



# Database Management Systems

## Module 36: Recovery/1

**Partha Pratim Das**

*Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur*

[ppd@cse.iitkgp.ernet.in](mailto:ppd@cse.iitkgp.ernet.in)

**Srijoni Majumdar  
Himadri B G S Bhuyan  
Gurunath Reddy M**



**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
[www.db-book.com](http://www.db-book.com)



# Week 07 Recap

- **Module 31: Transactions/1**
  - Transaction Concept
  - Transaction State
  - Concurrent Executions
- **Module 32: Transactions/2: Serializability**
  - Serializability
  - Conflict Serializability
- **Module 33: Transactions/3: Recoverability**
  - Recoverability and Isolation
  - Transaction Definition in SQL
  - View Serializability
  - Complex Notions of Serializability
- **Module 34: Concurrency Control/1**
  - Lock-Based Protocols
  - Implementing Locking
- **Module 35: Concurrency Control/2**
  - Deadlock Handling
  - Timestamp-Based Protocols

NPTEL



# Module Objectives

- We need to understand what are the possible sources for failure for transactions in a database
- Various types of storages are used for recovery from failures to ensure Atomicity, Consistency and Durability – these models need to be explored
- To understand recovery scheme based on logging
- To focus on single transactions only

NPTEL



# Module Outline

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

NPTEL



- **Failure Classification**
  - Storage Structure
  - Recovery and Atomicity
  - Log-Based Recovery

NPTEL

# FAILURE CLASSIFICATION



# Database System Recovery

- All database reads/writes are within a transaction
- Transactions have the “ACID” properties
  - Atomicity - all or nothing
  - Consistency - preserves database integrity
  - Isolation - execute as if they were run alone
  - Durability - results are not lost by a failure
- Concurrency control guarantees I, contributes to C
- Application program guarantees C
- Recovery subsystem guarantees A & D, contributes to C



# Failure Classification

- **Transaction failure:**

- **Logical errors:** transaction cannot complete due to some internal error condition
- **System errors:** the database system must terminate an active transaction due to an error condition (for example, deadlock)

- **System crash:** a power failure or other hardware or software failure causes the system to crash

- **Fail-stop assumption:** non-volatile storage contents are assumed to not be corrupted as result of a system crash
  - ▶ Database systems have numerous integrity checks to prevent corruption of disk data

- **Disk failure:** a head crash or similar disk failure destroys all or part of disk storage

- Destruction is assumed to be detectable
  - ▶ Disk drives use checksums to detect failures



# Recovery Algorithms

- Consider transaction  $T_i$  that transfers \$50 from account A to account B
  - Two updates: subtract 50 from A and add 50 to B
- Transaction  $T_i$  requires updates to A and B to be output to the database
  - A failure may occur after one of these modifications have been made but before both of them are made
  - Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state
  - Not modifying the database may result in lost updates if failure occurs just after transaction commits
- Recovery algorithms have two parts
  1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
  2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability



- Failure Classification
- **Storage Structure**
- Recovery and Atomicity
- Log-Based Recovery

# STORAGE STRUCTURE



# Storage Structure

## ■ **Volatile storage:**

- does not survive system crashes
- examples: main memory, cache memory

## ■ **Nonvolatile storage:**

- survives system crashes
- examples: disk, tape, flash memory, non-volatile (battery backed up) RAM
- but may still fail, losing data

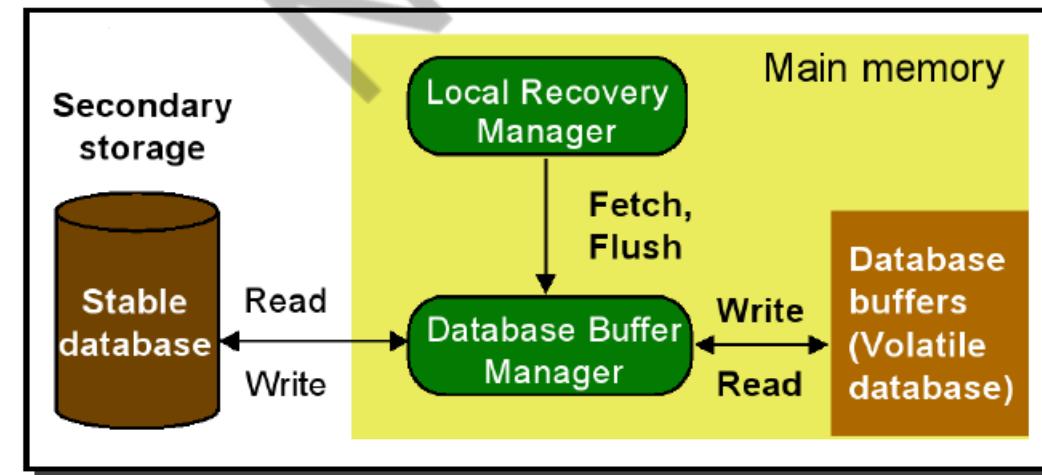
## ■ **Stable storage:**

- a mythical form of storage that survives all failures
- approximated by maintaining multiple copies on distinct nonvolatile media



# Stable-Storage Implementation

- Maintain multiple copies of each block on separate disks
  - copies can be at remote sites to protect against disasters such as fire or flooding
- Failure during data transfer can still result in inconsistent copies. Block transfer can result in
  - Successful completion
  - Partial failure: destination block has incorrect information
  - Total failure: destination block was never updated
- Protecting storage media from failure during data transfer (one solution):
  - Execute output operation as follows (assuming two copies of each block):
    1. Write the information onto the first physical block
    2. When the first write successfully completes, write the same information onto the second physical block
    3. The output is completed only after the second write successfully completes





# Stable-Storage Implementation (Cont.)

Protecting storage media from failure during data transfer (cont.):

- Copies of a block may differ due to failure during output operation. To recover from failure:
  1. First find inconsistent blocks:
    1. *Expensive solution:* Compare the two copies of every disk block
    2. *Better solution:*
      - Record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk)
      - Use this information during recovery to find blocks that may be inconsistent, and only compare copies of these
      - Used in hardware RAID systems
  2. If either copy of an inconsistent block is detected to have an error (bad checksum), overwrite it by the other copy
  3. If both have no error, but are different, overwrite the second block by the first block



# Data Access

- **Physical blocks** are those blocks residing on the disk
- **System buffer blocks** are the blocks residing temporarily in main memory
- Block movements between disk and main memory are initiated through the following two operations:
  - **input( $B$ )** transfers the physical block  $B$  to main memory
  - **output( $B$ )** transfers the buffer block  $B$  to the disk, and replaces the appropriate physical block there
- We assume, for simplicity, that each data item fits in, and is stored inside, a single block

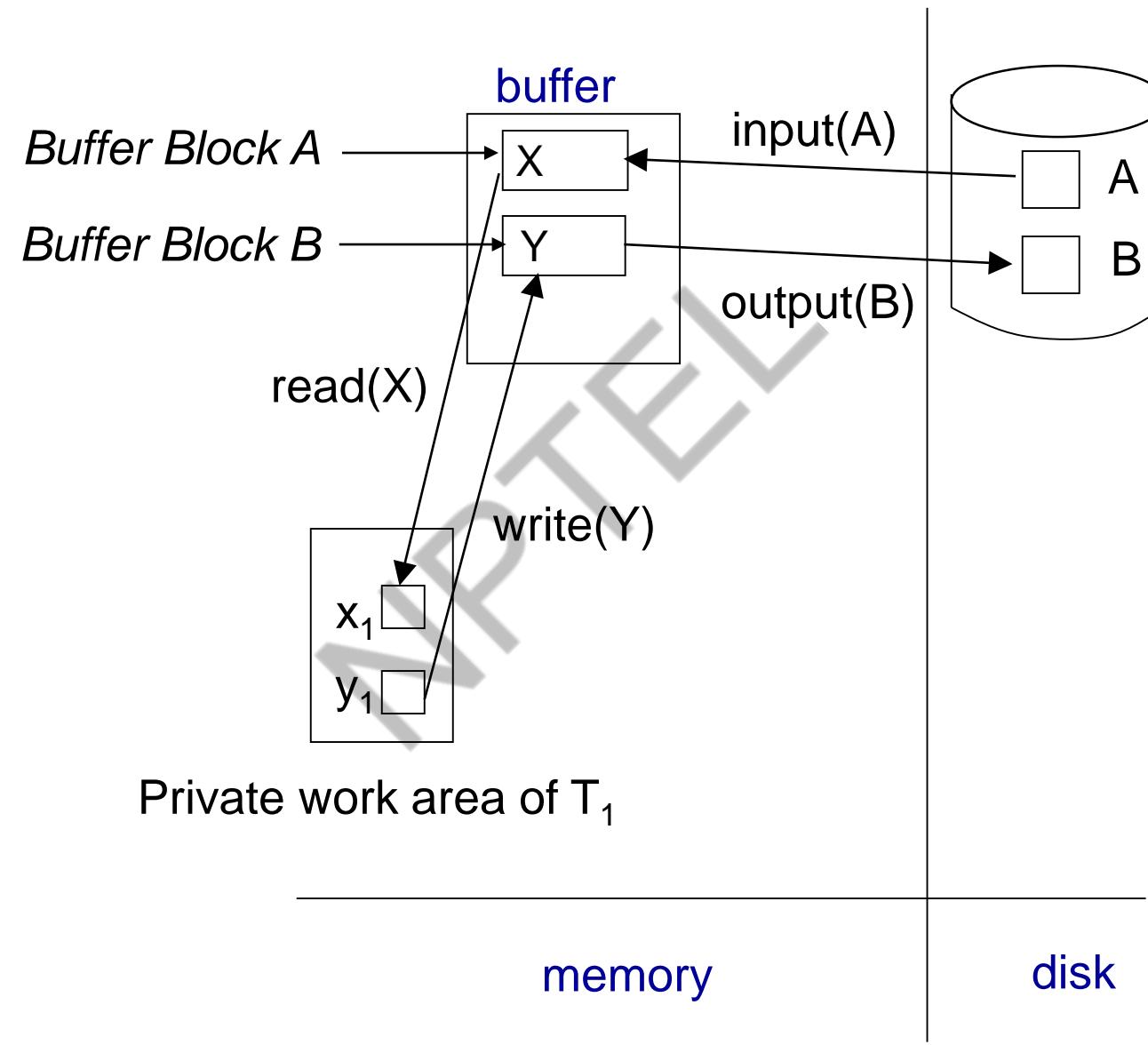


# Data Access (Cont.)

- Each transaction  $T_i$  has its private work-area in which local copies of all data items accessed and updated by it are kept
  - $T_i$ 's local copy of a data item  $X$  is denoted by  $x_i$
  - $B_X$  denotes block containing  $X$
- Transferring data items between system buffer blocks and its private work-area done by:
  - **read( $X$ )** assigns the value of data item  $X$  to the local variable  $x_i$
  - **write( $X$ )** assigns the value of local variable  $x_i$  to data item  $\{X\}$  in the buffer block
- Transactions
  - Must perform **read( $X$ )** before accessing  $X$  for the first time (subsequent reads can be from local copy)
  - The **write( $X$ )** can be executed at any time before the transaction commits
- Note that **output( $B_X$ )** need not immediately follow **write( $X$ )**. System can perform the **output** operation when it deems fit



# Example of Data Access





- Failure Classification
- Storage Structure
- **Recovery and Atomicity**
- Log-Based Recovery

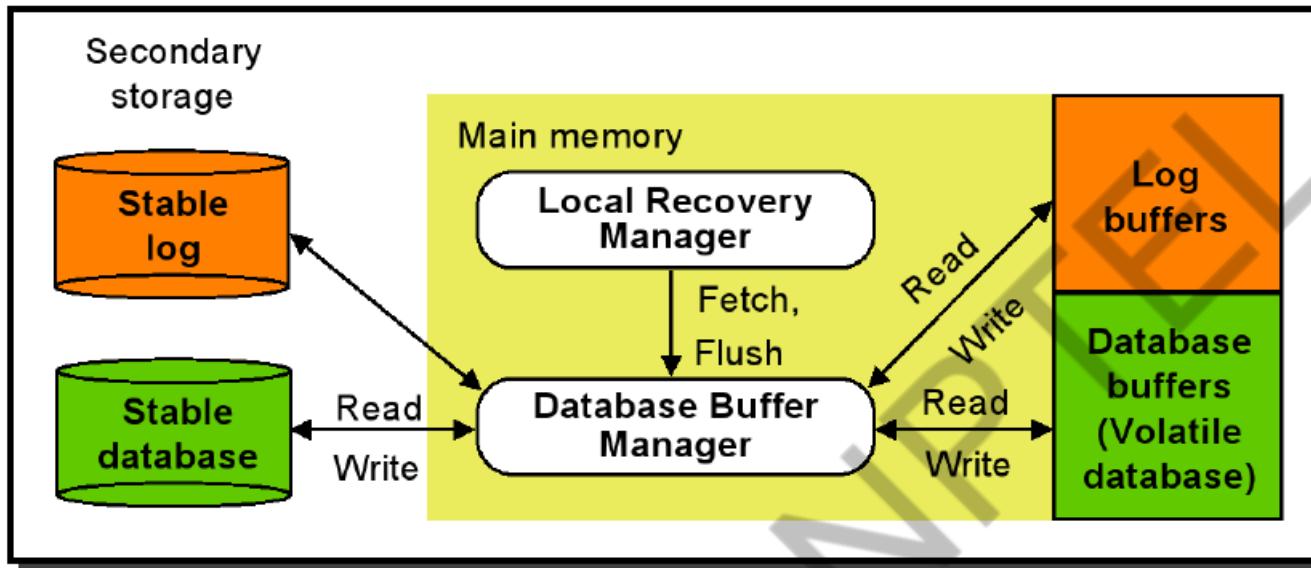
NPTEL

# RECOVERY AND ATOMICITY



# Recovery and Atomicity

- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself
- We study **log-based recovery mechanisms** in detail
  - We first present key concepts
  - And then present the actual recovery algorithm
- Less used alternative: **shadow-paging**
- In this Module we assume serial execution of transactions
- In the next Module, we consider the case of concurrent transaction execution



- Failure Classification
- Storage Structure
- Recovery and Atomicity
- **Log-Based Recovery**

# LOG-BASED RECOVERY



# Log-Based Recovery

- A **log** is kept on stable storage
  - The log is a sequence of **log records**, which maintains information about update activities on the database
- When transaction  $T_i$  starts, it registers itself by writing a record  
 $\langle T_i \text{ start} \rangle$   
to the log
- Before  $T_i$  executes **write**( $X$ ), a log record  
 $\langle T_i, X, V_1, V_2 \rangle$   
is written, where  $V_1$  is the value of  $X$  before the write (the **old value**), and  $V_2$  is the value to be written to  $X$  (the **new value**)
- When  $T_i$  finishes its last statement, the log record  $\langle T_i \text{ commit} \rangle$  is written
- Two approaches using logs
  - Immediate database modification
  - Deferred database modification



# Database Modification

- The **immediate-modification** scheme allows updates of an uncommitted transaction to be made to the buffer, or the disk itself, before the transaction commits
- Update log record must be written **before** a database item is written
  - We assume that the log record is output directly to stable storage
- Output of updated blocks to disk storage can take place at any time before or after transaction commit
- Order in which blocks are output can be different from the order in which they are written
- The **deferred-modification** scheme performs updates to buffer/disk only at the time of transaction commit
  - Simplifies some aspects of recovery
  - But has overhead of storing local copy
- We cover here only the immediate-modification scheme



# Transaction Commit

- A transaction is said to have committed when its commit log record is output to stable storage
  - All previous log records of the transaction must have been output already
- Writes performed by a transaction may still be in the buffer when the transaction commits, and may be output later

