone writes his one version
wr: no conflict, commuting the pair can only restrict the version choice for the read
rw: conflict, commuting the pair creates more possible choices for the read, which can lead to
errors

8 What is a monoversion schedule?

When a multiversion scheduler uses a version function that maps each read t the last preceding
write (on the same data item).

9 Show that 2V2PL is suspectible to deadlocks.

We show it by finding an example that leads to a deadlock.

\[ s = r_1(x) \ w_1(x) \ r_2(x) \ w_2(y) \ r_1(y) \ w_2(x) \ c_2 \ w_1(y) \ c_1 \]

T_2 waits for T_1 to acquire \( wr_2(x) \), while T_1 waits for T_2 on \( wr_1(y) \). Deadlock.

10 What is the advantage of MVTO in scenarios in which values are
often overwritten compared to 2V2PL?

MVTO doesn’t impose restrictions preventing many blind-writes from occurring. In 2V2PL those
can lead to deadlocks that might be unwarranted.

11 ROMV: For the schedule \( s \), give the resulting schedule under

ROMV. \( s = r_1(x) \ w_1(x) \ r_2(x) \ w_2(y) \ r_1(y) \ w_2(x) \ c_2 \ w_1(y) \ c_1 \)

\( r_1(x_0) \ w_1(x_1) \ r_1(y_0) \ r_2(x_1) \ w_1(y_1) \ c_1 \ w_2(y_2) \ w_2(x_2) \ c_2 \).

12 How many different versions of a data item can exist at the same
time when using 2V2PL and why?

At most 2. Since two different transactions can never write on the same item at the same time
since two write locks are not allowed there is always the last committed version and at most one
additional uncommitted one.
13 While implementing a real-world Transaction System, is it practical to have unlimited versions of data items? What issues may arise if an upper-bound is imposed on the number of versions?

While theoretically it is tempting to use unlimited versions of data items for generalizing the monoversion case, in practice it is unreasonable to have an unlimited number of versions in the system. Disk space or memory is not infinite, for one, and managing an unlimited number of versions of various data items can be very inefficient. For this reason, an upper-limit is often imposed. However, imposing an upper-bound has its consequences. Some histories that are multiversion serializable without any limit, may lose this property if a limit is given. This must be taken into account.

14 Give the example of history that it is both in MVSR and MCSR and prove.

\[ w_0(x_0) w_0(y_0) c_0 w_1(x_1) w_2(y_1) c_1 r_2(y_2) c_2 \]

MVSR: assume \( x_0 \ll x_1 \) and \( y_0 \ll y_1 \).
MCSR: conflict graph is empty.