CS3223 Tutorial 11

Gary Lim

Chapter Review

Recovery

- Undo/Redo logging
- Checkpointing

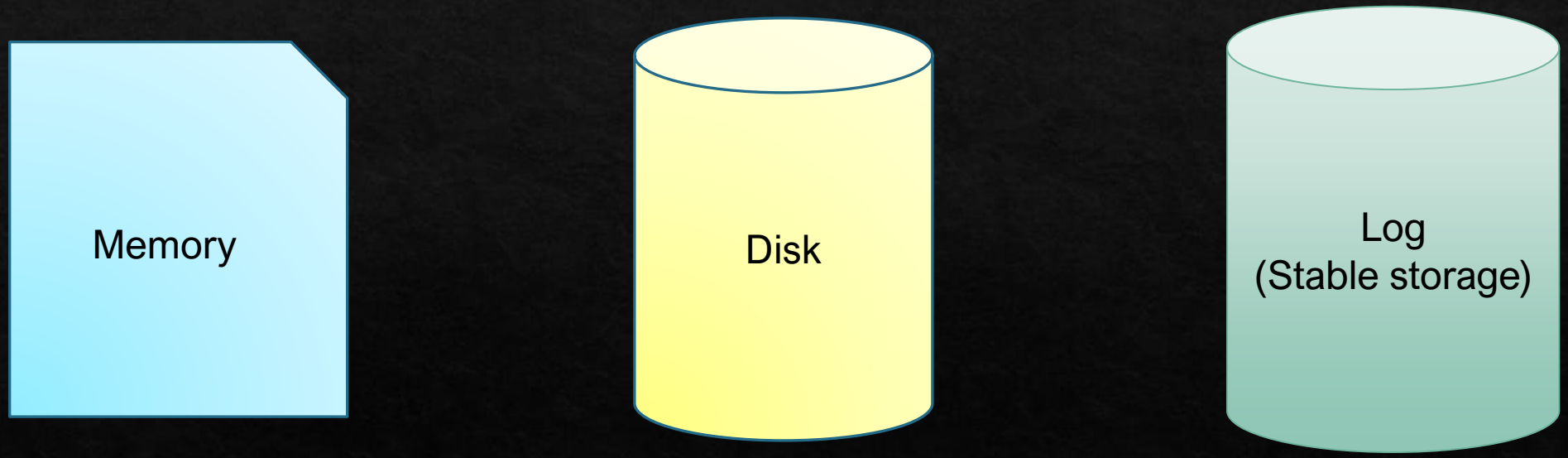
Recovery

A.C.I.D.

- Atomicity
  - **All** actions in a Xact happen, or **none** of them happen
- Durability
  - If a Xact commits, its effects **persist**

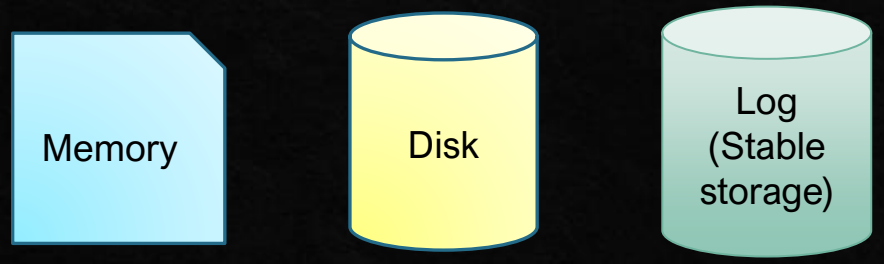
Logging

- Note the distinction between the disk (for data) and the stable storage for log records



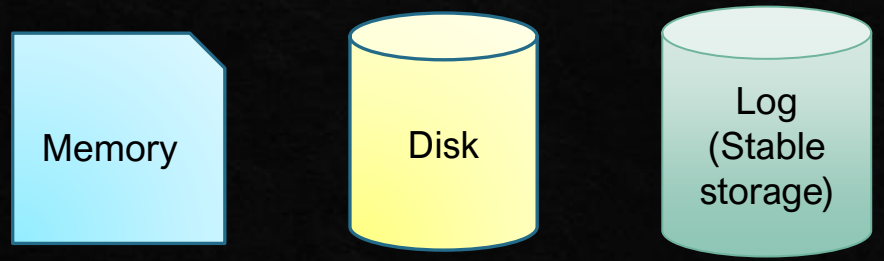
Undo Logging

- Allows **undoing** of actions of Xacts that **have not been committed**
- Rules:
  1. For every action generate undo log record with **old** value of object
  2. Before object is **modified** on disk, log record pertaining to object must be written out
  3. Before commit log is written out, **all writes of Xact** must be reflected on disk



Redo Logging

- Allows **redoing** of actions of Xacts that **have been committed**
- Rules:
  1. For every action generate redo log record with **new** value of object
  2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out



