Database Systems END_SEMS

MID 1

- **Intro Slide**
    - Data, Information, Database, DBMS, Database System, Metadata
    - Computer System Components
    - Database Functionality [4]
    - Three Schema Architecture
        - Two reasons for proposal [Prog Data Independence, Support for multiple views]
        - Three Schemas and their definitions
        - Diagram
        - Mappings
    - Miniworld
    - Data Independence
        - Logical Data Independence
        - Physical Data Independence
    - DDL
        - SDL
        - VDL
    - DML
        - Host Language
        - Query Language
    - High Level Language - Non Procedural
    - Low Level Language - Procedural
    - ER Models
    - Hierarchical Model
        - Tree Structure, Links
        - Difference with Network
        - DAG Structure, boxes and lines.
        - Diagram rules.
        - Single vs Many to many
    - Network Model
        - Restricted form of ER
        - Diagram rules
        - RLink and dummy records

- Relational Model
    - Column, Row, Table
    - Domain
    - Attribute
    - Relational Schema
- Integrity Constraints
    - Domain, Key, Entity, Referential Integrity
- SQL Summary
- Functional Dependencies
- Normalization
    - 1NF
    - 2NF
    - 3NF
    - BCNF
    - Algo for preserving NF in Join

- **Chapter 1**

    - Capabilities of DBMS
        - Persistent Storage
        - Programming Interface
        - Transaction Management
    - Problems with Megaton
        - Tuple Layout on Disk [Change, Deletion, ASCII Storage] all expensive.
        - Search expensive as no indexing, read entire file
        - Brute force querying
        - No buffer, cache
        - No concurrency control
        - No reliability
        - No Security
    - Main memory Buffer Management
    - Transaction Processing and Control
    - ACID
    - Query Processing
        - Parsing
        - Pre-processing
        - Optimiser

- **Chapter 2**

- Cache
  - Onboard, L2
  - Single/Multiprocessor system updation
- Virtual Memory
- Moore’s Law
  - Applicable on
  - Not applicable on
- Secondary Storage
  - Reading Writing
  - Block Allocation
- Tertiary Storage
- Volatility
- Disk
  - Platter, Head, Actuator, Cylinder
  - Track
  - Sectors, Gaps
  - Measures
- Time = Seek Time + Rotational Delay + Transfer Time + Other Delays
- Average Random Seek Time [Formula]
- What does other time consist of
- Why sequential access is better than random access
- Cost of reading vs writing [*With/without verification*]
- Typical values of all quantities
- Block Modification
- Block Address [4 constituents]
- **Secondary Storage Optimised Usage**
- Elevator Algorithm
- Disk Failures:
  - Intermitten - checksum parity
  - Media Decay
  - Drive Crash
  - Write failure
- Stable Storage
- Disk Mirroring
- Redundant Arrays of Independent Disks [RAID]
  - Solves 2 problems, transfer rate and reliability
- RAID Advantages:
  - Read speed due to multiple disks ***2x***
  - Read speed due to **Bit level Data Stripping** ***8x***
  - Redundancy via stripping and parity

- RAID Levels:
  - **LEVEL 0**
    - Block stripping, no redundancy, no parity
  - **LEVEL 1**
    - Disk Mirroring
  - **LEVEL 2**
    - Bit level stripping
  - **LEVEL 3**
    - Bit interleaved parity
  - **LEVEL 4**
    - Block interleaved parity
  - **LEVEL 5**
    - Block interleaved Distributed Parity
  - **LEVEL 6**
    - Additional information
  - **0-1 vs 1-0**

- **Chapter 4**

  - Index file is a sorted file
  - Search Key
  - Dense/Sparse Index
  - Efficient, permanently in main memory.
  - Multilevel index perks
  - Handling Duplicate Keys
  - Index modification on data manipulation:
    - To Manipulate Data:
      - Create overflow block
      - Insert new blocks in sequential order
      - slide tuples to adjacent block if no space.
    - **Index Modification Table**
  - **Secondary Index**
    - What it does
    - Why it's dense
    - Multi-level secondary index (can be sparse)
    - **Indirection** - When
  - Clustered File Structure
    - When
  - Inverted Indexes
    - When
    - Posting