NPTEL



# Immediate Database Modification Example

## Log

< $T_0$  start>  
< $T_0$ , A, 1000, 950>  
< $T_0$ , B, 2000, 2050>

## Write

$A = 950$   
 $B = 2050$

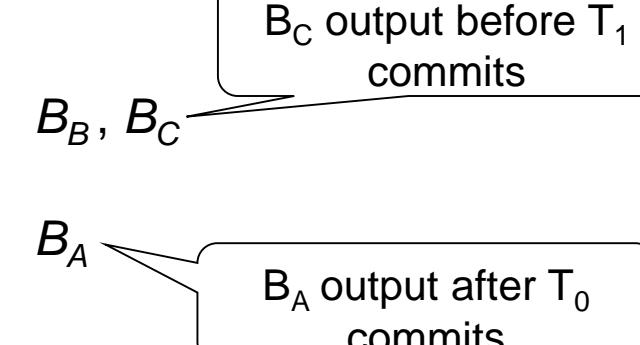
< $T_0$  commit>  
< $T_1$  start>  
< $T_1$ , C, 700, 600>

$C = 600$

< $T_1$  commit>

- Note:  $B_X$  denotes block containing X

## Output





# Undo and Redo Operations

- **Undo** of a log record  $\langle T_i, X, V_1, V_2 \rangle$  writes the **old** value  $V_1$  to  $X$
- **Redo** of a log record  $\langle T_i, X, V_1, V_2 \rangle$  writes the **new** value  $V_2$  to  $X$
- **Undo and Redo of Transactions**
  - **undo**( $T_i$ ) restores the value of all data items updated by  $T_i$  to their old values, going backwards from the last log record for  $T_i$ 
    - ▶ Each time a data item  $X$  is restored to its old value  $V$  a special log record (called **redo-only**)  $\langle T_i, X, V \rangle$  is written out
    - ▶ When undo of a transaction is complete, a log record  $\langle T_i \text{ abort} \rangle$  is written out (to indicate that the undo was completed)
  - **redo**( $T_i$ ) sets the value of all data items updated by  $T_i$  to the new values, going forward from the first log record for  $T_i$ 
    - ▶ No logging is done in this case



# Undo and Redo Operations (Cont.)

- The **undo** and **redo** operations are used in several different circumstances:
  - The **undo** is used for transaction rollback during normal operation
    - ▶ in case a transaction cannot complete its execution due to some logical error
  - The **undo** and **redo** operations are used during recovery from failure
- We need to deal with the case where during recovery from failure another failure occurs prior to the system having fully recovered



## Transaction rollback (during normal operation)

- Let  $T_i$  be the transaction to be rolled back
- Scan log backwards from the end, and for each log record of  $T_i$  of the form  $\langle T_i, X_j, V_1, V_2 \rangle$ 
  - Perform the undo by writing  $V_1$  to  $X_j$ ,
  - Write a log record  $\langle T_i, X_j, V_1 \rangle$ 
    - ▶ such log records are called **compensation log records**
- Once the record  $\langle T_i \text{ start} \rangle$  is found stop the scan and write the log record  $\langle T_i \text{ abort} \rangle$



# Undo and Redo on Recovering from Failure

- When recovering after failure:

- Transaction  $T_i$  needs to be undone if the log
  - ▶ contains the record  $\langle T_i \text{ start} \rangle$ ,
  - ▶ but does not contain either the record  $\langle T_i \text{ commit} \rangle$  or  $\langle T_i \text{ abort} \rangle$
- Transaction  $T_i$  needs to be redone if the log
  - ▶ contains the records  $\langle T_i \text{ start} \rangle$
  - ▶ and contains the record  $\langle T_i \text{ commit} \rangle$  or  $\langle T_i \text{ abort} \rangle$
- It may seem strange to redo transaction  $T_i$  if the record  $\langle T_i \text{ abort} \rangle$  record is in the log
  - ▶ To see why this works, note that if  $\langle T_i \text{ abort} \rangle$  is in the log, so are the redo-only records written by the undo operation. Thus, the end result will be to undo  $T_i$ 's modifications in this case. This slight redundancy simplifies the recovery algorithm and enables faster overall recovery time
  - ▶ such a redo redoes all the original actions including the steps that restored old value – Known as **repeating history**



# Immediate Modification Recovery Example

Below we show the log as it appears at three instances of time.

$\langle T_0 \text{ start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_0 \text{ start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_0 \text{ commit} \rangle$

$\langle T_1 \text{ start} \rangle$

$\langle T_1, C, 700, 600 \rangle$

$\langle T_0 \text{ start} \rangle$

$\langle T_0, A, 1000, 950 \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_0 \text{ commit} \rangle$

$\langle T_1 \text{ start} \rangle$

$\langle T_1, C, 700, 600 \rangle$

$\langle T_1 \text{ commit} \rangle$

(a)

(b)

(c)

Recovery actions in each case above are:

- undo ( $T_0$ ): B is restored to 2000 and A to 1000, and log records  $\langle T_0, B, 2000 \rangle$ ,  $\langle T_0, A, 1000 \rangle$ ,  $\langle T_0, \text{abort} \rangle$  are written out
- redo ( $T_0$ ) and undo ( $T_1$ ): A and B are set to 950 and 2050 and C is restored to 700. Log records  $\langle T_1, C, 700 \rangle$ ,  $\langle T_1, \text{abort} \rangle$  are written out
- redo ( $T_0$ ) and redo ( $T_1$ ): A and B are set to 950 and 2050 respectively. Then C is set to 600



# Checkpoints

- Redoing/undoing all transactions recorded in the log can be very slow
  - Processing the entire log is time-consuming if the system has run for a long time
  - We might unnecessarily redo transactions which have already output their updates to the database
- Streamline recovery procedure by periodically performing **checkpointing**
- All updates are stopped while doing checkpointing
  1. Output all log records currently residing in main memory onto stable storage
  2. Output all modified buffer blocks to the disk
  3. Write a log record < **checkpoint**  $L$  > onto stable storage where  $L$  is a list of all transactions active at the time of checkpoint

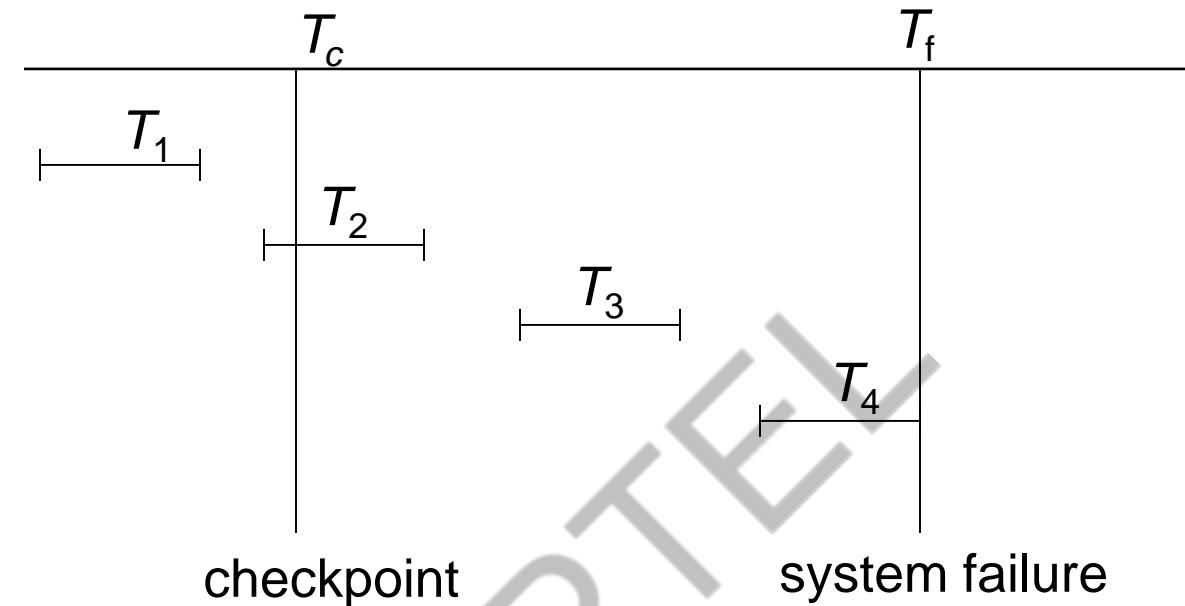


# Checkpoints (Cont.)

- During recovery we need to consider only the most recent transaction  $T_i$  that started before the checkpoint, and transactions that started after  $T_i$ 
  - Scan backwards from end of log to find the most recent <checkpoint  $L$ > record
  - Only transactions that are in  $L$  or started after the checkpoint need to be redone or undone
  - Transactions that committed or aborted before the checkpoint already have all their updates output to stable storage
- Some earlier part of the log may be needed for undo operations
  - Continue scanning backwards till a record < $T_i$ , start> is found for every transaction  $T_i$  in  $L$
  - Parts of log prior to earliest < $T_i$ , start> record above are not needed for recovery, and can be erased whenever desired



# Example of Checkpoints



- Any transactions that committed before the last checkpoint should be ignored
  - $T_1$  can be ignored (updates already output to disk due to checkpoint)
- Any transactions that committed since the last checkpoint need to be redone
  - $T_2$  and  $T_3$  redone
- Any transaction that was running at the time of failure needs to be undone and restarted
  - $T_4$  undone



# Module Summary

- Failures may be due to variety of sources – each needs a strategy for handling
- A proper mix and management of volatile, non-volatile and stable storage can guarantee recovery from failures and ensure Atomicity, Consistency and Durability
- Log-based recovery is efficient and effective

NPTEL



# Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

**Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors.**

**Edited and new slides are marked with “PPD”.**



# Database Management Systems

## Module 37: Recovery/2

**Partha Pratim Das**

*Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur*

[ppd@cse.iitkgp.ernet.in](mailto:ppd@cse.iitkgp.ernet.in)

**Srijoni Majumdar  
Himadri B G S Bhuyan  
Gurunath Reddy M**



**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
[www.db-book.com](http://www.db-book.com)



# Module Recap

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

NPTEL



# Module Objectives

- To focus on concurrent transactions and understand the recovery algorithms
- To understand operation logging for recovery with early lock release

NPTEL



# Module Outline

- Recovery Algorithm
- Recovery with Early Lock Release

NPTEL



- **Recovery Algorithm**
- Recovery with Early Lock Release

# RECOVERY ALGORITHM

NPTEL



# Recovery Schemes

- **So far:**
  - We covered key concepts
  - We assumed serial execution of transactions
- **Now:**
  - We discuss concurrency control issues
  - We present the components of the basic recovery algorithm

NPTEL

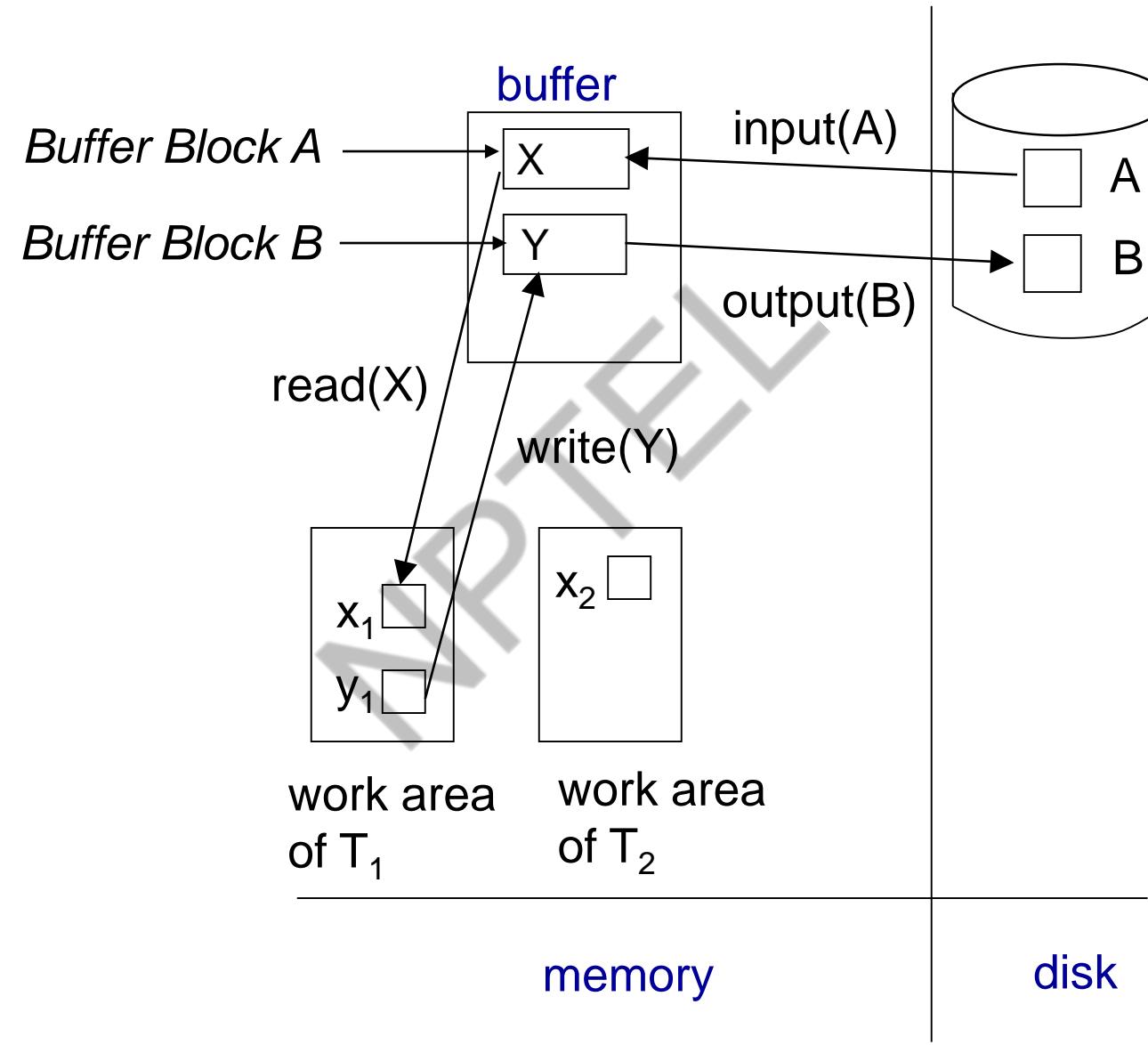


# Concurrency Control and Recovery

- With concurrent transactions, all transactions share a single disk buffer and a single log
  - A buffer block can have data items updated by one or more transactions
- We assume that *if a transaction  $T_i$  has modified an item, no other transaction can modify the same item until  $T_i$  has committed or aborted*
  - That is, the updates of uncommitted transactions should not be visible to other transactions
    - ▶ Otherwise how do we perform undo if  $T_1$  updates A, then  $T_2$  updates A and commits, and finally  $T_1$  has to abort?
  - Can be ensured by obtaining exclusive locks on updated items and holding the locks till end of transaction (strict two-phase locking)
- Log records of different transactions may be interspersed in the log



## Example of Data Access with Concurrent transactions





# Recovery Algorithm

- **Logging** (during normal operation):

- $\langle T_i \text{ start} \rangle$  at transaction start
- $\langle T_i, X_j, V_1, V_2 \rangle$  for each update, and
- $\langle T_i \text{ commit} \rangle$  at transaction end