Logging

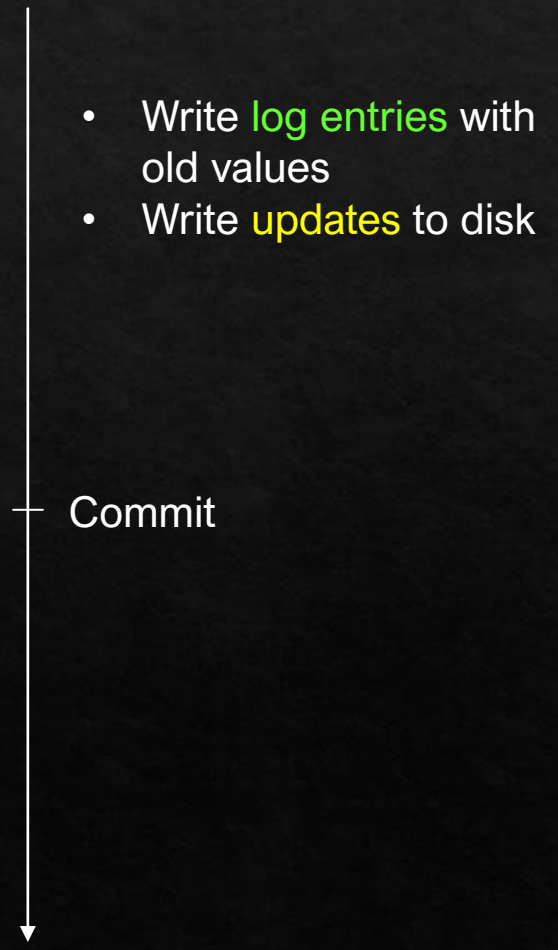
	Before T commits…	Logging Type
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	Logging Type
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO

Logging

	Before T commits…	Logging Type
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	Logging Type
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO



- Write log entries with old values
- Write updates to disk

Logging

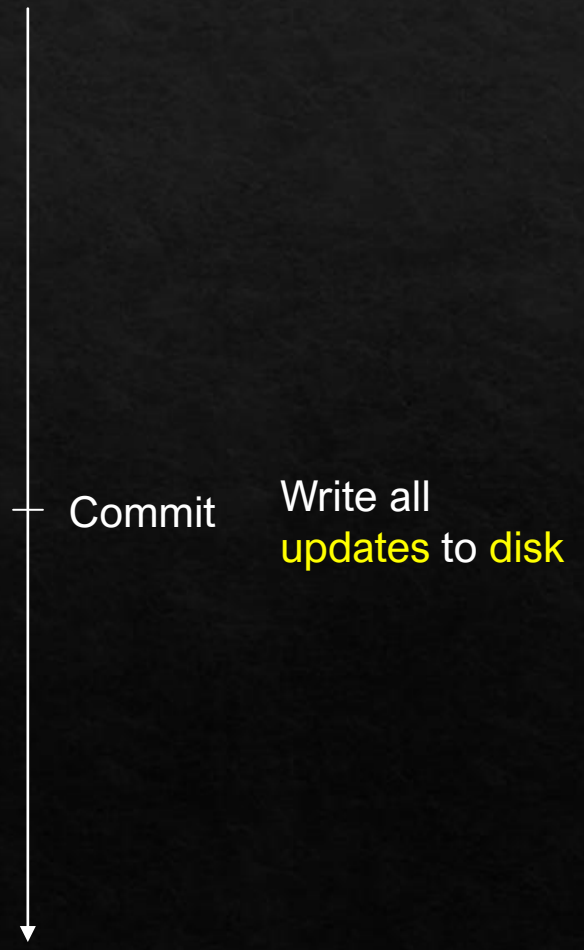
	Before T commits…	Logging Type
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	Logging Type
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO

Logging

	Before T commits…	Logging Type
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	Logging Type
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO



Logging

	Before T commits…	Logging Type
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	Logging Type
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO

Write all log entries with new values and commit log

Logging

Guarantee for **Undo** Logging:

- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all updates would be written to disk
- Failure before commit: Perform UNDO action
- Failure after commit: Nothing to be done

	Before T commits…	**Logging Type**
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	**Logging Type**
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO

- Write log entries with old values
- Write updates to disk

Commit — Write all updates to disk

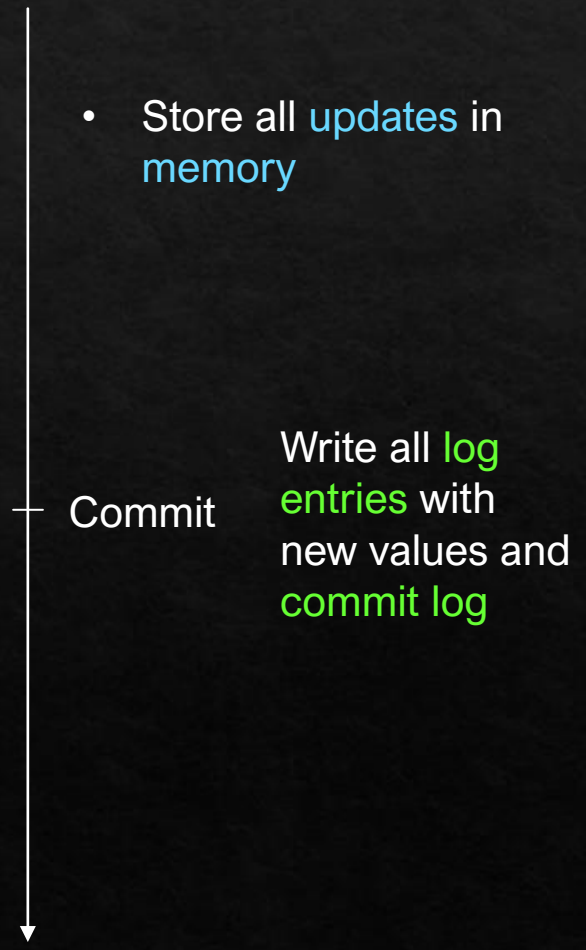
Logging

Guarantee for **Redo** Logging:

- Before commit, updates **are not** written to disk
- By the time commit log is written out, all redo log entries would be written to stable storage,
- Failure before commit: Nothing to be done
- Failure after commit: Perform REDO action

	Before T commits…	**Logging Type**
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	**Logging Type**
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO



Logging

Guarantee for **Undo/Redo** Logging:

- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all undo and redo log entries would be written out
- Failure before commit: Perform UNDO action
- Failure after commit: Perform REDO action

	Before T commits…	**Logging Type**
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	**Logging Type**
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO

- Write log entries with old values
- Write updates to disk

Commit

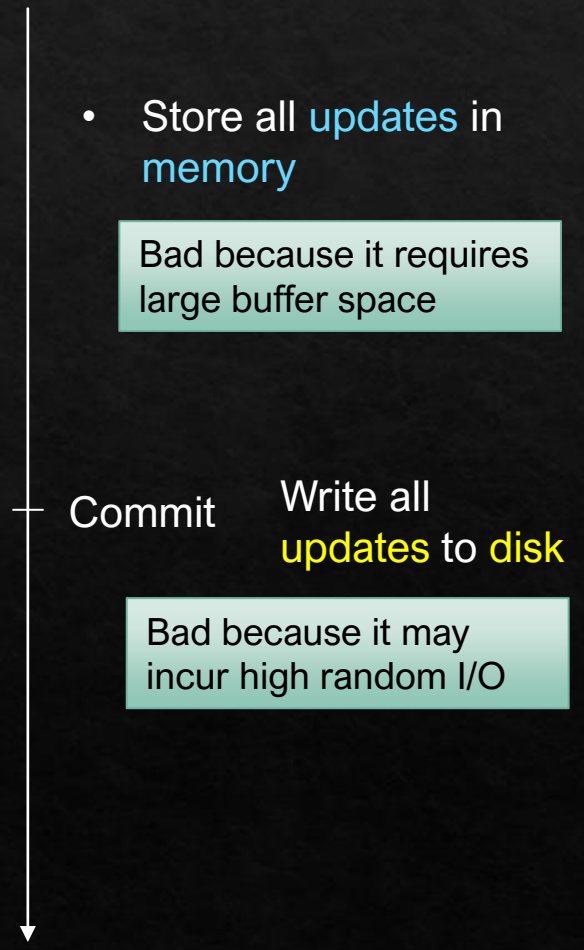
Write all log entries with new values and commit log

Logging

Guarantee for **no undo, no redo**:
- Before commit, updates **are not** written to disk
- By the time commit log is written out, all updates would be written to disk
- Failure before commit: Nothing to be done
- Failure after commit: Nothing to be done

	Before T commits…	**Logging Type**
STEAL	Write dirty pages to disk (must rmb old value)	UNDO
NO STEAL	Don't write dirty pages to disk (dirty pages are buffered until commit time)	NO UNDO

	Upon time of commit	**Logging Type**
FORCE	Write all dirty pages to disk	NO REDO
NO FORCE	Don't need to write all dirty pages (must rmb new value)	REDO



- Store all updates in memory

Bad because it requires large buffer space

Bad because it may incur high random I/O

Checkpointing

- A flag to determine how far back we need to apply the undo or redo logs
- Provides a GUARANTEE that
  - All actions before the START checkpoint have been written to disk
    (means for UNDO logging, need to UNDO them if Xact did not commit b4 crash)
  - All xacts that committed before the START checkpoint can be ignored
    (their actions are all reflected on disk already)

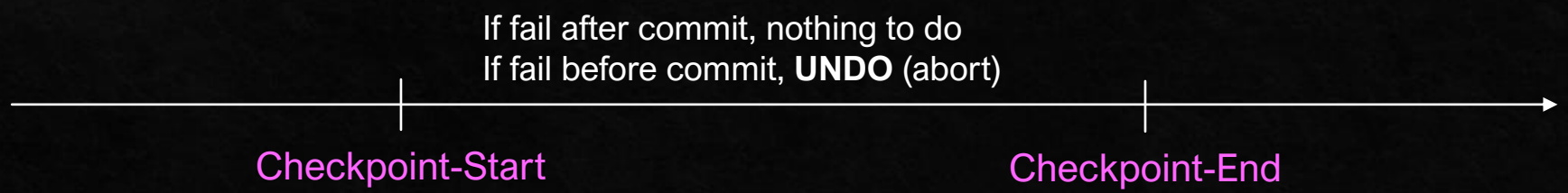
Checkpointing

- Undo Logging
- At Checkpoint Start
  - Ignore all Xacts that have committed before this point
- Active Xacts:
  - Started before checkpoint-start and not committed
  - Started after checkpoint-start

guarantee updates prior to checkpoint start are already reflected on disk

Guarantee for **Undo** Logging:

- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all updates would be written to disk
- Failure before commit: Perform UNDO action
- Failure after commit: Nothing to be done

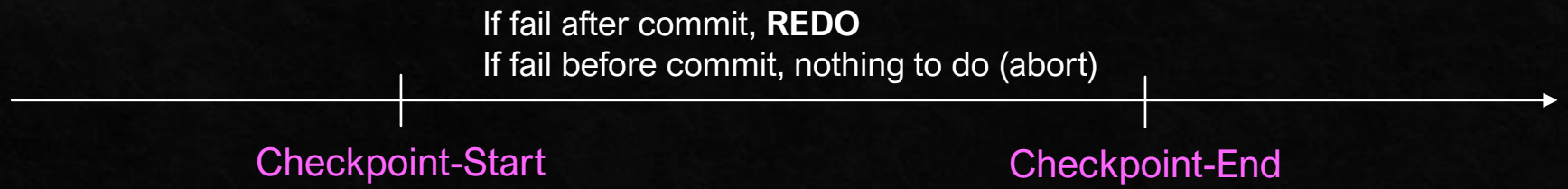


Checkpointing

- Redo Logging
- At Checkpoint Start
  - Ignore all Xacts that have committed before this point
- Active Xacts:
  - Started before checkpoint-start and not committed
  - Started after checkpoint-start

guarantee updates prior to checkpoint start are already reflected on disk

Guarantee for **Redo** Logging:
- Before commit, updates **are not** written to disk
- By the time commit log is written out, all redo log entries would be written to stable storage,
- Failure before commit: Nothing to be done
- Failure after commit: Perform REDO action

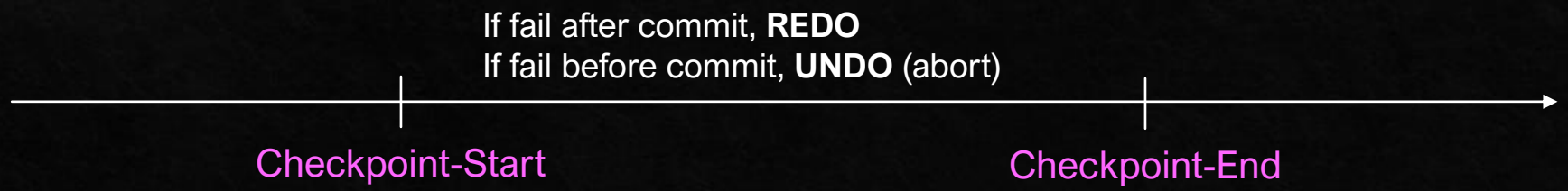


Checkpointing

- Undo/Redo Logging
- At Checkpoint Start
  - Ignore all Xacts that have committed before this point
- Active Xacts:
  - Started before checkpoint-start and not committed
  - Started after checkpoint-start

guarantee updates prior to checkpoint start are already reflected on disk

Guarantee for **Undo/Redo** Logging:
- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all undo and redo log entries would be written out
- Failure before commit: Perform UNDO action
- Failure after commit: Perform REDO action



Q1

- Assuming UNDO logging, given the following sequence of log records, enumerate all possible sequence of events

Log Records	Events on Disk
<START T>	
<T, A, 10>	A: 10 → 20
<T, B, 20>	B: 20 → 30
<T, C, 30>	C: 30 → 40
<COMMIT T>	

For illustration purposes, assume the action is incrementing by 10

Q1

- Assuming UNDO logging, given the following sequence of log records, enumerate all possible sequence of events

Log Records		Events on Disk	
<START T>	**START**		
<T, A, 10>	**LA**	A: 10 → 20	**A**
<T, B, 20>	**LB**	B: 20 → 30	**B**
<T, C, 30>	**LC**	C: 30 → 40	**C**
<COMMIT T>	**COM**		

For illustration purposes, assume the action is incrementing by 10

Q1

- Assuming UNDO logging, given the following sequence of log records, enumerate all possible sequence of events

Rules for UNDO logging
2. Before object is **modified** on disk, log record pertaining to object must be written out
3. Before commit log is written out, **all writes of Xact** must be reflected on disk

Log Records		**Events on Disk**	
<START T>	**START**		
<T, A, 10>	**LA**	A: 10 → 20	**A**
<T, B, 20>	**LB**	B: 20 → 30	**B**
<T, C, 30>	**LC**	C: 30 → 40	**C**
<COMMIT T>	**COM**		

Q1

- Assuming UNDO logging, given the following sequence of log records, enumerate all possible sequence of events