- Vector Space Model
- Weighted Vector Space Queries
- Benefits of simple index:
  - Good for scans
  - simple
- Demerits:
  - insert expensive
  - Loss of sequentiality
- **B-Tree**
  - **LOGIC**
  - block usage [2]
  - indexing
  - **Rules**
    - At least n+1/2 pointers used
    - Notation for leaf and non-leaf.
    - leaf has a node pointing to the next leaf.
    - balanced
  - Minimum table
  - B-Tree lookup
  - B-Tree insertions
    - *Simple, Logical, just focus*
  - B-Tree Deletion
    - *Minimize the tree as much as possible*
  - **Efficient**
    - Few I/O
    - Rare splitting, merging
    - root in main memory
- **Hash Tables**
  - Basic logic
  - Hash Function
  - Secondary Storage
  - Insertion
  - Deletional - Consolidation
  - Good Hash function criterion [***equal distribution***]
  - keys sorted in a bucket if CPU is time critical, and insertion deletion not frequent
  - **Utilization**
    - 50-80 Rule
  - **Very Efficient:**
    - 1 Disk I/O for lookup
    - 2 for insetion deletion

- Blocks per bucket should be kept low
- **Dynamic Hash Tables**
  - **Extendable Hash Table**
    - Indirection introduced
    - Blocks power of 2
    - Insertion
    - Deletion
    - Overflow for duplicate
    - Less wasted space, no full reorganisation
    - Directory doubles in size, indirection
  - **Linear Hashing**
    - No of buckets remains less
    - i, r, n important, r/n determinal factor for creating new bucket
    - # used slots / # of slots = U is important, we expand file
    - Less wasted space, no full reorganisation
    - No indirection, can still have overflow chains

Mid 2

- **Chapter 6**

  - Major parts of Query compilation:
    - Parsing
    - Rewrite
    - Physical plan generation
    - *Last two steps called query optimisation*

  - **Operations:**
    - Union, Intersection and Differences
      UNION, INTERSECT, EXCEPT
      What each operator does.
      Requirement : Schema of argument relations much be identical
      ****BAG vs SET****
      UNION ALL for BAG
    - Selection
      WHERE
      Takes a Relation and a Condition as argument.
      Condition can be Arithmetic, Boolean or Comparison.

- Projection
  SELECT
  Can take:
  A single attribute of R or a list
  x —> y, to rename an attribute x of R to y
  E —> y, to rename an expression E to y
- Product
  Cartesian product formation, FROM, WHERE, SELECT
- Joins
  JOIN
  Natural Join
  Theta (*any op other than =*) Join
  Equi join
  Outer Join
  Left Outer Join
  Right Outer Join
- Duplicate Elimination
  DISTINCT
- Grouping
  GROUP BY
- Sorting
  ORDER BY
  Tau is the Sort by operator, takes a list of attributes, in order to sort

- Expression Tree
- Scanning Tables
  - *Table Scan*
  - *Index Scan*
- Sorting while scanning
  - BTree Read
  - Main Memory fit
  - Too large, then multi-way merging
- Notations:
  - B(R)
  - T(R)
  - V(R,a)
  - M
- Time slide
- Iterator
  - Open

  - GetNext
  - Close
- Operators:
  - Unary
  - Full Relation (need full view of R)
  - Full Relation, Binary

- **One Pass Algorithm**
  - Unary, Complexity
  - DISTINCT, Complexity
  - Grouping, Complexity
  - Union, process, Complexity
  - Intersection, process, Complexity
  - Difference, process, Complexity
  - Bag Intersection, process, Complexity
  - Bag Difference, process, Complexity
  - Product
  - Natural Join, Process , Complexity = B(R)+B(S)
  - Nested Loop Join, Complexity tuple-wise= T(R)*T(S)
  - Nested Loop Join, Complexity tuple-wise= B(R)*B(S)/M , Derivation
  - **TABLE IN SLIDES**