NPTEL



# Recovery Algorithm (Contd.)

## ■ Transaction rollback (during normal operation)

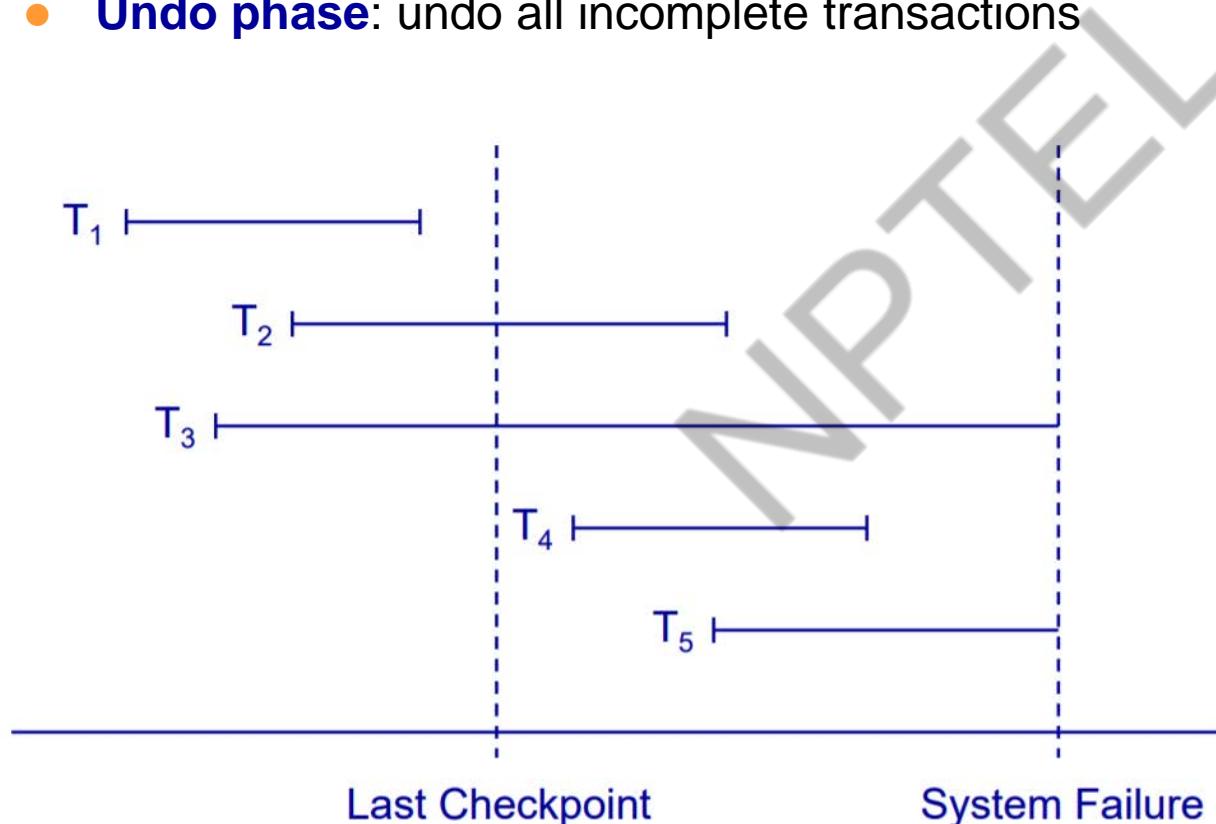
- Let  $T_i$  be the transaction to be rolled back
- Scan log backwards from the end, and for each log record of  $T_i$  of the form  $\langle T_i, X_j, V_1, V_2 \rangle$ 
  - ▶ perform the undo by writing  $V_1$  to  $X_j$ ,
  - ▶ write a log record  $\langle T_i, X_j, V_1 \rangle$ 
    - such log records are called **compensation log records**
- Once the record  $\langle T_i \text{ start} \rangle$  is found stop the scan and write the log record  $\langle T_i \text{ abort} \rangle$



# Recovery Algorithm (Cont.)

## ■ Recovery from failure: Two phases

- **Redo phase:** replay updates of **all** transactions, whether they committed, aborted, or are incomplete
- **Undo phase:** undo all incomplete transactions



### Requirement:

- Transactions of type T1 need no recovery
- Transactions of type T2 or T4 need to be redone
- Transactions of type T3 or T5 need to be undone and restarted

### Strategy:

- Ignore T1
- Redo T2, T3, T4 and T5
- Undo T3 and T5



# Recovery Algorithm (Cont.)

## ■ Redo phase:

1. Find last <checkpoint  $L$ > record, and set undo-list to  $L$
2. Scan forward from above <checkpoint  $L$ > record
  1. Whenever a record < $T_i, X_j, V_1, V_2$ > is found, redo it by writing  $V_2$  to  $X_j$
  2. Whenever a log record < $T_i$  start> is found, add  $T_i$  to undo-list
  3. Whenever a log record < $T_i$  commit> or < $T_i$  abort> is found, remove  $T_i$  from undo-list



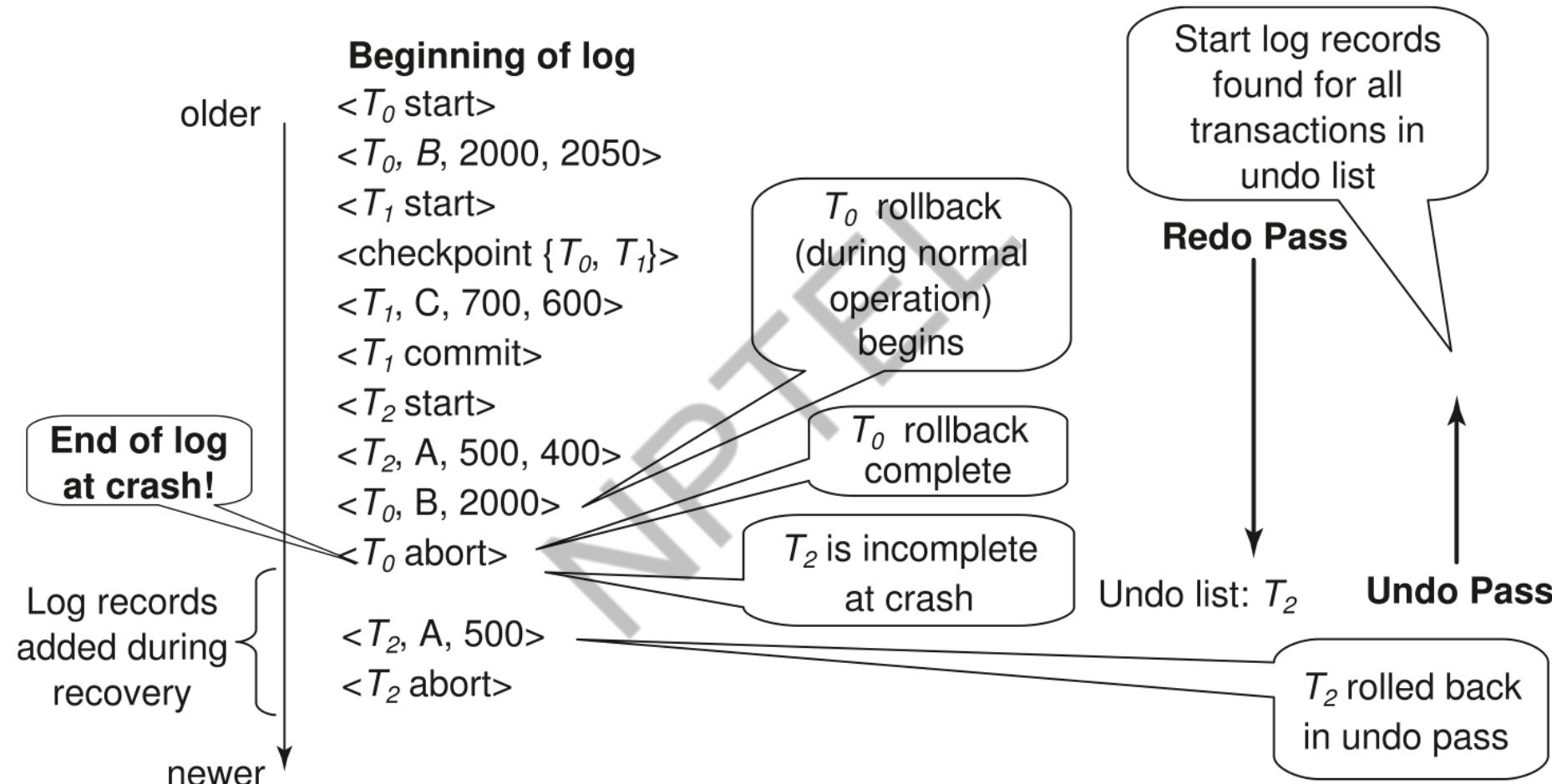
# Recovery Algorithm (Cont.)

## ■ Undo phase:

1. Scan log backwards from end
  1. Whenever a log record  $\langle T_i, X_j, V_1, V_2 \rangle$  is found where  $T_i$  is in undo-list perform same actions as for transaction rollback:
    1. Perform undo by writing  $V_1$  to  $X_j$
    2. Write a log record  $\langle T_i, X_j, V_1 \rangle$
  2. Whenever a log record  $\langle T_i \text{ start} \rangle$  is found where  $T_i$  is in undo-list,
    1. Write a log record  $\langle T_i \text{ abort} \rangle$
    2. Remove  $T_i$  from undo-list
  3. Stop when undo-list is empty
    - That is,  $\langle T_i \text{ start} \rangle$  has been found for every transaction in undo-list
- After undo phase completes, normal transaction processing can commence



# Example of Recovery





- Recovery Algorithm
- Recovery with Early Lock Release

NPTEL

# RECOVERY WITH EARLY LOCK RELEASE



# Recovery with Early Lock Release

- Support for high-concurrency locking techniques, such as those used for B<sup>+</sup>-tree concurrency control, which release locks early
  - Supports “logical undo”
- Recovery based on “repeating history”, whereby recovery executes exactly the same actions as normal processing

NPTEL



# Logical Undo Logging

- Operations like B<sup>+</sup>-tree insertions and deletions release locks early
  - They cannot be undone by restoring old values (**physical undo**), since once a lock is released, other transactions may have updated the B<sup>+</sup>-tree
  - Instead, insertions (resp. deletions) are undone by executing a deletion (resp. insertion) operation (known as **logical undo**)
- For such operations, undo log records should contain the undo operation to be executed
  - Such logging is called **logical undo logging**, in contrast to **physical undo logging**
    - ▶ Operations are called **logical operations**
  - Other examples:
    - ▶ delete of tuple, to undo insert of tuple
      - allows early lock release on space allocation information
    - ▶ subtract amount deposited, to undo deposit
      - allows early lock release on bank balance



# Physical Redo

- Redo information is logged **physically** (that is, new value for each write) even for operations with logical undo
  - Logical redo is very complicated since database state on disk may not be “operation consistent” when recovery starts
  - Physical redo logging does not conflict with early lock release

NPTEL



# Operation Logging

- Operation logging is done as follows:

1. When operation starts, log  $\langle T_i, O_j, \text{operation-begin} \rangle$ . Here  $O_j$  is a unique identifier of the operation instance
2. While operation is executing, normal log records with physical redo and physical undo information are logged
3. When operation completes,  $\langle T_i, O_j, \text{operation-end}, U \rangle$  is logged, where  $U$  contains information needed to perform a logical undo information

Example: insert of (key, record-id) pair (K5, RID7) into index I9

$\langle T1, O1, \text{operation-begin} \rangle$   
....  
 $\langle T1, X, 10, K5 \rangle$   
 $\langle T1, Y, 45, \text{RID7} \rangle$   
 $\langle T1, O1, \text{operation-end}, (\text{delete I9, K5, RID7}) \rangle$

$\left. \begin{matrix} \langle T1, X, 10, K5 \rangle \\ \langle T1, Y, 45, \text{RID7} \rangle \end{matrix} \right\}$  Physical redo of steps in insert



# Operation Logging (Cont.)

- If crash/rollback occurs before operation completes:
  - the **operation-end** log record is not found, and
  - the physical undo information is used to undo operation
- If crash/rollback occurs after the operation completes:
  - the **operation-end** log record is found, and in this case
  - logical undo is performed using  $U$ ; the physical undo information for the operation is ignored
- Redo of operation (after crash) still uses physical redo information



# Transaction Rollback with Logical Undo

Rollback of transaction  $T_i$ , scan the log backwards

1. If a log record  $\langle T_i, X, V_1, V_2 \rangle$  is found, perform the undo and log  $\langle T_i, X, V_1 \rangle$
2. If a  $\langle T_i, O_j, \text{operation-end}, U \rangle$  record is found
  - Rollback the operation logically using the undo information  $U$ 
    - Updates performed during roll back are logged just like during normal operation execution
    - At the end of the operation rollback, instead of logging an **operation-end** record, generate a record  $\langle T_i, O_j, \text{operation-abort} \rangle$
  - Skip all preceding log records for  $T_i$  until the record  $\langle T_i, O_j, \text{operation-begin} \rangle$  is found
3. If a redo-only record is found ignore it
4. If a  $\langle T_i, O_j, \text{operation-abort} \rangle$  record is found:
  - skip all preceding log records for  $T_i$  until the record  $\langle T_i, O_j, \text{operation-begin} \rangle$  is found
5. Stop the scan when the record  $\langle T_i, \text{start} \rangle$  is found
6. Add a  $\langle T_i, \text{abort} \rangle$  record to the log

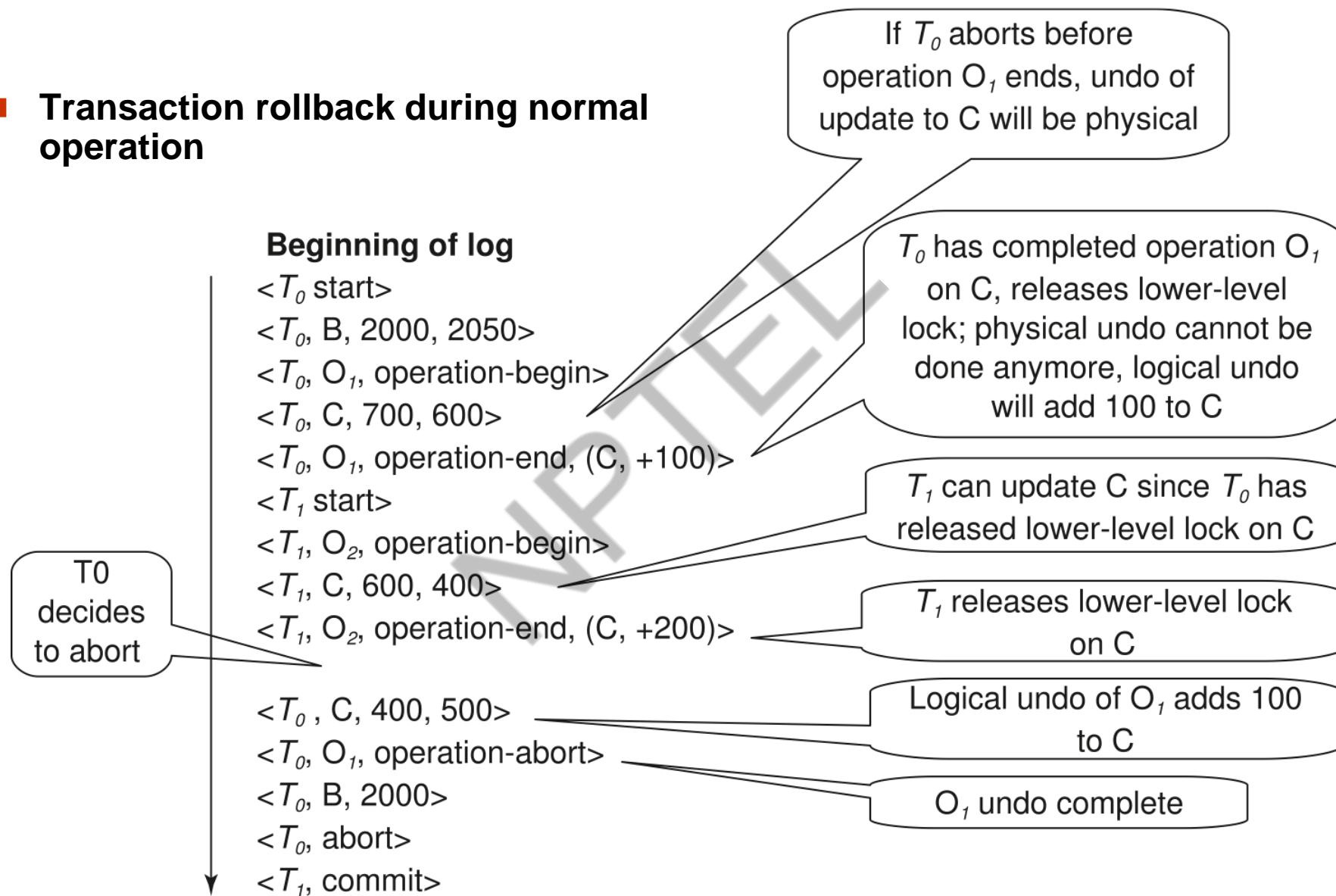
Note:

- Cases 3 and 4 above can occur only if the database crashes while a transaction is being rolled back
- Skipping of log records as in case 4 is important to prevent multiple rollback of the same operation



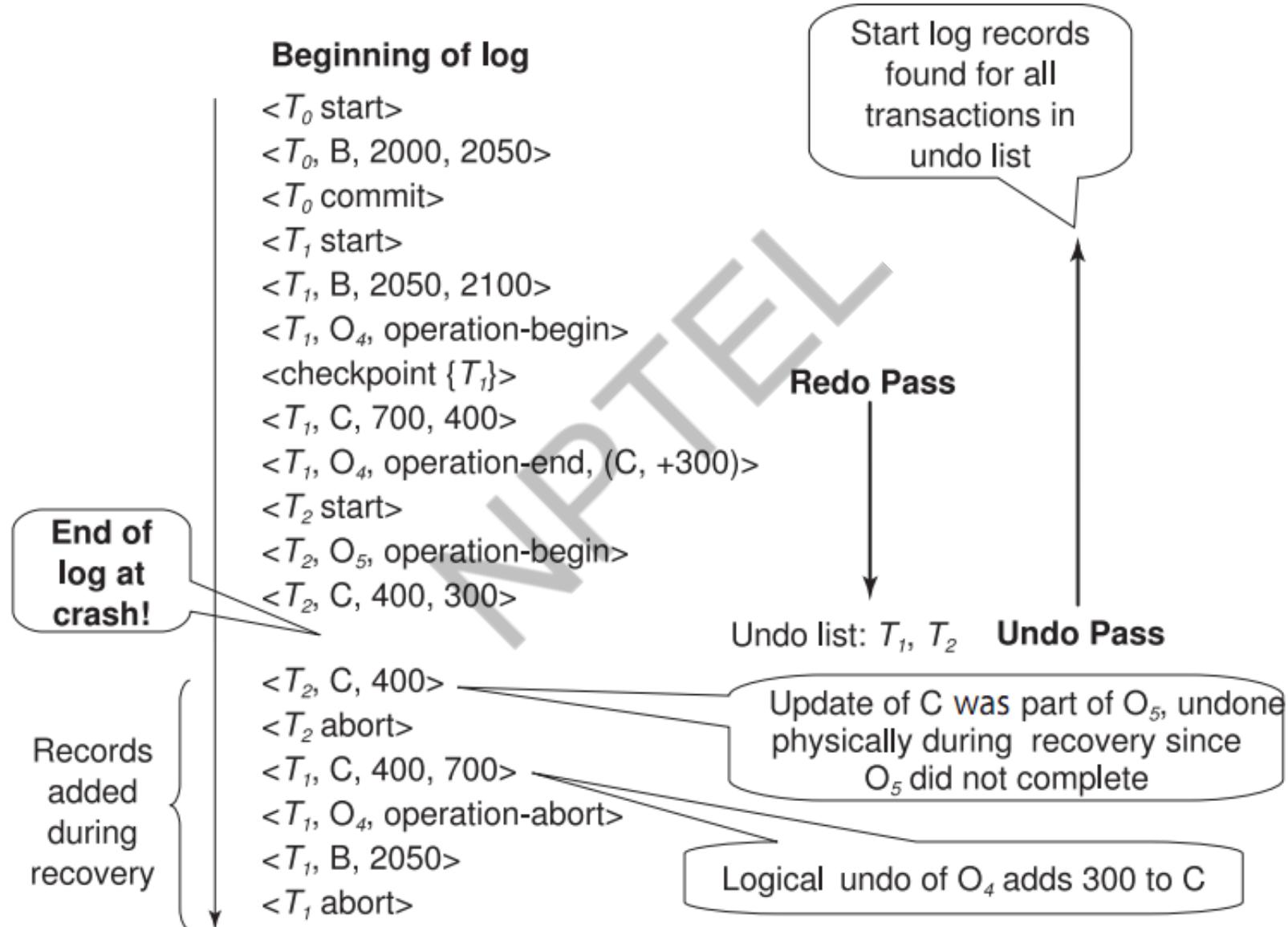
# Transaction Rollback with Logical Undo

## ■ Transaction rollback during normal operation





# Failure Recovery with Logical Undo





# Transaction Rollback: Another Example

- Example with a complete and an incomplete operation

<T1, start>

<T1, O1, operation-begin>

....

<T1, X, 10, K5>

<T1, Y, 45, RID7>

<T1, O1, operation-end, (delete I9, K5, RID7)>

<T1, O2, operation-begin>

<T1, Z, 45, 70>

    ← T1 Rollback begins here

<T1, Z, 45>   ← redo-only log record during physical undo (of incomplete O2)

<T1, Y, ... , ...>   ← Normal redo records for logical undo of O1

...

<T1, O1, operation-abort>   ← What if crash occurred immediately after this?

<T1, abort>



# Recovery Algorithm with Logical Undo

Basically same as earlier algorithm, except for changes described earlier for transaction rollback

1. (Redo phase): Scan log forward from last <checkpoint  $L$ > record till end of log
  1. Repeat history by physically redoing all updates of all transactions,
  2. Create an undo-list during the scan as follows
    - ▶  $undo-list$  is set to  $L$  initially
    - ▶ Whenever  $\langle T_i, \text{start} \rangle$  is found  $T_i$  is added to  $undo-list$
    - ▶ Whenever  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$  is found,  $T_i$  is deleted from  $undo-list$

This brings database to state as of crash, with committed as well as uncommitted transactions having been redone

Now  $undo-list$  contains transactions that are incomplete, that is, have neither committed nor been fully rolled back



# Recovery with Logical Undo (Cont.)

Recovery from system crash (cont.)

2. **(Undo phase):** Scan log backwards, performing undo on log records of transactions found in *undo-list*
  - Log records of transactions being rolled back are processed as described earlier, as they are found
    - ▶ Single shared scan for all transactions being undone
  - When  $\langle T_i \text{ start} \rangle$  is found for a transaction  $T_i$  in *undo-list*, write a  $\langle T_i \text{ abort} \rangle$  log record.
  - Stop scan when  $\langle T_i \text{ start} \rangle$  records have been found for all  $T_i$  in *undo-list*
- This undoes the effects of incomplete transactions (those with neither **commit** nor **abort** log records). Recovery is now complete



# Module Summary

- Studies the recovery algorithms for concurrent transactions
- Recovery based on operation logging supplements log-based recovery

NPTEL



# Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

**Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors.**

**Edited and new slides are marked with “PPD”.**



# Database Management Systems

## Module 38: Query Processing and Optimization/1: Processing

**Partha Pratim Das**

*Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur*

[ppd@cse.iitkgp.ernet.in](mailto:ppd@cse.iitkgp.ernet.in)

**Srijoni Majumdar  
Himadri B G S Bhuyan  
Gurunath Reddy M**



**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
[www.db-book.com](http://www.db-book.com)



# Module Recap

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

NPTEL



# Module Objectives

- To understand the overall flow for Query Processing
- To define the Measures of Query Cost
- To understand the algorithms for processing Selection Operations, Sorting, Join Operations, and a few Other Operations

NPTEL



# Module Outline

- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations

NPTEL



# OVERVIEW OF QUERY PROCESSING

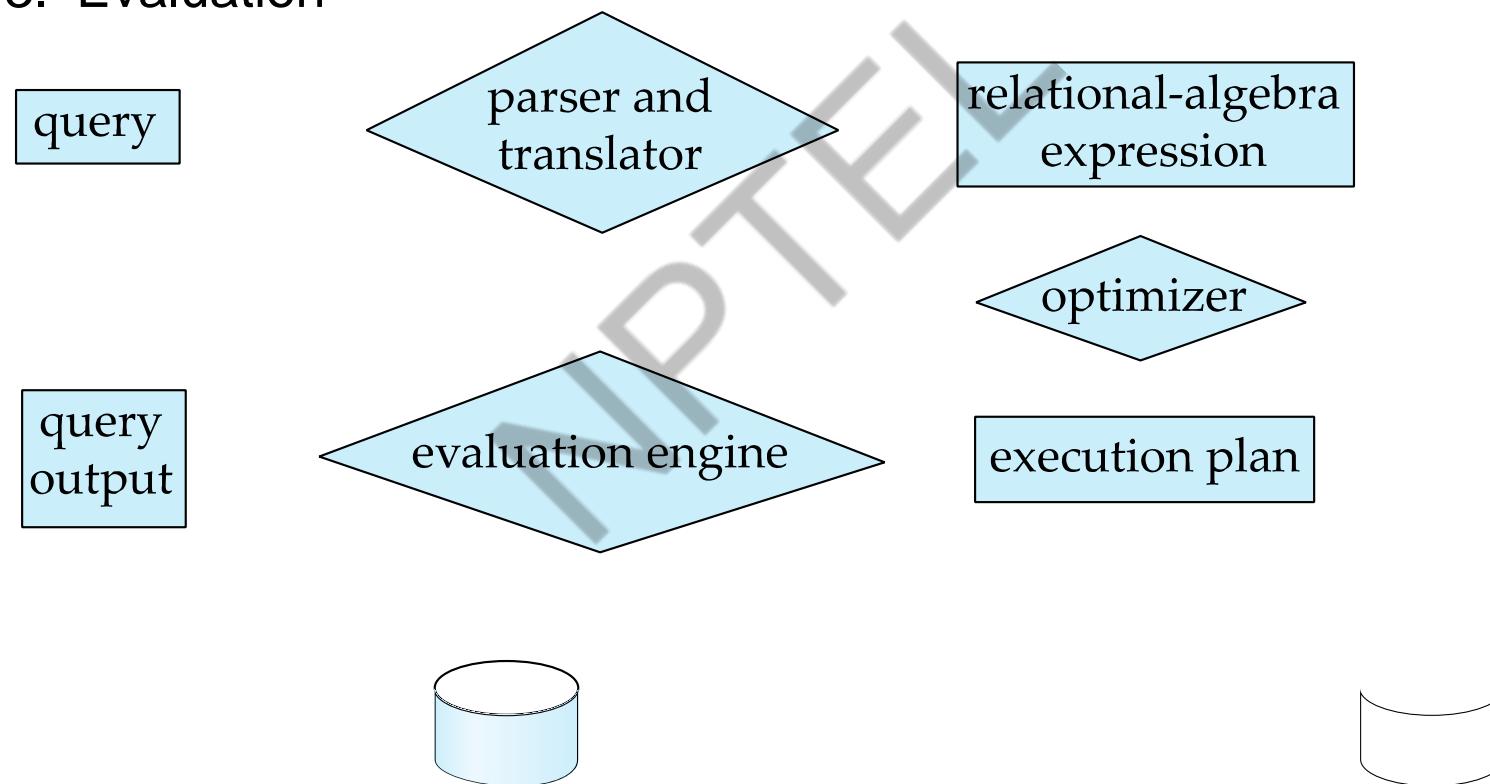
- **Overview of Query Processing**
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations

NPTEL



# Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation





# Basic Steps in Query Processing (Cont.)

- Parsing and translation
  - translate the query into its internal form
    - ▶ This is then translated into relational algebra
  - Parser checks syntax, verifies relations
- Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query



# Basic Steps in Query Processing : Optimization

- A relational algebra expression may have many equivalent expressions
  - E.g.,  $\sigma_{\text{salary} < 75000}(\Pi_{\text{salary}}(\text{instructor}))$  is equivalent to  
 $\Pi_{\text{salary}}(\sigma_{\text{salary} < 75000}(\text{instructor}))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.
  - E.g., can use an index on *salary* to find instructors with  $\text{salary} < 75000$ ,
  - or can perform complete relation scan and discard instructors with  $\text{salary} \geq 75000$



# Basic Steps: Optimization (Cont.)

- **Query Optimization:** Amongst all equivalent evaluation plans choose the one with lowest cost
  - Cost is estimated using statistical information from the database catalog
    - ▶ e.g. number of tuples in each relation, size of tuples, etc.
- In this module we study
  - How to measure query costs
  - Algorithms for evaluating relational algebra operations
  - How to combine algorithms for individual operations in order to evaluate a complete expression
- In the next module
  - We study how to optimize queries, that is, how to find an evaluation plan with lowest estimated cost



- Overview of Query Processing
- **Measures of Query Cost**
- Selection Operation
- Sorting
- Join Operation
- Other Operations

NPTEL

# MEASURES OF QUERY COST



# Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - ▶ *disk accesses, CPU, or even network communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate
- Measured by taking into account
  - Number of seeks                            \* average-seek-cost
  - Number of blocks read                    \* average-block-read-cost
  - Number of blocks written                \* average-block-write-cost
    - ▶ Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful



# Measures of Query Cost (Cont.)

- For simplicity we just use the **number of block transfers** from disk and the **number of seeks** as the cost measures
  - $t_T$  – time to transfer one block
  - $t_S$  – time for one seek
  - Cost for  $b$  block transfers plus  $S$  seeks
$$b * t_T + S * t_S$$
- We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae



# Measures of Query Cost (Cont.)

- Several algorithms can reduce disk IO by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
    - ▶ We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation



# SELECTION OPERATION

- Overview of Query Processing
- Measures of Query Cost
- **Selection Operation**
- Sorting
- Join Operation
- Other Operations

NPTEL



# Selection Operation

- **File scan**
- Algorithm **A1** (**linear search**). Scan each file block and test all records to see whether they satisfy the selection condition
  - Cost estimate =  $b_r$  block transfers + 1 seek
    - ▶  $b_r$  denotes number of blocks containing records from relation  $r$
  - If selection is on a key attribute, can stop on finding record
    - ▶ cost =  $(b_r/2)$  block transfers + 1 seek
  - Linear search can be applied regardless of
    - ▶ selection condition or
    - ▶ ordering of records in the file, or
    - ▶ availability of indices
- Note: binary search generally does not make sense since data is not stored consecutively
  - except when there is an index available,
  - and binary search requires more seeks than index search



# Selections Using Indices

- **Index scan** – search algorithms that use an index
  - selection condition must be on search-key of index
  - $h_i$  = height of B+ Tree
- **A2 (primary index, equality on key)**. Retrieve a single record that satisfies the corresponding equality condition
  - $\text{Cost} = (h_i + 1) * (t_T + t_S)$
- **A3 (primary index, equality on nonkey)** Retrieve multiple records
  - Records will be on consecutive blocks
    - ▶ Let  $b$  = number of blocks containing matching records
  - $\text{Cost} = h_i * (t_T + t_S) + t_S + t_T * b$



# Selections Using Indices

## ■ A4 (secondary index, equality on nonkey).

- Retrieve a single record if the search-key is a candidate key
  - ▶  $\text{Cost} = (h_i + 1) * (t_T + t_S)$
- Retrieve multiple records if search-key is not a candidate key
  - ▶ each of  $n$  matching records may be on a different block
  - ▶  $\text{Cost} = (h_i + n) * (t_T + t_S)$ 
    - Can be very expensive!