Rules for UNDO logging
2. Before object is **modified** on disk, log record pertaining to object must be written out
3. Before commit log is written out, **all writes of Xact** must be reflected on disk

Log Records		**Events on Disk**	
<START T>	**START**		
<T, A, 10>	**LA**	A: 10 → 20	**A**
<T, B, 20>	**LB**	B: 20 → 30	**B**
<T, C, 30>	**LC**	C: 30 → 40	**C**
<COMMIT T>	**COM**		

- **LA → A**
- **LB → B**
- **LC → C**
- **COM** is at the end

Q1

- Assuming UNDO logging, given the following sequence of log records, enumerate all possible sequence of events

Rules for UNDO logging
2. Before object is **modified** on disk, log record pertaining to object must be written out
3. Before commit log is written out, **all writes of Xact** must be reflected on disk

Sequence	**Number of ways**
START, LA, A, LB, B, LC, C, COM	1
START, LA, A, LB, LC, B, C, COM	2! (swap B and C)
START, LA, LB, A, LC, B, C, COM	2! (swap B and C)
START, LA, LB, A, B, LC, C, COM	2! (swap A and B)
START, LA, LB, B, LC, A, C, COM	2! (swap A and C)
START, LA, LB, LC, A, B, C, COM	3! (swap A, B and C)

15 possible sequences

- **LA → A**
- **LB → B**
- **LC → C**
- **COM** is at the end

Q2

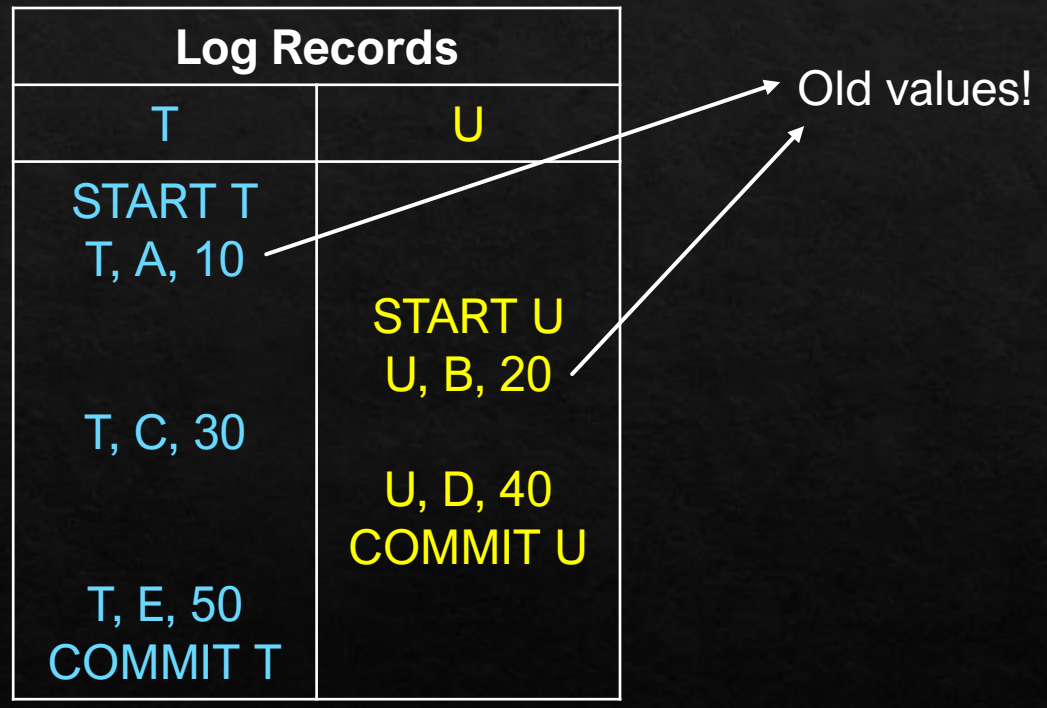
Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) <T, E, 50>

d) <COMMIT T>



Q2a

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) **<START U>**

b) <COMMIT U>

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10	START U
Not committed!	Not committed!

Changes to disk

1. Undo <T, A, 10> by writing A=10

Q2a

It is possible that the update of A didn't get written on disk. Thus A can either be the old value of 10 or new value.

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) **<START U>**

b) <COMMIT U>

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10	START U
Not committed!	Not committed!

Changes to disk

1. Undo <T, A, 10> by writing A=10

	Before Recovery	After Recovery
A	10 / New	10
B	Old	Old
C	Old	Old
D	Old	Old
E	Old	Old

Q2b

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) **<COMMIT U>**

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 Not committed!	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Undo <T, C, 30> by writing C=30
2. Undo <T, A, 10> by writing A=10

Q2b

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) **<COMMIT U>**

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 Not committed!	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Undo <T, C, 30> by writing C=30
2. Undo <T, A, 10> by writing A=10

	Before Recovery	After Recovery
A	10 / New	10
B	New	New
C	30 / New	30
D	New	New
E	50	50

Log record <T, E, 50> not written out yet, therefore E is definitely not updated on disk yet

Q2c

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) **<T, E, 50>**

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 Not committed!	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Undo <T, E, 50> by writing E=50
2. Undo <T, C, 30> by writing C=30
3. Undo <T, A, 10> by writing A=10

Q2c

Rules for UNDO logging
2. Before object is **modified** on disk, log record pertaining to object must be written out
3. Before commit log is written out, **all writes of Xact** must be reflected on disk

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) **<T, E, 50>**

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50	START U U, B, 20 U, D, 40 COMMIT U

Not committed!