- **Two Pass Algorithms : Sort Based**
  - Distinct : C = 3*B(R), M = root(B(R)), B < M^2 required
  - Same for grouping and aggregation
  - Union, Intersection and Bag : C = 3*(B(R)+B(S)), required : (B(R)+B(S) < M^2
  - Simple Sort based Join : C = 5*(B(R)+B(S)), required : B(R)< M^2 and B(S) < M^2
  - Modification for worst case, Simple Sort based Join
  - Sort-Merge-Join C = 3*(B(R)+B(S)), required : B(R)+B(S) < M^2
  - Worst case, use Nested Loop Join

- **Two Pass Algorithm based Hashing**
  - Basic Idea
  - Partitioning Relation
  - Distinct, required : B(R) < M^2
  - Grouping Aggregation, C = 3*B(R), M = root(B(R))
  - Binary : C = 3*(B(R)+B(S)), required : B(R)+B(S) < M^2

  - Join : One pass of corresponding buckets : $C = 3*(B(R)+B(S))$, required : $\min(B(R),B(S)) < M^2$
  - Hybrid Hash Join : $C = (3 - (2M/B(S)))(B(R)+B(S))$

- Sort based vs Hash Based
  - Size
  - Sort Order
  - Bucket Size
  - Writing sub-lists

- **Index based Algorithms**
  - Clustered relations and index
  - For clustered, $C = B(R)/V(R,a)$
  - For non-clustered, $C = T(R)/V(R,a)$
  - Organisation:
    - Clustered File Organisation
    - Clustered Relation
    - Clustered Index
  - Join : $C = T(R)*T(S)/V(S,Y)$
  - Advantages of Index Join:
    - R is small, and V large, thus C is low.
    - If select before join, then most tuples not examined.
  - Join using sorted index

- **Buffer Management**
  - Purpose delay minimisation
  - Control main and virtual memory, target to avoid thrashing
  - To clear buffer when full
    - LRU
    - FIFO
    - Clock : upon read, use set to 1, manager removes 0, sets 1 to 0
    - System Control
  - Pinned Blocks, manager avoids

- Sort based Algos allow change of M.
- Hash based allow change in number of buckets as long as it fits in main memory

- **Parallel Algorithms**

- Modes:
  - Shared memory
  - Shared Disk
  - Shared Nothing
- Shared nothing has tuple-at-a-time operations in parallel
- Hash used to distribute operations
- No. Of disk I/Os remain constant, time reduces
- Time = 1/processor of time + cost of shipping

- **Chapter 7**

  - SQL Query -> Expression Tree -> Logical Plan Tree -> Physical Plan Tree
  - Steps:
    - Parsing
    - Rewrite
    - Physical Plan Generation
    - *Last two called **Query Optimiser***
  - Query -> Parser -> Pre-processor -> LQ Plan Generator -> Rewriter -> Preferred LQ Plan
  - **Parse Tree** - Nodes are atoms or syntactic categories
  - Pre-processor - Semantic checker
    - Relation use
    - Attribute use
    - Types
  - Commutativity, Associativity used used to improve query plans
  - Pushing Selections - Reduces no. of tuples **LAWS**
  - Pushing Projections - **LAWS**
  - Pushing Distinct - **LAWS**
  - Pushing Aggregation and Grouping - **LAWS**
  - **Parse Tree to LQ Tree**
    - Replaces nodes with Relational Algebra terms
    - Simplify plan
    - **Including Conditions through *two argument selection and next equate selection***

  - **Estimating Cost of Operations**
    - Estimating Projection
    - Estimating Selection
      - Inequality, assumption 1/3