# Selections Involving Comparisons

- Can implement selections of the form  $\sigma_{A \leq v}(r)$  or  $\sigma_{A \geq v}(r)$  by using
  - a linear file scan,
  - or by using indices in the following ways:
- **A5 (primary index, comparison).** (Relation is sorted on A)
  - ▶ For  $\sigma_{A \geq v}(r)$  use index to find first tuple  $\geq v$  and scan relation sequentially from there
    - Cost =  $h_i * (t_T + t_S) + b * t_T$
  - ▶ For  $\sigma_{A \leq v}(r)$  just scan relation sequentially till first tuple  $> v$ ; do not use index
    - Cost =  $t_S + b * t_T$



# Selections Involving Comparisons

- Can implement selections of the form  $\sigma_{A \leq v}(r)$  or  $\sigma_{A \geq v}(r)$  by using
  - a linear file scan,
  - or by using indices in the following ways:
- **A6 (secondary index, comparison).**
  - ▶ For  $\sigma_{A \geq v}(r)$  use index to find first index entry  $\geq v$  and scan index sequentially from there, to find pointers to records
    - Cost =  $(h_i + n) * (t_T + t_S)$
  - ▶ For  $\sigma_{A \leq v}(r)$  just scan leaf pages of index finding pointers to records, till first entry  $> v$
  - ▶ In either case, retrieve records that are pointed to
    - requires an I/O for each record
    - Linear file scan may be cheaper



# Implementation of Complex Selections

- **Conjunction:**  $\sigma_{\theta_1} \wedge \theta_2 \wedge \dots \wedge \theta_n(r)$
- **A7 (conjunctive selection using one index)**
  - Select a combination of  $\theta_i$ , and algorithms A1 through A6 that results in the least cost for  $\sigma_{\theta_i}(r)$
  - Test other conditions on tuple after fetching it into memory buffer
- **A8 (conjunctive selection using composite index)**
  - Use appropriate composite (multiple-key) index if available
- **A9 (conjunctive selection by intersection of identifiers)**
  - Requires indices with record pointers
  - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory



# Algorithms for Complex Selections

- **Disjunction:**  $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$ .
- **A10 (disjunctive selection by union of identifiers)**
  - Applicable if *all* conditions have available indices
    - ▶ Otherwise use linear scan
  - Use corresponding index for each condition, and take union of all the obtained sets of record pointers
  - Then fetch records from file
- **Negation:**  $\sigma_{\neg\theta}(r)$ 
  - Use linear scan on file
  - If very few records satisfy  $\neg\theta$ , and an index is applicable to  $\theta$ 
    - ▶ Find satisfying records using index and fetch from file



# SORTING

- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- **Sorting**
- Join Operation
- Other Operations

NPTEL



# Sorting

- We may build an index on the relation, and then use the index to read the relation in sorted order
  - May lead to one disk block access for each tuple
- For relations that fit in memory, techniques like quicksort can be used
- For relations that do not fit in memory, **external sort-merge** is a good choice

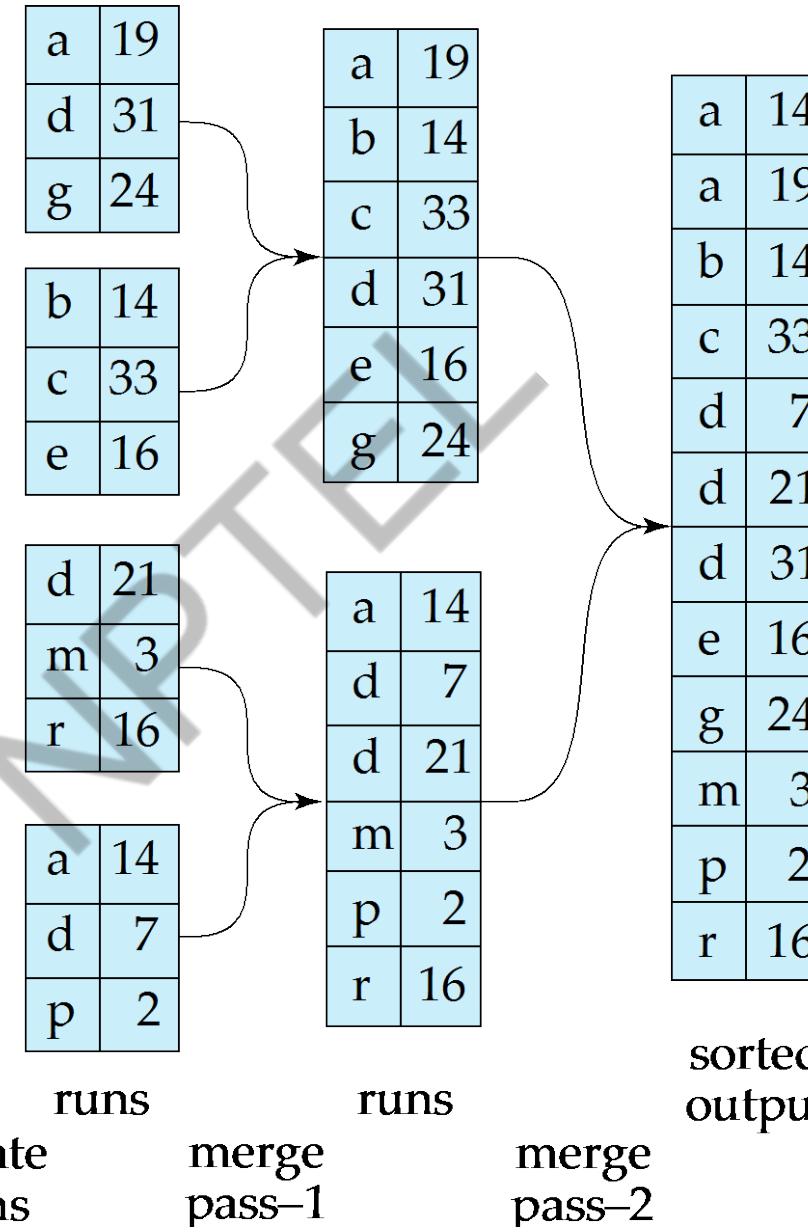
NPTEL



# Example: External Sorting Using Sort-Merge

g	24
a	19
d	31
c	33
b	14
e	16
r	16
d	21
m	3
p	2
d	7
a	14

initial  
relation





# External Sort-Merge

Let  $M$  denote memory size (in pages)

1. **Create sorted runs.** Let  $i$  be 0 initially

Repeatedly do the following till the end of the relation:

- (a) Read  $M$  blocks of relation into memory
- (b) Sort the in-memory blocks
- (c) Write sorted data to run  $R_i$ ; increment  $i$ .

Let the final value of  $i$  be  $N$

2. *Merge the runs (next slide).....*



# External Sort-Merge (Cont.)

## 2. Merge the runs (N-way merge). We assume (for now) that $N < M$

1. Use  $N$  blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page
2. **repeat**
  1. Select the first record (in sort order) among all buffer pages
  2. Write the record to the output buffer. If the output buffer is full write it to disk.
  3. Delete the record from its input buffer page  
**If** the buffer page becomes empty **then**  
read the next block (if any) of the run into the buffer
3. **until** all input buffer pages are empty:



# External Sort-Merge (Cont.)

- If  $N \geq M$ , several merge passes are required
  - In each pass, contiguous groups of  $M - 1$  runs are merged.
  - A pass reduces the number of runs by a factor of  $M - 1$ , and creates runs longer by the same factor
    - ▶ E.g. If  $M=11$ , and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
  - Repeated passes are performed till all runs have been merged into one



- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- **Join Operation**
- Other Operations

# JOIN OPERATION



# Join Operation

- Several different algorithms to implement joins
  - **Nested-loop join**
  - **Block nested-loop join**
  - **Indexed nested-loop join**
  - Merge-join
  - Hash-join
- Choice based on cost estimate
- Examples use the following information
  - Number of records of *student*: 5,000      *takes*: 10,000
  - Number of blocks of *student*: 100      *takes*: 400



# Nested-Loop Join

- To compute the theta join  $r \bowtie_{\theta} s$   
**for each tuple  $t_r$  in  $r$  do begin**  
    **for each tuple  $t_s$  in  $s$  do begin**  
        test pair  $(t_r, t_s)$  to see if they satisfy the join condition  $\theta$   
        if they do, add  $t_r \bullet t_s$  to the result.  
    **end**  
**end**
- $r$  is called the **outer relation** and  $s$  the **inner relation** of the join
- Requires no indices and can be used with any kind of join condition
- Expensive since it examines every pair of tuples in the two relations



# Nested-Loop Join (Cont.)

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is
  - $n_r * b_s + b_r$  block transfers, plus
  - $n_r + b_r$  seeks
- If the smaller relation fits entirely in memory, use that as the inner relation.
  - Reduces cost to  $b_r + b_s$  block transfers and 2 seeks
- Example of join of *students* and *takes*:
  - Number of records of *student*: 5,000    *takes*: 10,000
  - Number of blocks of *student*: 100    *takes*: 400
- Assuming worst case memory availability cost estimate is
  - with *student* as outer relation:
    - ▶  $5000 * 400 + 100 = 2,000,100$  block transfers,
    - ▶  $5000 + 100 = 5100$  seeks
  - with *takes* as the outer relation
    - ▶  $10000 * 100 + 400 = 1,000,400$  block transfers and 10,400 seeks
- If smaller relation (*student*) fits entirely in memory, the cost estimate will be 500 block transfers
- Block nested-loops algorithm (next slide) is preferable



# Block Nested-Loop Join

- Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation

```
for each block  $B_r$  of  $r$  do begin  
    for each block  $B_s$  of  $s$  do begin  
        for each tuple  $t_r$  in  $B_r$  do begin  
            for each tuple  $t_s$  in  $B_s$  do begin  
                Check if  $(t_r, t_s)$  satisfy the join condition  
                if they do, add  $t_r \bullet t_s$  to the result.  
            end  
        end  
    end  
end
```



# Block Nested-Loop Join (Cont.)

- Worst case estimate:  $b_r * b_s + b_r$  block transfers +  $2 * b_r$  seeks
  - Each block in the inner relation  $s$  is read once for each *block* in the outer relation
- Best case:  $b_r + b_s$  block transfers + 2 seeks.
- Improvements to nested loop and block nested loop algorithms:
  - In block nested-loop, use  $M - 2$  disk blocks as blocking unit for outer relations, where  $M$  = memory size in blocks; use remaining two blocks to buffer inner relation and output
    - ▶ Cost =  $\lceil b_r / (M-2) \rceil * b_s + b_r$  block transfers +  $2 \lceil b_r / (M-2) \rceil$  seeks
  - If equi-join attribute forms a key or inner relation, stop inner loop on first match
  - Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)
  - Use index on inner relation if available (next slide)



# Indexed Nested-Loop Join

- Index lookups can replace file scans if
  - join is an equi-join or natural join and
  - an index is available on the inner relation's join attribute
    - ▶ Can construct an index just to compute a join.
- For each tuple  $t_r$  in the outer relation  $r$ , use the index to look up tuples in  $s$  that satisfy the join condition with tuple  $t_r$ .
- Worst case: buffer has space for only one page of  $r$ , and, for each tuple in  $r$ , we perform an index lookup on  $s$ .
- Cost of the join:  $b_r(t_T + t_S) + n_r * c$ 
  - Where  $c$  is the cost of traversing index and fetching all matching  $s$  tuples for one tuple of  $r$
  - $c$  can be estimated as cost of a single selection on  $s$  using the join condition.
- If indices are available on join attributes of both  $r$  and  $s$ , use the relation with fewer tuples as the outer relation.



# Example of Nested-Loop Join Costs

- Compute  $student \bowtie takes$ , with  $student$  as the outer relation.
- Let  $takes$  have a primary B<sup>+</sup>-tree index on the attribute  $ID$ , which contains 20 entries in each index node.
- Since  $takes$  has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data
- $student$  has 5000 tuples
- Cost of block nested loops join
  - $400 * 100 + 100 = 40,100$  block transfers +  $2 * 100 = 200$  seeks
    - ▶ assuming worst case memory
    - ▶ may be significantly less with more memory
- Cost of indexed nested loops join
  - $100 + 5000 * 5 = 25,100$  block transfers and seeks.
  - CPU cost likely to be less than that for block nested loops join



- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- **Other Operations**

NPTEL

# OTHER OPERATIONS



# Other Operations

- **Duplicate elimination**
- **Projection**
- **Aggregation**
- Set Operations
- Outer Join

NPTEL



# Other Operations

- **Duplicate elimination** can be implemented via hashing or sorting
  - On sorting duplicates will come adjacent to each other, and all but one set of duplicates can be deleted
  - *Optimization:* duplicates can be deleted during run generation as well as at intermediate merge steps in external sort-merge
  - Hashing is similar – duplicates will come into the same bucket
- **Projection:**
  - perform projection on each tuple
  - followed by duplicate elimination



# Other Operations : Aggregation

- **Aggregation** can be implemented in a manner similar to duplicate elimination
  - Sorting or hashing can be used to bring tuples in the same group together, and then the aggregate functions can be applied on each group
  - *Optimization:* combine tuples in the same group during run generation and intermediate merges, by computing partial aggregate values
    - ▶ For count, min, max, sum: keep aggregate values on tuples found so far in the group
      - When combining partial aggregate for count, add up the aggregates
    - ▶ For avg, keep sum and count, and divide sum by count at the end



# Module Summary

- Understood the overall flow for Query Processing and defined the Measures of Query Cost
- Studied the algorithms for processing Selection Operations, Sorting, Join Operations and a few Other Operations

NPTEL



# Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

**Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors.**

**Edited and new slides are marked with “PPD”.**



# Database Management Systems

## Module 39: Query Processing and Optimization/2: Optimization

**Partha Pratim Das**

*Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur*

[ppd@cse.iitkgp.ernet.in](mailto:ppd@cse.iitkgp.ernet.in)

**Srijoni Majumdar  
Himadri B G S Bhuyan  
Gurunath Reddy M**



**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
[www.db-book.com](http://www.db-book.com)



# Module Recap

- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations

NPTEL



# Module Objectives

- To understand the basic issues for optimizing queries
- To understand how transformation of Relational Expressions can create alternates for optimization

NPTEL



# Module Outline

- Introduction to Query Optimization
- Transformation of Relational Expressions

NPTEL



- **Introduction to Query Optimization**
- Transformation of Relational Expressions