Changes to disk

1. Undo <T, E, 50> by writing E=50
2. Undo <T, C, 30> by writing C=30
3. Undo <T, A, 10> by writing A=10

Nothing to do for xact U because all its writes have already been updated on disk

Q2c

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) **<T, E, 50>**

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 Not committed!	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Undo <T, E, 50> by writing E=50
2. Undo <T, C, 30> by writing C=30
3. Undo <T, A, 10> by writing A=10

	Before Recovery	After Recovery
A	10 / New	10
B	New	New
C	30 / New	30
D	New	New
E	50 / New	50

Q2d

Rules for UNDO logging
2. Before object is **modified** on disk, log record pertaining to object must be written out
3. Before commit log is written out, **all writes of Xact** must be reflected on disk

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) <T, E, 50>

d) **<COMMIT T>**

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 COMMIT T	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Nothing (undo logging ensures that by time commit log is flushed, all changes would **already have been written to disk**)

Q2d

Rules for UNDO logging
2. Before object is **modified** on disk, log record pertaining to object must be written out
3. Before commit log is written out, **all writes of Xact** must be reflected on disk

Assuming **undo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) <T, E, 50>

d) **<COMMIT T>**

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 COMMIT T	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Nothing (undo logging ensures that by time commit log is flushed, all changes would **already have been written to disk**)

	Before Recovery	After Recovery
A	New	New
B	New	New
C	New	New
D	New	New
E	New	New

Q3

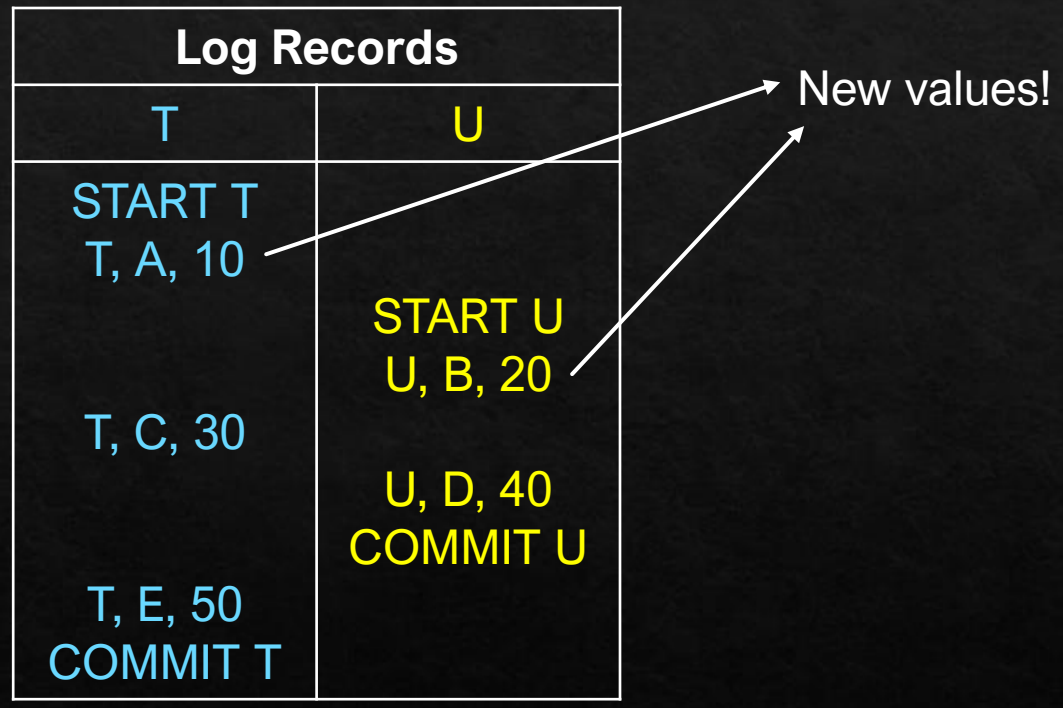
Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) <T, E, 50>

d) <COMMIT T>



Q3a

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) **<START U>**

b) <COMMIT U>

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10	START U
Not committed!	Not committed!

Changes to disk

1. Nothing (object is not modified on disk yet)

Q3a

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) **<START U>**

b) <COMMIT U>

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10	START U
Not committed!	Not committed!

Changes to disk

1. Nothing (object is not modified on disk yet)

	Before Recovery	After Recovery
A	Old	Old
B	Old	Old
C	Old	Old
D	Old	Old
E	Old	Old

Q3b

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) **<COMMIT U>**

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30	START U U, B, 20 U, D, 40 COMMIT U

Not committed!

