4
Concurrency Protocols – Part 2: MVCC

Reading: [Kl], chapter 7; [Ha] chapter 9; [Me] chapter 4

2PL

T1	T2
Begin	
Read (A)	Begin
	Read (A)
	Commit
Read (A)	
Commit	

Would this schedule run under 2PL?

yes, only shared lock is needed

2PL

T1	T2
Begin	
Read (A)	Begin
	Write (A)
	Commit
Read (A)	
Commit	

Would this schedule run under 2PL?

T1 gets a shared lock, T2 needs exclusive lock

once shared lock is given no other transaction can get exclusive lock(recall compatibility matrix)
So T2 waits until T1 continues and commits, after commit lock is released and T2 can acquire x-lock. So there is no rollback but T2 has to wait

MVCC

there is a lot of waiting involved in 2PL. We only have no waiting when both transaction reads. As soon as even 1 transaction starts to write there is waiting involved.

in MVCC only write operations block write operations.

2PL

Object A	T2 Read	T2 Write
T1 Read	✓	X
T1 Write	X	X

MVCC

Object A	T2 Read	T2 Write
T1 Read	✓	✓
T1 Write	✓	✗

MVCC: decouple read and write operations:

- Read operations do not block write operations.
- Write operations do not block read operations

MVCC

T1	T2
Begin	
Read (A)	Begin
	Write (A)
	Commit
Read (A)	
Commit	

T1 also writes but in temporary table and that doesn't count here

```
/*  T1 */
Begin;
drop table if exists transaction_log;
CREATE TEMPORARY TABLE transaction_log (message_text
varchar(50), t_payment_value INT);
INSERT INTO transaction_log (message_text, t_payment_value)
select 'Payment_amount',t_payment FROM teacher WHERE t_id = 1;  20
/* T2  */
Begin;
update teacher set t_payment = t_payment + 10 where t_id = 1;  30
commit;
/*  T1 */
INSERT INTO transaction_log (message_text, t_payment_value)
select 'Payment_amount',t_payment
                FROM teacher WHERE t_id = 1;  30
commit;
```

so after first read of T1 we get 20, then T2 updates it to 30 and commits. since T2 commited T1 can see this change since we are in default isolation level and when T1 reads second time we get updated version which is 30. If we read T1 second time without T2 being commited then we would always get 20 because T2 has not completed updating and it keeps lock.

- What read results do we get if T2 commits after second read of T1?
- What read results do we get when we run T1 in isolation level repeatable read? 20

Multiversion Concurrency Control (MVCC)

→ When a transaction writes an object, a new version of the object is created.

→ When a transaction reads an object,

- → it reads the latest committed version that existed when the transaction started (repeatable read)
- → it reads the latest committed version that existed when the select statement started (read committed)

Visibility Rules

Read Transactions Repeatable Read

1. Any writes made by transactions that are active (not yet committed or aborted) - when the transaction starts - are ignored. Any writes that those transactions have made are ignored, even if the transactions subsequently commit.
2. Any writes made by aborted transactions are ignored.
3. Any writes made by transactions with a later transaction ID (i.e., which started after the current transaction started) are ignored, regardless of whether those transactions have committed.
4. All other writes are visible to the transactions' queries.

Read Transactions Read Committed

1. Any writes made by transactions with an older transaction ID that are active but not yet committed - when the transaction starts - are ignored. If the transactions subsequently commit, the writes become visible to the subsequent queries.
2. Any writes made by aborted transactions are ignored.
3. Any writes made by transactions with a later transaction ID (i.e., which started after the current transaction started) are visible to subsequent queries after those transactions have committed.
4. All other writes are visible to the transactions' queries.