- Estimate of Join : C = T(R)*T(S) / max (V(R), V(S))
- T(R)T(S)/max(V(R,y1), V(S,y1)) max (V(R,y2), V(S,y2)), for multiple attributes
- Estimation of other operations

- **Cost-based Plan Selection**
  - Incremental Computation of Statistics
    - T(R)+1
  - Approaches:
    - Top Down
    - Bottom Up : Selinger Style
  - Greedy Heuristic
  - Brach and Bound
  - Hill Climbing
  - Dynamic Programming
  - Selinger-style Optimisation

- **Choosing the Order of Joins**
  - Join Tree
  - **Left Deep Join Tree : Best, 2 reasons** [Limit permutations, Fit well with algos]
  - Right Deep Join Tree
  - Bushy Join Tree
  - Dynamic:
    - Consider All
    - Subset
    - Heuristic
  - Seliger style Optimisation
  - Greedy Algorithm

- **Physical Query Plan Selection**
  - Selection
  - Join
  - Materialisation vs Pipeline
  - Selection, Projection excellent for pipelining

Post Mid2

- **Chapter 8**

  - Failure Modes

    - Erroneous Data Entry — fix
    - Media Failure — fix (3)
    - Catastrophic Failure — fix
- System Failure
    - Transaction Failure
    - Power Failure
    - Main Memory Failure
- Transaction
- Link Diagram (*Transaction Manager, Buffer Manager, Query Processor, Log Manager, Recovery Manager, Disk*)
- Transaction Manager
- Log Manager
- Recovery Manager
- Elements
- Correctness Principle
- ACID
    - *Recovery Manager helps with A&D, also C while rollback.*
- Motivations for Atomicity and Durability
- Primitive Operations of Transaction
    - Interact with Database
    - Reading
    - Writing
    - Buffer Manager Task
- Primitives
    - Input
    - Output
    - Read
    - Write
- Log
    - *Commands -*
      (Start, Commit, Abort)

- **UNDO Logging**
    - Uncommitted transactions are undone.
    - U1: Write log with old value before new value written to disk. <T, X, v>
    - U2: Make sure values reflected on disk before commit.
    - Flush Log Command
    - Complication — 1
    - Recovery Rules:
        - Search for Ti commit or abort after start.

        - If not found, write old values logged **in reverse order.**
        - Add abort to log.
    - Undo idempotent if error during recovery.
    - Checkpointing
        - Why Checkpoint
        - Algo:
            - Stop accepting new transactions, wait for current ones to commit.
            - Flush log to disk.
            - Write CKPT.
            - Resume accepting transactions.
    - Non-quiescent Checkpointing
        - To avoid shutting the system when checkpoint is made.
        - Write log START CKPT(*transactions*).
        - Wait for said transactions to finish/abort, don't stop other transactions.
        - Write END CKPT and flush log.
        - Scan for END CKPT, then backtrack normally.

- **REDO Logging**
    - Problem with UNDO Logging - Can't commit without first writing all changes to disk.
    - Can save on disk I/Os if data kept as long as possible in main memory with ability to fix things in case of crash.
    - Differences with UNDO - 3
    - Logging Rule:
        - T,X,v stored, v is new value.
        - All previous change logs to X should be committed and should appear on disk.
        - Order:
            - Log shows changed database elements.
            - Commit recorded
            - Database elements changed.
        - Recovery:
            - Scan log forward from beginning.
            - If T is committed, change value to v, else do nothing.
            - For incomplete transaction T, write ABORT record to log.
            - Flush Log.

    - Checkpoint:
        - Write START CKPT <Ts> for active transactions.
        - Write to disk all database elements that were written to buffer by already committed transactions before START CKPT.
        - Write END CKPT
        - Flush Log.
    - Recovery with Checkpoint:
        - When at END CKPT, every transaction before START CKPT has changes written to disk.
        - Without END, redo START transactions.