Changes to disk

1. Redo <U, B, 20> by writing B=20
2. Redo <U, D, 40> by writing D=40

Q3b

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) **<COMMIT U>**

c) <T, E, 50>

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 Not committed!	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Redo <U, B, 20> by writing B=20
2. Redo <U, D, 40> by writing D=40

	Before Recovery	After Recovery
A	Old	Old
B	Old / 20	20
C	Old	Old
D	Old / 40	40
E	Old	Old

Q3c

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) **<T, E, 50>**

d) <COMMIT T>

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 Not committed!	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Redo <U, B, 20> by writing B=20
2. Redo <U, D, 40> by writing D=40

	Before Recovery	After Recovery
A	Old	Old
B	Old / 20	20
C	Old	Old
D	Old / 40	40
E	Old	Old

Q3d

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) <T, E, 50>

d) **<COMMIT T>**

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 COMMIT T	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Redo <T, A, 10> by writing A=10
2. Redo <U, B, 20> by writing B=20
3. Redo <T, C, 30> by writing C=30
4. Redo <U, D, 40> by writing D=40
5. Redo <T, E, 50> by writing E=50

Q3d

Rules for REDO logging
2. Before object is **modified** on disk, ALL log records including commit log, pertaining to object must be written out

Assuming **redo logging** is used, describe the actions of the recovery manager, including changes to both disk and log, if there is a crash and the last record on disk is:

a) <START U>

b) <COMMIT U>

c) <T, E, 50>

d) **<COMMIT T>**

Log Records	
T	U
START T T, A, 10 T, C, 30 T, E, 50 COMMIT T	START U U, B, 20 U, D, 40 COMMIT U

Changes to disk

1. Redo <T, A, 10> by writing A=10
2. Redo <U, B, 20> by writing B=20
3. Redo <T, C, 30> by writing C=30
4. Redo <U, D, 40> by writing D=40
5. Redo <T, E, 50> by writing E=50

	Before Recovery	After Recovery
A	Old / 10	10
B	Old / 20	20
C	Old / 30	30
D	Old / 40	40
E	Old / 50	50

Q4

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- Log entries: <Xact ID, Object, Old val, New val>
- Objects: X=0, Y=0 (initially)

	T_1	T_2	T_3
IC1	T_1 Start T_1, X, 0, 10	T_2 Start **T_2, ?, ?, 20**	
	Checkpoint Start		
IC2	**?, Y. ?, 30** T_1, Y, 30, 40	**?, Y. ?, 30**	T_3 Start
	Checkpoint End		
IC3	T_1 commit		**T_3, X, ?, 100**
	CRASH		

Q4a

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- What are possible combinations of IC1, IC2 and IC3? (Concept being tested: 2PL)

	T_1	T_2	T_3
IC1	T_1 Start T_1, X, 0, 10	T_2 Start **T_2, ?, ?, 20**	
	Checkpoint Start		
IC2	**?, Y. ?, 30** T_1, Y, 30, 40	**?, Y. ?, 30**	T_3 Start
	Checkpoint End		
IC3	T_1 commit		**T_3, X, ?, 100**
	CRASH		

Q4a

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- What are possible combinations of IC1, IC2 and IC3? (Concept being tested: 2PL)
  - IC2 cannot be a write by T_3 since T_3 has not started

List all possibilities:
IC1: either X or Y
IC2: either T1 or T2

	I	II	III	IV
IC1	Y, 0	Y, 0	X, 10	X, 10
IC2	T_1, 20	T_2, 20	T_1, 0	T_2, 0
IC3	10	10	20	20

	T_1	T_2	T_3
IC1	T_1 Start T_1, X, 0, 10	T_2 Start **T_2, ?, ?, 20**	
	Checkpoint Start		
IC2	**?, Y. ?, 30** T_1, Y, 30, 40	**?, Y. ?, 30**	T_3 Start
	Checkpoint End		
IC3	T_1 commit		**T_3, X, ?, 100**
	CRASH		

Q4a

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- What are possible combinations of IC1, IC2 and IC3? (Concept being tested: 2PL)
  - IC2 cannot be a write by T_3 since T_3 has not started
  - Not possible: T_2 write X in IC1 **AND** T_2 write Y in IC2 (forces T_1 to release X for T_2, when T_1 have not acq Y)

	I	II	III	IV
IC1	Y, 0	Y, 0	X, 10	X, 10
IC2	T_1, 20	T_2, 20	T_1, 0	T_2, 0
IC3	10	10	20	20

The rest are okay

	T_1	T_2	T_3
	T_1 Start	T_2 Start	
IC1	T_1, X, 0, 10	**T_2, X, 10, 20**	
	Checkpoint Start		
IC2	T_1, Y, 30, 40	**T_2, Y. 0, 30**	T_3 Start
	Checkpoint End		
IC3	T_1 commit		**T_3, X, ?, 100**
	CRASH		

Q4b

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- Based on the log entries in part (a), list all possible values of And B on disk **before** recovery.