NPTEL

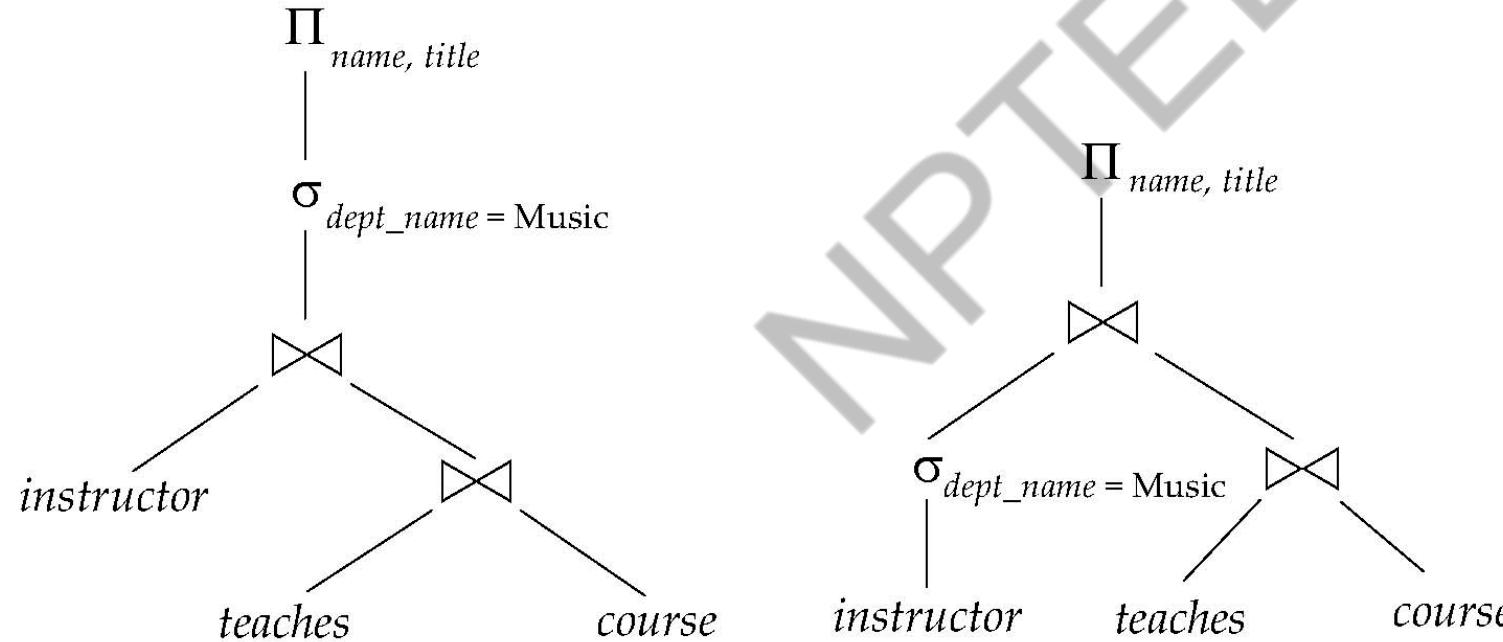
# INTRODUCTION TO QUERY OPTIMIZATION



# Introduction

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation

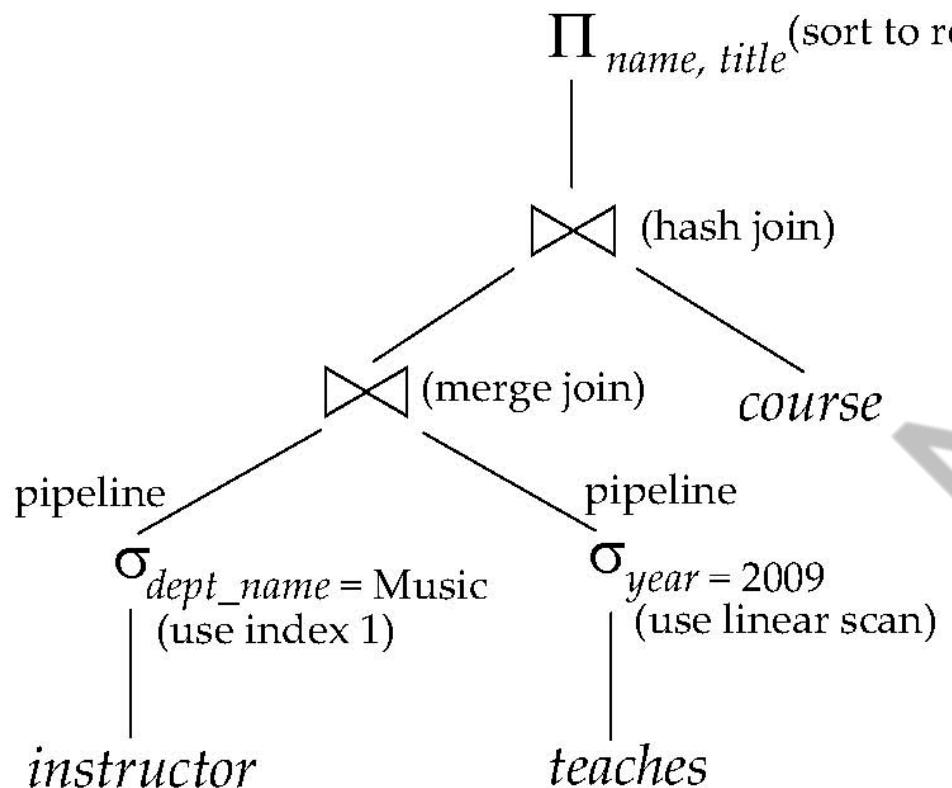
*course(course id, title, dept name, credits)*  
*instructor(ID, name, dept name, salary)*  
*teaches(ID, course id, sec id, semester, year)*





# Introduction (Cont.)

- An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated

$$\Pi_{name, title}(\sigma_{dept\_name = "Music"} \wedge year = 2009 (instructor \bowtie teaches \bowtie course))$$


course(*course id*, *title*, *dept name*, *credits*)  
instructor(*ID*, *name*, *dept name*, *salary*)  
teaches(*ID*, *course id*, *sec id*, *semester*, *year*)

- Find out how to view query execution plans on your favorite database



# Introduction (Cont.)

- Cost difference between evaluation plans for a query can be enormous
  - E.g. seconds vs. days in some cases
- Steps in **cost-based query optimization**
  1. Generate logically equivalent expressions using **equivalence rules**
  2. Annotate resultant expressions to get alternative query plans
  3. Choose the cheapest plan based on **estimated cost**
- Estimation of plan cost based on:
  - Statistical information about relations.
    - ▶ Examples: number of tuples, number of distinct values for an attribute
  - Statistics estimation for intermediate results
    - ▶ to compute cost of complex expressions
  - Cost formulae for algorithms, computed using statistics



- Introduction to Query Optimization
- **Transformation of Relational Expressions**

NPTEL

# TRANSFORMATION OF RELATIONAL EXPRESSIONS



# Transformation of Relational Expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every *legal* database instance
  - Note: order of tuples is irrelevant
  - We do not care if they generate different results on databases that violate integrity constraints
- In SQL, inputs and outputs are multisets of tuples
  - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance.
- An **equivalence rule** says that expressions of two forms are equivalent
  - Can replace expression of first form by second, or vice versa



# Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection operations are commutative

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted

$$\Pi_{L_1}(\Pi_{L_2}(\dots(\Pi_{L_n}(E))\dots)) = \Pi_{L_1}(E)$$

4. Selections can be combined with Cartesian products and theta joins

a.  $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$

b.  $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$



# Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative

$$E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$$

6. (a) Natural join operations are associative:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

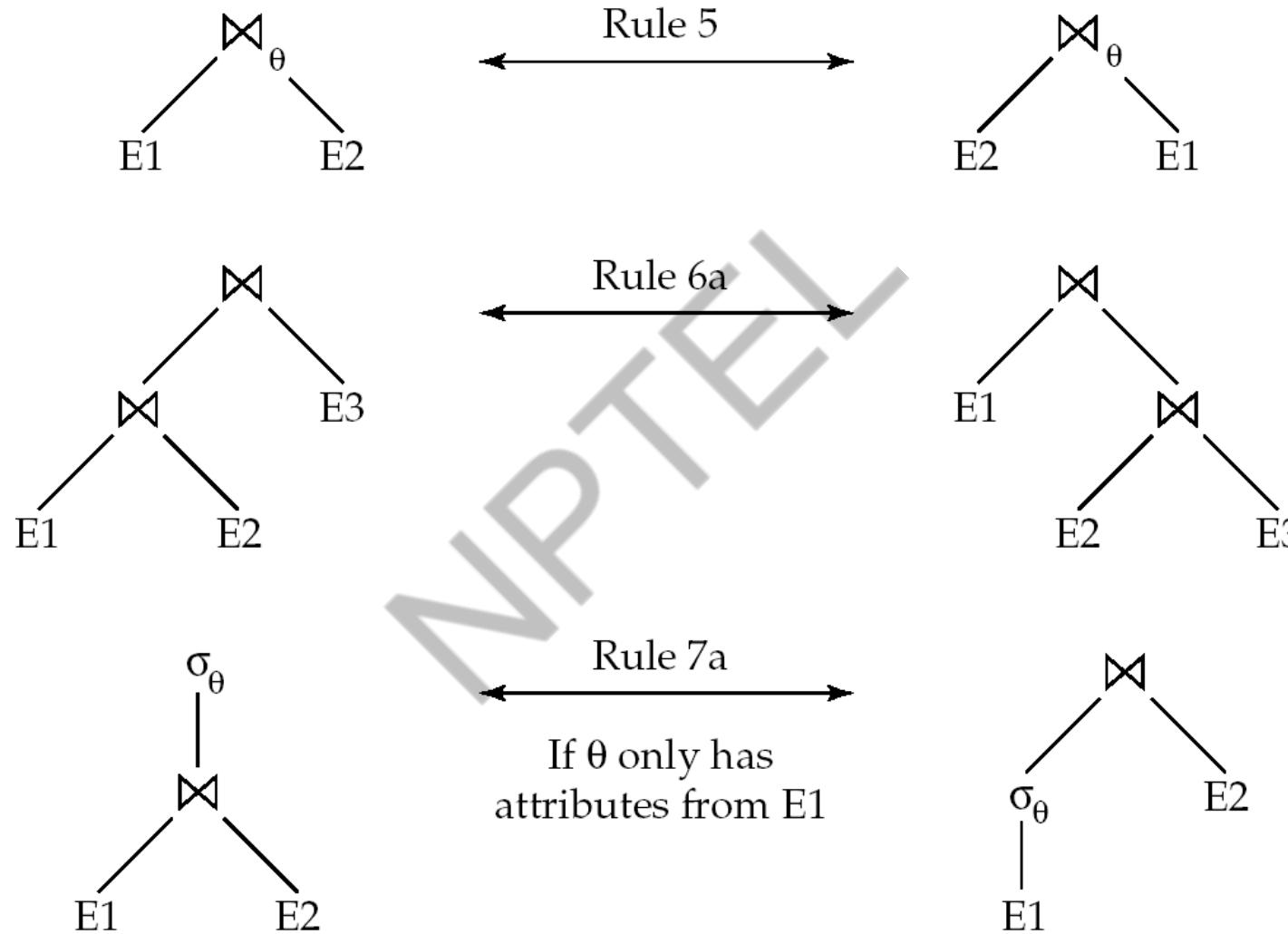
- (b) Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

where  $\theta_2$  involves attributes from only  $E_2$  and  $E_3$ .



# Pictorial Depiction of Equivalence Rules





# Equivalence Rules (Cont.)

7. The selection operation distributes over the theta join operation under the following two conditions:
  - (a) When all the attributes in  $\theta_0$  involve only the attributes of one of the expressions ( $E_1$ ) being joined

$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

- (b) When  $\theta_1$  involves only the attributes of  $E_1$  and  $\theta_2$  involves only the attributes of  $E_2$ .

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$



# Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation as follows:

- (a) if  $\theta$  involves only attributes from  $L_1 \cup L_2$ :

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = (\Pi_{L_1}(E_1)) \bowtie_{\theta} (\Pi_{L_2}(E_2))$$

- (b) Consider a join  $E_1 \bowtie_{\theta} E_2$ .

- Let  $L_1$  and  $L_2$  be sets of attributes from  $E_1$  and  $E_2$ , respectively
- Let  $L_3$  be attributes of  $E_1$  that are involved in join condition  $\theta$ , but are not in  $L_1 \cup L_2$ , and
- Let  $L_4$  be attributes of  $E_2$  that are involved in join condition  $\theta$ , but are not in  $L_1 \cup L_2$ .

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4}(E_2)))$$



# Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$\begin{aligned} E_1 \cup E_2 &= E_2 \cup E_1 \\ E_1 \cap E_2 &= E_2 \cap E_1 \end{aligned}$$

■ (set difference is not commutative).

10. Set union and intersection are associative.

$$\begin{aligned} (E_1 \cup E_2) \cup E_3 &= E_1 \cup (E_2 \cup E_3) \\ (E_1 \cap E_2) \cap E_3 &= E_1 \cap (E_2 \cap E_3) \end{aligned}$$

11. The selection operation distributes over  $\cup$ ,  $\cap$  and  $-$ .

$$\sigma_{\theta} (E_1 - E_2) = \sigma_{\theta} (E_1) - \sigma_{\theta} (E_2)$$

and similarly for  $\cup$  and  $\cap$  in place of  $-$

Also:  $\sigma_{\theta} (E_1 - E_2) = \sigma_{\theta} (E_1) - E_2$   
and similarly for  $\cap$  in place of  $-$ , but not for  $\cup$

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$



# Exercise

- Create equivalence rules involving
  - The group by/aggregation operation
  - Left outer join operation

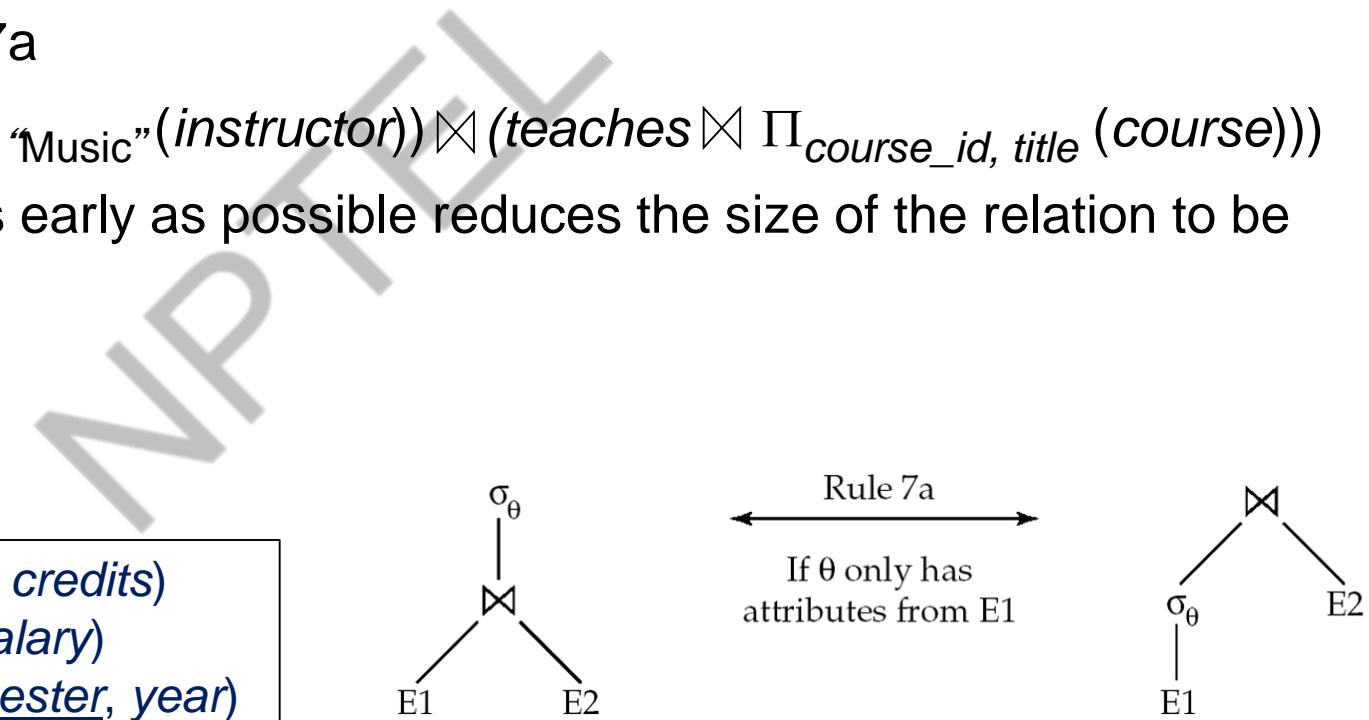
NPTEL



# Transformation Example: Pushing Selections

- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
  - $\Pi_{name, title}(\sigma_{dept\_name= "Music"}(instructor) \bowtie (teaches \bowtie \Pi_{course\_id, title}(course)))$
- Transformation using rule 7a
  - $\Pi_{name, title}((\sigma_{dept\_name= "Music"}(instructor)) \bowtie (teaches \bowtie \Pi_{course\_id, title}(course)))$
- Performing the selection as early as possible reduces the size of the relation to be joined

course(course id, title, dept name, credits)  
instructor(ID, name, dept name, salary)  
teaches(ID, course id, sec id, semester, year)



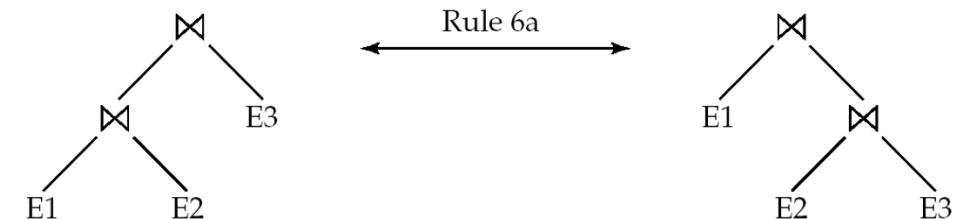


# Example with Multiple Transformations

- Query: Find the names of all instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught
  - $\Pi_{name, title}(\sigma_{dept\_name = "Music"} \wedge year = 2009 (instructor \bowtie (teaches \bowtie \Pi_{course\_id, title} (course))))$
- Transformation using join associatively (Rule 6a):
  - $\Pi_{name, title}(\sigma_{dept\_name = "Music"} \wedge year = 2009 ((instructor \bowtie teaches) \bowtie \Pi_{course\_id, title} (course)))$
- Second form provides an opportunity to apply the “perform selections early” rule, resulting in the subexpression

$$\sigma_{dept\_name = "Music"} (instructor) \bowtie \sigma_{year = 2009} (teaches)$$