Write short scripts that test / verify these rules.

eva.knirsch@kiu.edu.ge

PostgreSQL MVCC Repeatable Read

Xmin

created by T1

Xmax

0 doesn't mean deleted it's just valid

T10	T11
Begin	
Read(A)	Begin
	Write(A)
	Commit
Read(A)	
Commit	

Version	Value	Created_by	Deleted_by
A0	100	01	0
A1	200	11	0

PostgreSQL MVCC Repeatable Read

T10	T11
Begin	
Read(A)	Begin
	Write(A)
	Commit
Read(A)	
Commit	

Version	Value	Created_by	Deleted_by
A0	100	01	11
A1	200	11	0

Is the write of T11 an insert or an update?

it's update, done by insertion or creation of new version

insert would mean starting with version 0

PostgreSQL MVCC Repeatable Read

T10	T11
Begin	
Read(A)	Begin
	Write(A)
	Commit
Read(A)	
Commit	

Version	Value	Created_by	Deleted_by
A0	100	01	11
A1	200	11	0

T10

What would T1 read in 2nd read in Read Committed?

it would have read 200

PostgreSQL MVCC Repeatable Read

T10	T11
Begin	
Read(A)	
Write(A)	
	Begin
	Read(A)
Read(A)	
Commit	
	Commit

Version	Value	Created_by	Deleted_by
A0	100	01	1 0
A1	50	10	0

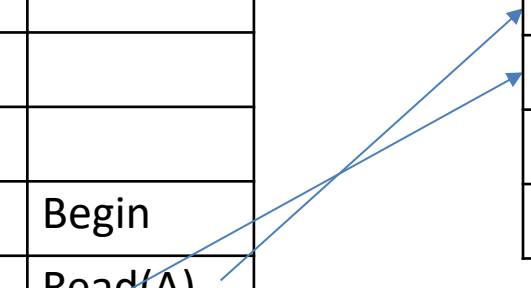
EXCEPTION!