- **UNDO/REDO Logging**
    - Undo requires more I/O.
    - Redo requires more buffers.
    - UNDO/REDO is flexible.
    - <T,X,v,w> v -> w
    - U[dated record T,X,v,w should appear on disk before modifying database element X on disk.
    - Commit T can precede or follow.
    - Recovery:
        - Redo committed as earliest first.
        - Undo uncommitted with latest first.
    - Checkpoint:
        - START CKPT (*trans*)
        - Write changes to Disk
        - END CKPT

- Recovering from Disk failure:
    - Log maintained on separate Disk
    - Log preferably of Redo or Undo/Redo type.
    - **ARCHIVE** true complete solution - making a copy of database periodically.
    - Recovery = Backup + Log
    - Full Dump vs Incremental Dump
    - Non-quiescent Archiving:
        - Without shutdown.
        - START DUMP
        - Perform checkpoint
        - Perform full or incremental data dump
        - Log secured and stored

        - Log record END DUMP
    - Restoring from Archive:
        - Reconstruct from Full Dump.
        - Modification, earliest first, for incremental dump.
        - Modification using surviving log.

- **Chapter 9**

  - Scheduler
    - Transactions pass requests to Scheduler.
  - Problems with inconsistency:
    - Dirty Read
    - Non-Repeatable Read
    - Phantom Problem
  - Isolation Levels:
    - READ UNCOMMITTED - *Dirty, Non-Repeatable, Phantom*
    - READ COMMITTED - *Non-Repeatable, Phantom*
    - REPEATABLE READ - *Phantom*
    - SERIALISED
  - Serial and Serialisable Schedulers
    - Correctness Principle
    - Schedule - def
      - Read Write ops in buffer counted.
      - INPUT OUTPUT ignored.
    - Serial Schedule.
    - Serialisable Schedule.
  - Notations
  - **Conflict Serialisability** - 4 conditions
  - Conditions for Swap eligibility - 2
  - **Conflict Equivalent Schedules**
  - Conflict Serialisable (= *Serialisable Schedule*) - def
  - *Conflict Serialisability is not required to be serialisable.*
  - Precedence Graph
  - Single Lock based Scheduler
    - Transaction Consistency - 2
    - Schedule Legality - 1
    - Rule 1 : Request Release
    - Rule 2 : Schedule Legality
  - Two Phase Locking
    - All lock requests precede unlock requests.
    - Deadlock problem

- Locking as conflict serialisable schedule
- Shared Lock - sli
- Exclusive Lock - xli
- Unlock - uli
- Requirements - 3
- Upgrading Locks - Deadlock
- Update Lock
  - Locks allowed Table
- Increment Locks
  - Locks allowed Table

- Architecture of a Lock Scheduler
  - Don’t trust transactions, insert locks.
  - Hold locks till commit/abort.
  - 2 part scheduler
    - 1 : Select appropriate lock
    - 2 : execute operation
- Lock Table
  - Hash table usage
  - Tran Mode Wait Next
- Selection
  - FCFS
  - Priority to shared
  - Priority to Upgrade
- Hierarchy
  - Lockable elements — relation, tuple, disk block
  - B-Tree of Data
- Warning and Ordinary Locks
- Warning Protocol
  - IS and IX Locks
  - Compatibility Matrix
- Locking relations to deal with phantoms
- Tree Protocol - B-Tree
  - Monkey Bar Logic
  - **Tree Rules**
- Optimistic Protocols
  - Timestamps
  - Validation
  - *Assumption that conflicts are rare.*
- Locking protocols are pessimistic protocols.
- Structure:

- Read Timestamp
    - Write Timestamp
    - Commit Bit
  - READ - Algorithm for Timestamp based scheduling
  - WRITE - Algorithm for Timestamp based scheduling
  - Multiversion timestamp
  - Multiversion timestamp Protocol
  - Timestamp vs Locking
  - Validation Protocol
    - Read, Validate and Write phases
    - Sets Maintained:
      - START
      - VAL
      - FIN
    - Validation Rules
  - Locks vs Timestamps vs Validation