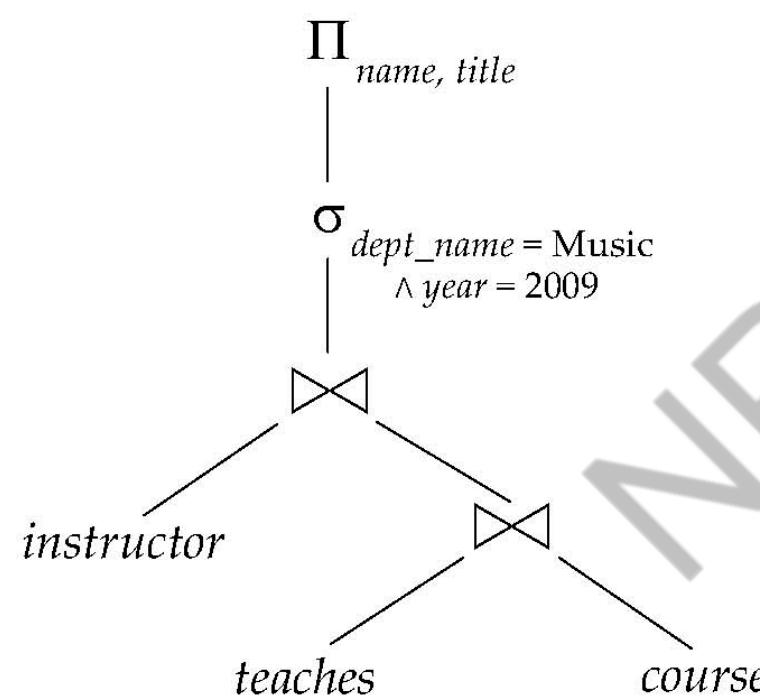
course(course id, title, dept name, credits)  
instructor(ID, name, dept name, salary)  
teaches(ID, course id, sec id, semester, year)



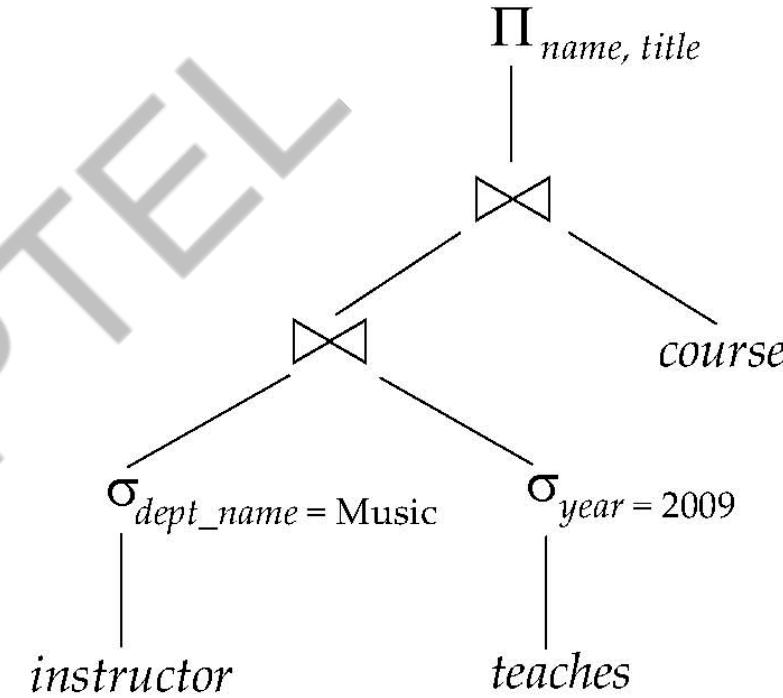


# Multiple Transformations (Cont.)

*course(course id, title, dept name, credits)*  
*instructor(ID, name, dept name, salary)*  
*teaches(ID, course id, sec id, semester, year)*



(a) Initial expression tree



(b) Tree after multiple transformations



# Transformation Example: Pushing Projections

- Consider:  $\Pi_{name, title}(\sigma_{dept\_name = "Music"}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id, title}(course)$
- When we compute

$$(\sigma_{dept\_name = "Music"}(instructor) \bowtie teaches)$$

we obtain a relation whose schema is:

$(ID, name, dept\_name, salary, course\_id, sec\_id, semester, year)$

- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:

$$\Pi_{name, title}(\Pi_{name, course\_id}(\sigma_{dept\_name = "Music"}(instructor) \bowtie teaches)) \bowtie \Pi_{course\_id, title}(course)$$

- Performing the projection as early as possible reduces the size of the relation to be joined

*course(course id, title, dept name, credits)  
instructor(ID, name, dept name, salary)  
teaches(ID, course id, sec id, semester, year)*

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$



# Join Ordering Example

- For all relations  $r_1$ ,  $r_2$ , and  $r_3$ ,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

(Join Associativity)

- If  $r_2 \bowtie r_3$  is quite large and  $r_1 \bowtie r_2$  is small, we choose

$$(r_1 \bowtie r_2) \bowtie r_3$$

so that we compute and store a smaller temporary relation



# Join Ordering Example (Cont.)

- Consider the expression

$$\Pi_{name, title}(\sigma_{dept\_name = \text{``Music''}}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id, title}(course)))$$

- Could compute  $teaches \bowtie \Pi_{course\_id, title}(course)$  first, and join result with  
 $\sigma_{dept\_name = \text{``Music''}}(instructor)$

but the result of the first join is likely to be a large relation

- Only a small fraction of the university's instructors are likely to be from the Music department
  - it is better to compute

$$\sigma_{dept\_name = \text{``Music''}}(instructor) \bowtie teaches$$

first

```
course(course id, title, dept name, credits)
instructor(ID, name, dept name, salary)
teaches(ID, course id, sec id, semester, year)
```

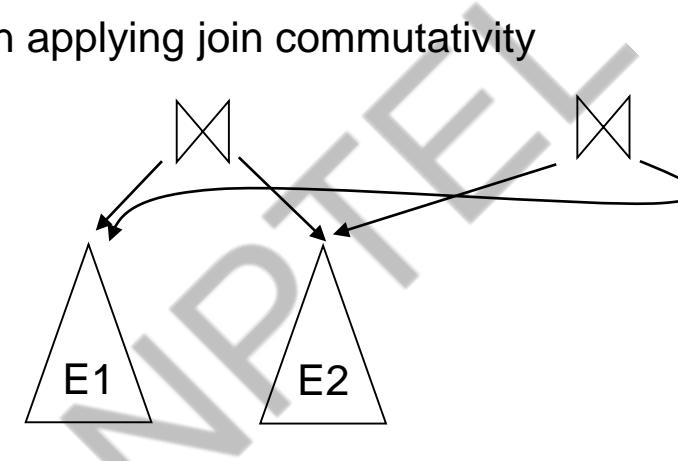


# Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions as follows:
  - Repeat
    - ▶ apply all applicable equivalence rules on every subexpression of every equivalent expression found so far
    - ▶ add newly generated expressions to the set of equivalent expressions
  - Until no new equivalent expressions are generated above
- The above approach is very expensive in space and time
  - Two approaches
    - ▶ Optimized plan generation based on transformation rules
    - ▶ Special case approach for queries with only selections, projections and joins



# Implementing Transformation Based Optimization

- Space requirements reduced by sharing common sub-expressions:
  - when E1 is generated from E2 by an equivalence rule, usually only the top level of the two are different, subtrees below are the same and can be shared using pointers
    - ▶ E.g. when applying join commutativity
- 

  - Same sub-expression may get generated multiple times
    - ▶ Detect duplicate sub-expressions and share one copy

- Time requirements are reduced by not generating all expressions
  - Dynamic programming



# Module Summary

- Understood the basic issues for optimizing queries
- For every relational expression, usually there are a number of equivalent expressions that can be created by simple transformations
- Final execution plan can be created by choose the estimated least cost expression from the alternates

NPTEL



# Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

**Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors.**

**Edited and new slides are marked with “PPD”.**



# Database Management Systems

## Module 40: Course Summarization

*Material presented  
in this module is  
for overall  
understanding.*

*These are not  
included in the  
assignments or  
examination.*

**Partha Pratim Das**

*Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur*

[ppd@cse.iitkgp.ernet.in](mailto:ppd@cse.iitkgp.ernet.in)

**Srijoni Majumdar  
Himadri B G S Bhuyan  
Gurunath Reddy M**



**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
[www.db-book.com](http://www.db-book.com)



# Course Recap

- What did we do?

NPTEL



# Week 01: Introduction to DBMS & Relational Model

- **Module 01: Course Overview**
  - Why Databases?
  - KYC: Know Your Course
- **Module 02: Introduction to DBMS/1**
  - Levels of Abstraction
  - Schema & Instance
  - Data Models
  - DDL & DML
  - SQL
  - Database Design
- **Module 03: Introduction to DBMS/2**
  - Database Design
  - Database Engine, Users, Architecture
  - History of DBMS
- **Module 04: Introduction to Relational Model/1**
  - Attribute Types
  - Relation Schema and Instance
  - Keys
  - Relational Query Languages
- **Module 05: Introduction to Relational Model/2**
  - Relational Operations
  - Aggregate Operations



# Week 02: SQL – Introductory & Intermediate

- **Module 06: Introduction to SQL/1**
  - History of SQL
  - Data Definition Language (DDL)
  - Basic Query Structure (DML)
- **Module 07: Introduction to SQL/2**
  - Additional Basic Operations
  - Set Operations
  - Null Values
  - Aggregate Functions
- **Module 08: Introduction to SQL/3**
  - Nested Subqueries
  - Modification of the Database
- **Module 09: Intermediate SQL/1**
  - Join Expressions
  - Views
  - Transactions
- **Module 10: Intermediate SQL/2**
  - Integrity Constraints
  - SQL Data Types and Schemas
  - Authorization



# Week 03: Advanced SQL and ER Modeling

- **Module 11: Advanced SQL**
  - Accessing SQL From a Programming Language
  - Functions and Procedural Constructs
  - Triggers
- **Module 12: Formal Relational Query Languages**
  - Relational Algebra
  - Tuple Relational Calculus (Overview only)
  - Domain Relational Calculus (Overview only)
  - Equivalence of Algebra and Calculus
- **Module 13: Entity-Relationship Model/1**
  - Design Process
  - E-R Model
- **Module 14: Entity-Relationship Model/2**
  - E-R Diagram
  - E-R Model to Relational Schema
- **Module 15: Entity-Relationship Model/3**
  - Extended E-R Features
  - Design Issues



# Week 04: RDBMS Design – Dependency & Normal Form

- **Module 16: Relational Database Design/1**
  - Features of Good Relational Design
  - Atomic Domains and First Normal Form
  - Functional Dependencies
- **Module 17: Relational Database Design/2**
  - Decomposition Using Functional Dependencies
  - Functional Dependency Theory
- **Module 18: Relational Database Design/3**
  - Algorithms for Functional Dependencies
  - Lossless Join Decomposition
  - Dependency Preservation
- **Module 19: Relational Database Design/4**
  - Normal Forms
  - Decomposition to 3NF
  - Decomposition to BCNF
- **Module 20: Relational Database Design/5**
  - Multivalued Dependencies
  - Decomposition to 4NF
  - Database-Design Process
  - Modeling Temporal Data



# Week 05: Application Design and Storage

- **Module 21: Application Design and Development/1**
  - Application Programs and User Interfaces
  - Web Fundamentals
  - Servlets and JSP
- **Module 22: Application Design and Development/2**
  - Application Architectures
  - Rapid Application Development
  - Application Performance
  - Application Security
  - Mobile Apps
- **Module 23: Application Design and Development/3**
  - Case Studies of Database Applications
- **Module 24: Storage and File Structure/1 (Storage)**
  - Overview of Physical Storage Media
  - Magnetic Disks
  - RAID
  - Tertiary Storage
- **Module 25: Storage and File Structure/2 (File Structure)**
  - File Organization
  - Organization of Records in Files
  - Data-Dictionary Storage
  - Storage Access



# Week 06: Indexing and Hashing

- **Module 26: Indexing and Hashing/1 (Indexing/1)**
  - Basic Concepts of Indexing
  - Ordered Indices
- **Module 27: Indexing and Hashing/2 (Indexing/2)**
  - Balanced Binary Search Trees
  - 2-3-4 Tree
- **Module 28: Indexing and Hashing/3 (Indexing/3)**
  - B+-Tree Index Files
  - B-Tree Index Files
- **Module 29: Indexing and Hashing/4 (Hashing)**
  - Static Hashing
  - Dynamic Hashing
  - Comparison of Ordered Indexing and Hashing
  - Bitmap Indices
- **Module 30: Indexing and Hashing/5 (Index Design)**
  - Index Definition in SQL
  - Guidelines for Indexing



# Week 07: Transaction and Concurrency Control

- **Module 31: Transactions/1**
  - Transaction Concept
  - Transaction State
  - Concurrent Executions
- **Module 32: Transactions/2: Serializability**
  - Serializability
  - Conflict Serializability
- **Module 33: Transactions/3: Recoverability**
  - Recoverability and Isolation
  - Transaction Definition in SQL
  - View Serializability
  - Complex Notions of Serializability
- **Module 34: Concurrency Control/1**
  - Lock-Based Protocols
  - Implementing Locking
- **Module 35: Concurrency Control/2**
  - Deadlock Handling
  - Timestamp-Based Protocols



# Week 08: Recovery and Query Processing & Optimization

- **Module 36: Recovery/1**
  - Failure Classification
  - Storage Structure
  - Recovery and Atomicity
  - Log-Based Recovery
- **Module 37: Recovery/2**
  - Recovery Algorithm
  - Recovery with Early Lock Release
- **Module 38: Query Processing and Optimization/1: Processing**
  - Overview of Query Processing
  - Measures of Query Cost
  - Selection Operation
  - Sorting
  - Join Operation
  - Other Operations
- **Module 39: Query Processing and Optimization/2: Optimization**
  - Introduction to Query Optimization
  - Transformation of Relational Expressions
- **Module 40: Course Summarization**



# Module Objectives

- The space of RDBMSs is crowded. We take a look into common RDBMS systems
- Non-Relational database systems are starting to dominate emerging applications. We present a brief overview
- What is the road forward? We outline likely job profiles in terms of a skills-profiles matrix and the companies to work for

NPTEL



# Module Outline

- Common RDBMSs
- Non-relational DBMSs
- Skill – Job Profile Matrix

NPTEL



- Common RDBMSs
- Non-relational DBMSs
- Skill – Job Profile Matrix

# COMMON RDBMS

Ref: [https://en.wikipedia.org/wiki/Comparison\\_of\\_relational\\_database\\_management\\_systems](https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems)

Ref: <http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc38151.1540/doc/html/san1278453579697.html>

Ref: [https://www.ibm.com/support/knowledgecenter/en/SSEPGG\\_11.1.0/com.ibm.db2.luw.welcome.doc/doc/welcome.html](https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.welcome.doc/doc/welcome.html)

Ref: <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-develop-cplusplus-simple>



# Relational Databases

- Based on the relational model of data
- This model organizes data into one or more tables (or “relations”) of rows and columns, with a unique key for each row
- Since each row in a table has its own unique key, rows in a table can be linked to rows in other tables by storing the unique key of the row to which it should be linked (where such unique key is known as a “foreign key”)
- Mostly the relational database systems use SQL (Structured Query Language) as the language for querying and maintaining the database
- The reasons for the dominance of relational databases are: simplicity, robustness, flexibility, performance, scalability and compatibility in managing generic data
- The most popular of these are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2.
- These RDBMS's are mostly used in large enterprise scenarios