as T10 itself updated it is allowed to read new version

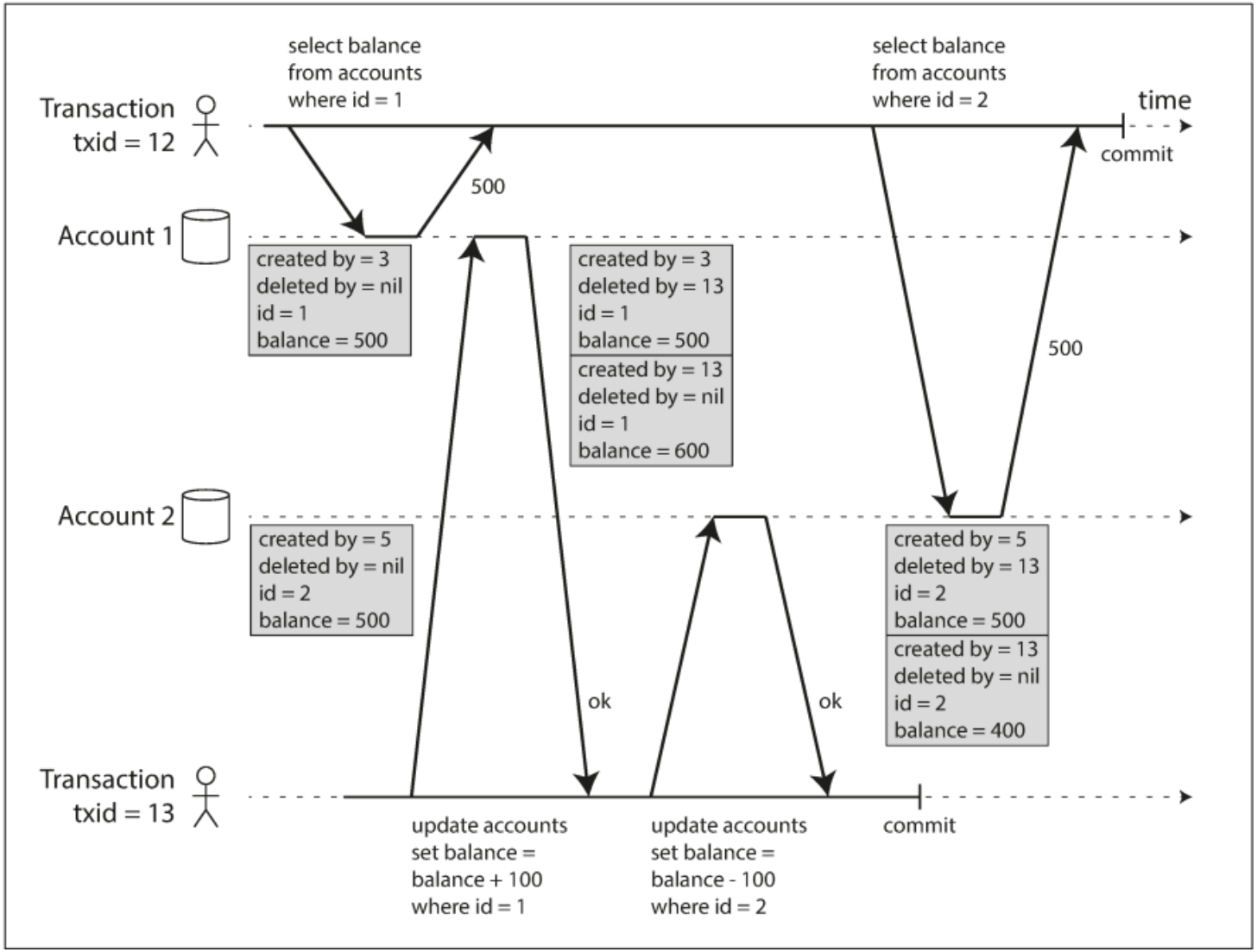
MVCC

T10	T11
Begin	
Read(A)	
Write(A)	
	Begin
	Read(A)
Read(A)	
Commit	
	Commit

Version	Value	Created_by	Deleted_by
A0	**100**	**01**	**10**
A1	**50**	**10**	**∞**



MVCC



[Kl], p.240

eva.knirsch@kiu.edu.ge

MVCC - Update

```
/*  session 1 */
Begin;
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 1;
/*  session 2 */
begin;
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 1;
/*  session 1 */
update teacher set t_payment = t_payment + 10 where t_id = 1;
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 1;
select txid_current();
/*  session 2 */
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 1;
/* returns the old value AND the old CTID but already marked as deleted with the current Xmax */
/*  session 1 */
commit;
/*  session 2 */
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 1;
/* returns the new value AND the new CTID and new Xmin and Xmax
```


eva.knirsch@kiu.edu.ge

MVCC - Delete

```
/*  session 1 */
Begin;
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 23;
/*  session 2 */
begin;
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 23;
/*  session 1 */
delete from teacher where t_id = 23;
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 23;
/* transaction does not see tuple anymore */
/*  session 2 */
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 23;
/* returns the "marked for deletion" tuple
/*  session 1 */
commit;
/*  session 2 */
select ctid, t_id, t_payment, t_name, Xmin, Xmax from teacher where t_id = 1;
/* empty return table */
```


eva.knirsch@kiu.edu.ge

MVCC - writes

Operation	Description	Version	Value	Created_by Xmin	Deleted_by Xmax
Insert	create a version, created_by: TS (Ti), deleted_by: 0	A0	x	TS (Ti)	0
Delete	Mark for deletion	A0	x	TS (Ti)	TS(Td)
Update	delete and create	A0	x	TS (Ti)	TS(Tu)
		A1	y	TS(Tu)	0

Garbage Collection – in PostgreSQL called Vacuum Process: Needs to run regularly in order to remove old versions and remove deleted objects (usually rows).

```
SELECT * FROM pgstattuple('teacher'::regclass);
/* returns number of dead tuples in table teacher */
```


eva.knirsch@kiu.edu.ge

Mariadb Expand

MariaDB Expand row version format

Record Type
NULL-column indicator
Length of non-NULL variable length column values
Next Record Offset
Key
row-store of non-key / non-null fields
TransactionID
InvocationID
CommitID
Roll-Back Pointer (Log-Sequence-NumberID)

ID	Description
xID	is generated when the transaction begins (TS Start)
iID	is generated when a statement begins execution within the transaction (TS start statement)
cID	is generated when the transaction is committed (TS commit)

What requirement holds in regard to the cID in order to safely remove the row version?
Database keeps a list with all currently active transaction xIDs.

https://mariadb.com/docs/xpand/architecture/components/xpand/concurrency/mvcc/

eva.knirsch@kiu.edu.ge

MVCC – Write – Write Conflict

T1	T2
Begin	
Read(A)	
Write(A)	
	Begin
	Read(A)
	Write(A)
Read(A)	
Commit	
	Commit

Version	Value	Created_by	Deleted_by
A0	**100**	**00**	**01**
A1	**50**	**01**	**∞**

Additional rules are needed to solve write-write conflicts under MVCC

MVCC does NOT provide serializability.