	I	II	III
IC1	Y, 0	Y, 0	X, 10
IC2	T_1, 20	T_2, 20	T_1, 0
IC3	10	10	20

	T_1	T_2	T_3
IC1	T_1 Start T_1, X, 0, 10	T_2 Start **T_2, ?, ?, 20**	
	Checkpoint Start		
IC2	**?, Y. ?, 30** T_1, Y, 30, 40	**?, Y. ?, 30**	T_3 Start
	Checkpoint End		
IC3	T_1 commit		**T_3, X, ?, 100**
	CRASH		

Q4b

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- Based on the log entries in part (a), list all possible values of And B on disk **before** recovery.

	I	II	III
IC1	**Y, 0**	Y, 0	X, 10
IC2	**T_1, 20**	T_2, 20	T_1, 0
IC3	**10**	10	20

	X	Y
Log I	10, 100	20, 30, 40
Log II		
Log III		

Guarantee for **Undo/Redo** Logging:
- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all undo and redo log entries would be written out
- Failure before commit: Perform UNDO action
- Failure after commit: Perform REDO action

	T_1	T_2	T_3
IC1	T_1 Start **T_1, X, 0, 10**	T_2 Start **T_2, Y, 0, 20**	
	Checkpoint Start		
IC2	T_1, Y. 20, 30 T_1, Y, 30, 40		T_3 Start
	Checkpoint End		
IC3	T_1 commit		T_3, X, 10, 100
	CRASH		

guarantee updates prior to **checkpoint start** are already reflected on disk

Q4b

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- Based on the log entries in part (a), list all possible values of And B on disk **before** recovery.

	I	**II**	III
IC1	Y, 0	**Y, 0**	X, 10
IC2	T_1, 20	**T_2, 20**	T_1, 0
IC3	10	**10**	20

	X	Y
Log I	10, 100	20, 30, 40
Log II	10, 100	20, 30, 40
Log III		

Guarantee for **Undo/Redo** Logging:
- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all undo and redo log entries would be written out
- Failure before commit: Perform UNDO action
- Failure after commit: Perform REDO action

	T_1	T_2	T_3
IC1	T_1 Start **T_1, X, 0, 10**	T_2 Start **T_2, Y, 0, 20**	
	Checkpoint Start		
IC2	T_1, Y, 30, 40	T_2, Y. 20, 30	T_3 Start
	Checkpoint End		
IC3	T_1 commit		T_3, X, 10, 100
	CRASH		

Q4b

Guarantee for **Undo/Redo** Logging:

- Before commit, updates may or may not be written to disk
- By the time commit log is written out, all undo and redo log entries would be written out
- Failure before commit: Perform UNDO action
- Failure after commit: Perform REDO action

- Consider a DBMS that uses **undo/redo logging** with **non-quiescent checkpoints** and **basic 2PL protocol**
- Based on the log entries in part (a), list all possible values of And B on disk **before** recovery.

	I	II	**III**
IC1	Y, 0	Y, 0	**X, 10**
IC2	T_1, 20	T_2, 20	**T_1, 0**
IC3	10	10	**20**

	X	Y
Log I	10, 100	20, 30, 40
Log II	10, 100	20, 30, 40
Log III	20, 100	0, 30, 40

	T_1	T_2	T_3
IC1	T_1 Start T_1, X, 0, 10	T_2 Start **T_2, X, 10, 20**	
	Checkpoint Start		
IC2	T_1, Y. 0, 30 T_1, Y, 30, 40		T_3 Start
	Checkpoint End		
IC3	T_1 commit		T_3, X, 10, 100
	CRASH		

End

Final Tutorial!

- Yay, you guys made it!

Final Tutorial!

- Yay, you guys made it!
- Finals
  - 1pm – 3pm, Wed, 27 Apr 2022
  - MPSH6
  - Open-book, Lectures 7 to 13
  - Bring calculator, rough paper for working
- Revision
  - OPTIONAL - Gradiance HW6 and HW7
  - Consultation - Email me and propose a few timeslots

Concluding Thoughts

Taken from Sean Low

Concluding Thoughts

Advanced Database Modules

CS4224 Distributed Databases

a. Is ACID still applicable in a distributed context?
b. How are transactions managed in a distributed context?
c. What are the ways to partition data to maintain data integrity?
d. Master-Replica and Master-Master replication
e. NoSQL and NewSQL Project :)

Concluding Thoughts

Advanced Database Modules

CS4225 Big Data Systems for Data Science

a. How do we store and process large amounts of data
b. How we study new algorithms and systems for big data processing via batches, graphs and streams
c. How MapReduce came about and how it brought about new systems for big data processing.

Concluding Thoughts

Advanced Database Modules

CS4221 Database Applications Design and Tuning

a. How query optimization helps us to create faster or slower queries
b. How functional dependencies and normalization work practically
c. How NoSQL and XML technologies came about and their algorithms
d. Data Warehousing techniques for OLAP applications

Concluding Thoughts

Important Concepts for Backend Interviews (Database) not covered in this module but covered in CS4224

1. Explain Database Partitioning (Horizontal and Vertical)
2. What is CAP theorem in distributed databases? (Consistency, Availability, Presence of Partitions)?

Thank you!