# Common RDBMS

- Commercial Software
  - Oracle (Oracle)
  - Sybase (Sybase Corporation / SAP AG)
  - DB2 (IBM)
  - SQL Server (Microsoft)
  - Teradata (Caltech and Citibank)
- Free / GPL / Open Source:
  - PostgreSQL (PostgreSQL Global Development Group)
  - MySQL (MySQL AB / Oracle Corporation)
- Object Oriented DBMSs



# Oracle

- Multi-model commercial database management system produced and marketed by **Oracle Corporation**.
- Larry Ellison, Bob Miner and Ed Oates started a consultancy called Software Development Laboratories (SDL) in 1977, and developed the original version of Oracle.
- Latest Version: **Oracle Database 12c Release 2: 12.2.0.1 (patchset as of March 2017)**
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
- Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)
- Tools/ Editions: Oracle SQL Developer, Oracle Forms, Oracle Jdeveloper, Oracle Reports for development of applications, Oracle Live SQL for test environment
- Oracle can be accessed from Java through JDBC, Microsoft.NET through ODP.NET, C, C++ through OCI, ODBC, ODPI-C, Python through cx\_Oracle



# Sybase

- Relational model database server product for businesses developed by **Sybase Corporation which became part of SAP AG**.
- Originally for unix platforms in 1987, Sybase Corporation's primary DBMS product was initially marketed under the name Sybase SQL Server.
- Latest Version: **Sybase 16, released on 2014**
- Languages: Sybase IQ, Transact-SQL
- Tools/ Editions: Sybase SQL server for development of applications. Has a developer and express edition.
- Sybase can be accessed from C, C++ through SQLAPI++, Java through JDBC



# DB2

- Db2 contains database-server products developed by **IBM**. Mostly relational models, but now includes object relational models
- In 1970, Edgar F.Codd, researcher in IBM published the model for data manipulation.
- Latest Version: **DB2 LUW 11.1 released on 2016**
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
- Languages: Structured Query language (SQL), XML Query
- Tools/ Editions: Advanced Enterprise Server Edition, Enterprise Server Edition, Advanced Workgroup Server Edition, Workgroup Server Edition, Direct and Developer Editions and Express-C.
- Db2 can be accessed from C, C++, Java, Ruby, Perl through a package of DB2 API's



# SQL Server

- Relational database management system developed by **Microsoft**.
- SQL Server 1.0, a 16-bit server for the OS/2 operating system in 1989
- Latest Version: **SQL Server 2017**
- Used for running online transaction processing (OLTP) and online analytical processing (OLAP)
- Languages: Transact SQL
- Tools/ Editions: Enterprise, Standard, Web, Business Intelligence, WorkGroup, Express
- SQL server can be accessed from Java through JDBC, C, C++ through ODBC



# Teradata

- Relational database management system developed by **Caltech and Citibank's advanced technology group**
- In 1984, the first version of Teradata was released
- Latest Version: **Teradata 15**
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
- Languages: BTEQ(Basic Teradata query)
- Tools/ Editions: Developer Edition, Express Edition
- SQL server can be accessed from Java through JDBC, C, C++ through ODBC



# PostgreSQL

- Open source relational database management system produced by **PostgreSQL Global Development Group**, a diverse group of many companies and individual contributors.
- First version in 1988 by researchers of POSTGRES project
- Latest Version: **PostgreSQL 10.3 released on 2018**
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads, Supports big data analytics
- Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)
- Oracle can be accessed from Java through JDBC, Microsoft.NET through npgsql, C, C++ through libpq,



# MYSQL

- Open source relational database management system produced by **Swedish company MySQL AB, now owned by Oracle Corporation**
- First internal release on 23 May 1995
- Latest Version: **MySQL 8.0.4 released on 2018**
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
- Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)
- Oracle can be accessed from Java through JDBC, Microsoft.NET through ADO.NET, C, C++ through ODBC



# Object-oriented DBMS (OODBMSs)

- Combines database capabilities with object oriented programming language capabilities
- OODBMSs allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS
- Objects have a many to many relationship and are accessed by the use of pointers.
- Access to data can be faster because an object can be retrieved directly without a search, by following pointers.
- Most object databases also offer some kind of query language, allowing objects to be found using a declarative programming approach.
- Examples:
  - Objectivity/DB,
  - O2,
  - Object Store

Ref: [https://en.wikipedia.org/wiki/Object\\_database](https://en.wikipedia.org/wiki/Object_database)



# Parameters

- We compare the RDMBSs based on the following parameters:
  - OS support
  - Fundamental features
  - Limits
  - Tables and views
  - Indexes
  - Database capabilities
  - Data types
  - Other objects
  - Partitioning
  - Access control
  - Programming Language Support

NPTEL



# Comparative Study

## OS support

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Linux, Window, Mac, Unix, Haiku, z/OS, OpenVMS	Linux, Window, Mac, Unix, BareMetal, Android	Linux, Window	Linux, Window, Mac (Express- C image), OS/2, Unix, z/OS, iOS	Linux, Window, Mac, Unix,	Linux, Window, Mac, Unix, BSD, AmigaOS, z/OS, Android	Linux, Window, Mac, Unix, BSD, AmigaOS, z/OS, Android



# Comparative Study

## Basic Features

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports ACID properties for transactions, implicit commit for DDL, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, hash and partition for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control



# Comparative Study

## Limits

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Max DB Size: 2PB	Max DB Size: 104TB	Max DB Size: 524,272 TB	Max DB Size: Unlimited	Max DB Size: Unlimited	Max DB Size: Unlimited	Max DB Size: Unlimited
Max Table Size: 4GB * block size	Max Table Size: File size	Max Table Size: 524,272 TB	Max Table Size: 2 ZB	Max Table Size: Unlimited	Max Table Size: 32 TB	Max Table Size: 256 TB
Max Row Size: 8KB	Max Row Size: File size	Max Row Size: 2TB	Max Row Size: 32,677 B	Max Row Size: 64 GB	Max Row Size: 1.6TB	Max Row Size: 64KB
Max Column per Row: 1,000	Max Column per Row: 45,000	Max Column per Row: 1,024	Max Column per Row: 1,012	Max Column per Row: 2048	Max Column per Row: 1600	Max Column per Row: 4096
Max CHAR size: 32,767 B	Max CHAR size: 2GB	Max CHAR size: 2GB	Max CHAR size: 32 KB	Max CHAR size: 64,000 bits	Max CHAR size: 1GB	Max CHAR size: 64 KB
Max Number size: 126 bits	Max Number size: 64 bits	Max Number size: 126 bits	Max Number size: 64 bits	Max Number size: 38 bits	Max Number size: Unlimited	Max Number size: 64 bits
Max Column Name size: 128		Max Column Name size: 128	Max Column Name size: 128	Max Column Name size: 128	Max Column Name size: 63	Max Column Name size: 64



# Comparative Study

## Tables and Views

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables (apart from basic)

## Type System

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Static+Dynamic	Static	Static	Static+Dynamic	Static	Static	Static



# Comparative Study

## Data Types

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; And other miscellaneous types like Spacial, Image, Audio, Dicom, Video	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Boolean	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Boolean	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit



# Comparative Study

## Indexes

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports	Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports Hash, Partial, Bitmap,	Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports R/R++, Hash, Partial, Bitmap, Reverse
Apart from Basic B/B++ indexes	only Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes



# Comparative Study

## Database Capabilities

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports  Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports  Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports  Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports  Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports  Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports  Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Blobs and Clobs	Supports  Union, Outer Joins, Except, Inner Selects, Blobs and Clobs, Common Table Expressions



# Comparative Study

## Other Objects

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Supports Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Supports Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Supports Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Supports Cursor, Trigger, Function, Procedure, External Routine	Supports Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Supports Cursor, Trigger, Function, Procedure, External Routine



# Comparative Study

## Partitioning

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports Range, Hash, Composite, List	Supports none	Supports Range, Hash, Composite, List	Supports Range, Hash, Composite, List	Supports Range, Hash, Composite, List	Supports Range, Hash, Composite, List	Supports Range, Hash, Composite, List



# Comparative Study

## Access Control

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports  Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Supports  Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Supports  Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Supports  Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Supports  Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Supports  Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Supports  Native network encryption, Enterprise Directory compatibility, Patch Access



- Common RDBMSs
- **Non-relational DBMSs**
- Skill – Job Profile Matrix

**Basis for these DBMSs have not been covered in the course**

# NON-RELATIONAL DATABASES

- Ref: <https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models>
- Ref: <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>
- Ref: <https://www.mongodb.com/scale/what-is-a-non-relational-database>
- Ref: <http://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/>
- Ref: <https://www.mongodb.com/scale/relational-vs-non-relational-database>
- Ref: <https://dzone.com/articles/eclipse-jnosql-a-quick-overview-with-redis-cassand>
- Ref: <http://coreviewsystems.com/nosql-databases-key-value-store/>



# What is Big Data?

- Big data is data sets that are so voluminous and complex that traditional data-processing application software are inadequate to deal with them
- Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source
- **5V's (characteristics) of big data:**
  - **Volume:** The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.
  - **Variety:** The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
  - **Velocity:** In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time.
  - **Variability:** Inconsistency of the data set can hamper processes to handle and manage it.
  - **Veracity:** The data quality of captured data can vary greatly, affecting the accurate analysis



# Why do we need non-relational databases?

- To effectively support Big Data

NPTEL



# Non-Relational Databases

- Does not follow the relational model
- Offer flexible schema design
- Handle unstructured data that does not fit neatly into rows and columns
- Typically Open Source
- Scalable using cheap commodity servers
- The popular ones are:
  - MongoDB, DocumentDB, Cassandra, Couchbase, HBase, Redis, and Neo4j
- These databases are usually grouped into four categories:
  - Key-value stores
  - Graph stores
  - Column stores, and
  - Document stores

Non-Relational Databases are also known as NoSQL Databases



# Relational vs. Non-Relational

	Relational	Non-Relational
<b>Flexible Data Model</b>	Works on only structured data (relational tables)	Can handle unstructured and semi structured data (xml, json), along with dynamic modification of schema. Suited for BigData
<b>Cost, Complexity, Speed</b>	Faster but less capable operations, Cheaper and less complex	Much more database operations supported, Highly complex internally and costlier
<b>Performance and Scalability</b>	Incurs issues, as data integrity needs to be maintained at all levels, in case of distributed architecture	Better Scalability and performance by exploring distributed systems using sharding and partitioning
<b>Consistency</b>	Enforces strong consistent systems. Less overhead for developers	Enforces eventual consistent systems. More overhead for developers
<b>Enterprise Management and Integrations</b>	Fits into the Enterprise IT stack, much more secure and robust	Designed to cope with agility of modern cloud based applications



# Types of NoSQL Databases

## 1. Key-value stores:

- These type of databases work by matching keys with values, similar to a dictionary. There is no structure nor relation. **Example: Redis, MemcacheDB**

## 2. Graph stores:

- These use tree-like structures (i.e. graphs) with nodes and edges connecting each other through relations. **Example: OrientDB, Neo4J**

## 3. Column stores:

- Column-based NoSQL databases are two dimensional arrays whereby each key (i.e. row / record) has one or more key / value pairs attached to it and these management systems allow very large and unstructured data to be kept and used (e.g. a record with tons of information). **Example: Cassandra, Hbase**

## 4. Document stores:

- These DBMS work in a similar fashion to column-based ones; however, they allow much deeper nesting and complex structures to be achieved (e.g. a document, within a document, within a document).

Documents overcome the constraints of one or two level of key / value nesting of columnar databases. **Example: MongoDB, Couchbase**



# Comparative Study

## When to Use

Redis	MemcacheDB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Caching, Queuing frequent information, Keeping Live information, Supports lists, sets, queues and more	Caching, Queuing frequent information, Keeping Live information,	Handling complex relational information, Modelling and handling classifications	Handling complex relational information, Modelling and handling classifications	Keeping unstructured non-volatile information, useful for content management	Keeping unstructured non-volatile information, useful for content management	Works with deeply nested and complex data structures, JavaScript friendly, useful for content management	Works with deeply nested and complex data structures, JavaScript friendly, useful for content management



# Comparative Study

## Handling Relational Data

Redis	Memcache DB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Supports ACID, query only on key	Supports ACID, query only on key	Supports ACID and joins	Supports ACID and joins	Supports ACID, Multiple Queries	Supports ACID, Multiple Queries	Supports ACID, Multiple Queries, Nesting Data	Supports ACID, Multiple Queries, Nesting Data



# Comparative Study

## Performance

<b>Redis</b>	<b>Memcache DB</b>	<b>OrientDB</b>	<b>Neo4J</b>	<b>Cassandra</b>	<b>Hbase</b>	<b>MongoDB</b>	<b>Couchbase</b>
Highly Scalable, Highly Flexible, not complex in terms of representation and use	Highly Scalable, Highly Flexible, not complex in terms of representation and use	Variable Scalability, Highly Flexible, Highly complex in terms of representation and use	Variable Scalability, Highly Flexible, Highly complex in terms of representation and use	Highly Scalable, Moderately Flexible, Moderately complex in terms of representation and use	Highly Scalable, Moderately Flexible, Moderately complex in terms of representation and use	Highly Scalable, Highly Flexible, Moderately complex in terms of representation and use	Highly Scalable, Highly Flexible, Moderately complex in terms of representation and use



- Common RDBMSs
- Non-relational DBMSs
- **Skill – Job Profile Matrix**

NPTEL

# SKILL – JOB PROFILE MATRIX



# Skill – Profile Mapping

Skills	Years of Experience	Under-standing Specs and Schema	Coding Query in SQL (DML)	Analyzing Specification, Schema Design and Normalization (DDL)	Application / Database Architecture, Indexing, Performance Optimization	Database Administration	Databases, Algorithms, Architecture, Compilers, ...	Data Mining, Machine Learning, Big Data, Programming
<b>Job Profiles</b>								
Application Programmer	0-4	X	X					
Senior Application Programmer	2-6	X	X	X	X			
Database Analyst / Architect	4-8	X	X	X	X			
Database Administrator	8-10	X	X	X	X	X		
Database Engineer (multiple grades)	2-10	X	X	X	X	X	X	
Big Data Programmer, Analyst	0-6	X	X	X	X			X



# Companies in RDBMS & Big Data Space

All consultancies and product development companies use database management systems as backend:

1. TCS
2. Cognizant Ltd
3. IBM
4. Lexmark
5. Accenture
6. Intel
7. Amazon
8. NetApp
9. Ericsson
10. PWC
11. Deloitte
12. Tech mahindra
13. Wipro
14. HCL
15. Mindtree
16. Mphasis
17. Capgemini
18. Microsoft
19. Adobe
20. and so on.

DB Application Development

Companies working to develop new DBMS products and provide services:

1. Oracle - Oracle
2. Teradata - Teradata
3. IBM - DB2, Informix
4. Microsoft - SQL Server, My SQL
5. SAP - Sybase
6. Amazon - SimpleDB

DB System Development

Companies working to develop new DBMS products in NoSQL area and providing services:

1. KPMG, Google, SAP, Nokia, Adobe, Facebook - MongoDB (Document)
2. Microsoft, Ebay, Walmart, Nvidia, Sony - Cassandra (Column)
3. Uber, Twitter, Instagram, Airbnb - Redis (key value)
4. Blockchain, IBM, Walmart, ebay, Cisco, Microsoft - Neo4j (graph stores)

Big Data Application or System Development



# Final Words

- Read the DBMS Text book thoroughly and solve exercises
- Practice query coding
- Practice database design from specs
- Besides DBMS, develop good knowledge in programming, data structure, algorithms and discrete structures
- Seek help, if you need to – mail us

NPTEL



# Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

**Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors.**

**Edited and new slides are marked with “PPD”.**