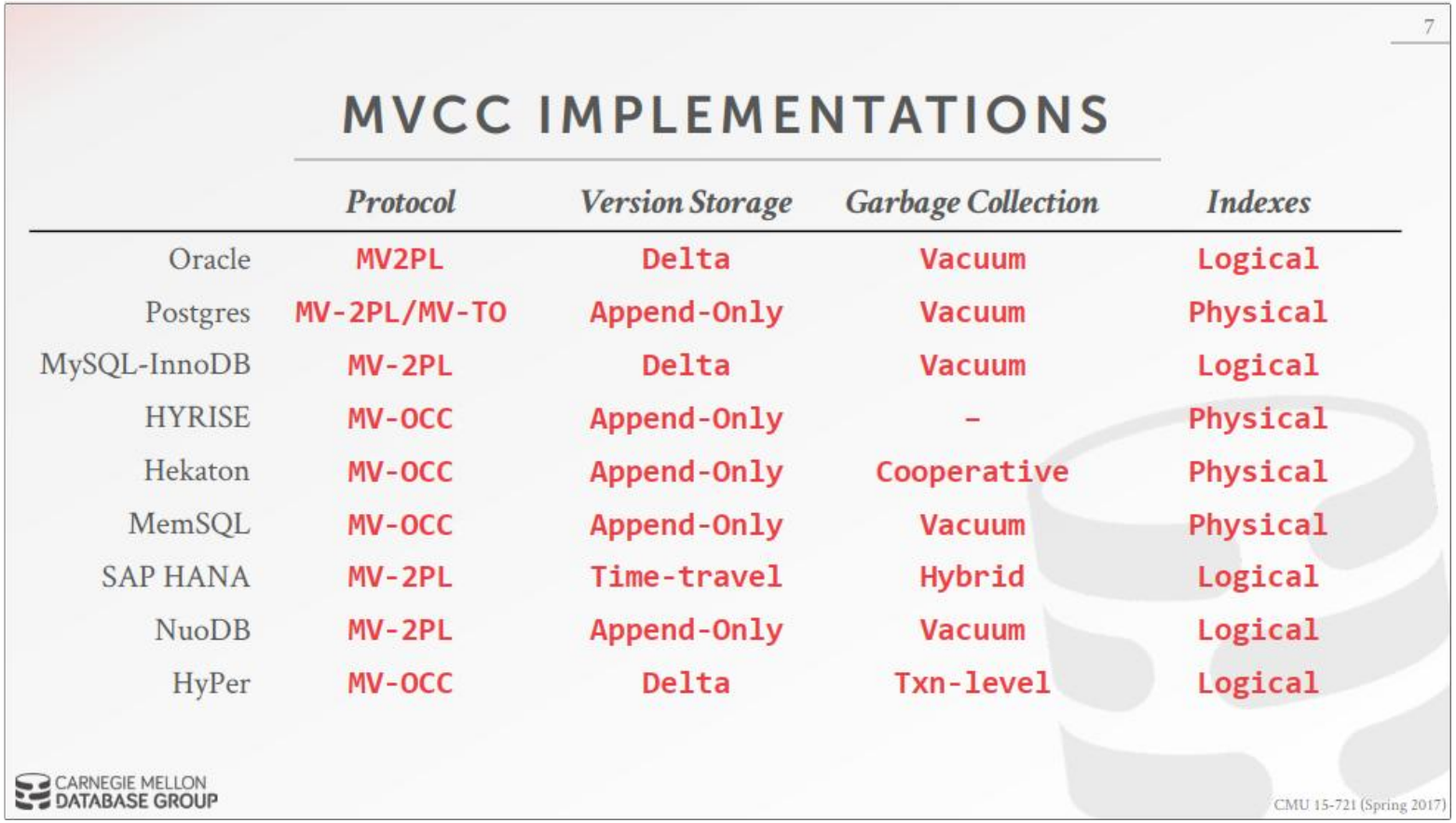


Table is not up-to-date as of 2025 and not correct. But it shows that MVCC needs another protocol to complement it.

eva.knirsch@kiu.edu.ge