- **Chapter 10**

  - Dirty Data Problem
  - Isolation levels
  - Cascading Rollback
    - Timestamp and Validation schedulers avoid it.
  - Strict Locking
  - Recoverable Schedules
  - If blocks, if tuples.
  - Group Commit
  - Logical Logging
  - View Serialisability
  - View Equivalence - 3

  - Resolving Deadlock
    - DETECTION
      - Wait-for graph, rollback victim on cycle.
    - PREVENTION
      - Time based wait time cutoff, rollback
      - Resource Ordering - locking only in a particular order
      - Wait-Die - timestamp comparison
        - Starvation - resubmit with original timestamp
      - Would-Wait - timestamp comparison
        - Starvation - resubmit with old timestamp

- Distributed Database Systems
    - Advantages - 5
    - Disadvantages - 4
    - Parallelism
        - Pipeline
        - Concurrent Operation
    - Distributed Transactions
    - Data Replication
    - Distributed Query Optimisation

- Distributed Commit
    - Why necessary - 2
    - 2 Phase Commit
        - Phase 1 : Coordinator, Site — prepare and ready
        - Phase 2 : Commit T or abort T, based on communication.
    - Recovery of Distributed Transactions
    - Recovery In case of Site Fails - 3
    - Coordinator Fail
    - Leader election
    - DBA intervention

- Distributed Locking
    - Problems with central locking
    - Messages - 3 *req, grant, release*
    - Primary Copy Locking
    - Global Locks

- Long Duration Systems
    - Examples
    - Sagas Model
        - Rules, Structure
    - Compensation Transactions

- **No SQL**

    - Issues faced with SQL - 3
    - Steps taken - 6
    - Requirements from No SQL
    - Cloud

- Parallel Databases
- Sharding
- Parallel Key-Value Data Stores
- No-SQL relax one or more of ACID Properties (mostly C,D)
- Data Representation:
  - Key/Value
  - JSON
  - BSON
  - Big Table
- API:
  - Get - key
  - Put - key, value
  - Delete - key
  - Execute - key, operation, parameter
- Classifications:
  - Uninterrupted key/value - S3
  - Flexible Schema
    - MongoDB
    - CoudDB

- CAP Theorem - at most 2 of following properties in any system
  - Consistency
  - Availability
  - Partitions
- Eventual Consistency
- BASE - (*Basically Available, Soft state, Eventual Consistency*)
- Advantages of NoSQL:
  - Cheap
  - Partition-able
  - Distributable
  - No Schema required
- When use NoSQL

- **CAP Theorem**

  - The theorem.
  - 2 node proof
  - Why important
  - CA, CP, PA
  - Partitioning tolerance important

- AP : Best effort consistency
  - Caching
  - DNS
- CP : Best Effort Availability
  - Protocols
  - Distributed Locking
- Consistency:
  - Strong
  - Weak
  - Eventual
    - Casual
    - Read-your-Write
    - Session
    - Monotonic Read
    - Monotonic Write
    - In Practice
- Dynamic A/C Trade off
- Heterogeneity
- Partitioning:
  - Data
  - Operational
  - Functional
  - User
  - Hierarchical
- PACELC
  - Types of systems

- **Additional Content**

  - Parallel DBMS
    - Pipeline
    - Partition
  - Why DBMS is successful with Parallelism
  - Parallelism
    - Intra Operator
    - Inter Operator
    - Inter Query
  - Distributed Data Independence
  - Distributed Transaction Atomicity
  - Distributed Databases
    - Homogenous

    - Heterogenous
- Fragmentation
- Replication
- Datalog
- OLAP, OLTP
- Data Warehousing
    - Issues - 5
- Data Mining
- Multidimensional Data Storage
    - MOLAP
    - ROLAP
- Drill Down
- Pivoting
- XML
- Semistructured Data
- Spatial Data
- R-Tree