Hano Strydom

hanostrydom8@gmail.com

Abstract

These notes summarize the work from the Slides, videos, and Textbook.  
Textbook: NoSQL – Database for Storage and Retrieval of Data in Cloud

ITRI 623 Notes

Databases

Contents

[Table of Figure 2](#_Toc111302595)

[List of Tables 2](#_Toc111302596)

[Acronyms 3](#_Toc111302597)

[Glossary 4](#_Toc111302598)

[1. Distributed Transaction Processing 6](#_Toc111302599)

[1.1. Introduction to Distributed Database 6](#_Toc111302600)

[1.1.1. Distributed Processing and Distributed Database 7](#_Toc111302601)

[1.1.2. Parallel DBMS and DDBMS 8](#_Toc111302602)

[1.1.3. Distributed Database Techniques 10](#_Toc111302603)

[1.1.4. Concurrency Control in Distributed Database 12](#_Toc111302604)

[1.1.5. Promises of DDBMS 16](#_Toc111302605)

[1.2. Introduction to Distributed Transaction Processing 17](#_Toc111302606)

[1.2.1. Background of Distributed Transaction Processing 17](#_Toc111302607)

[1.2.2. Introduction to Distributed Transaction Processing Models 19](#_Toc111302608)

[1.2.2.1. Atomic Actions and Flat Transactions 19](#_Toc111302609)

[1.2.2.2. Nested Transactions 20](#_Toc111302610)

[1.2.3. Distributed Transaction Processing in Relational and Non-Relational Databases 21](#_Toc111302611)

[1.2.3.1. Distributed Transaction Processing in Relational Database 22](#_Toc111302612)

[1.2.3.2. Distributed Transaction Processing in Non-Relational Database 23](#_Toc111302613)

[1.3. Return of ACID Property in Distributed Transaction Processing 23](#_Toc111302614)

[1.3.1. Introduction to ACID Property 24](#_Toc111302615)

[1.3.2. ACID Property and Non-Relational Database 25](#_Toc111302616)

[1.4. NoSQL in Distributed Transaction Processing 26](#_Toc111302617)

[1.5. Security Issues in Distributed Transaction Processing Systems 29](#_Toc111302618)

# Table of Figure

[Figure 1 - Components of Distributed Database 7](#_Toc111302619)

[Figure 2 - DBMS vs DDBMS 9](#_Toc111302620)

[Figure 3 - Different types of architecture: (a) - Shared Memory; (b) - Shared Disk; (c) - Shared Nothing 9](#_Toc111302621)

[Figure 4 - Different Types of Fragmentation 11](#_Toc111302622)

[Figure 5 - Absence of concurrency control scenario 12](#_Toc111302623)

[Figure 6 - Centralized 2PL communication 13](#_Toc111302624)

[Figure 7 - Communication in distributed 2PL in distributed environment 14](#_Toc111302625)

[Figure 8 - Wound-wait Algorithm 14](#_Toc111302626)

[Figure 9 - Transactions in distributed database system 18](#_Toc111302627)

[Figure 10 - Flat Transaction Structure 20](#_Toc111302628)

[Figure 11 - Nested Transaction Structure 21](#_Toc111302629)

[Figure 12 - Significance of ACID Property 24](#_Toc111302630)

[Figure 13 - Significance of CAP theorem 26](#_Toc111302631)

[Figure 14 - Column family store database 26](file:///C:\Users\hanos\Desktop\University\Honneurs\ITRI623_DB\Opsommings\ITRI623%20Notes.docx#_Toc111302632)

[Figure 15 - Key- Value pair database 27](#_Toc111302633)

[Figure 16 - Graph Database 28](#_Toc111302634)

[Figure 17 - Comparison of NoSQL and SQL 28](#_Toc111302635)

[Figure 18 - Difference in Queries between NoSQL and SQL 29](#_Toc111302636)

# List of Tables

[Table 1 - Acronyms 3](#_Toc111302637)

[Table 2 - Glossary 4](#_Toc111302638)

# Acronyms

Table 1 - Acronyms

|  |  |
| --- | --- |
| DB | Database |
| DBMS | Database Management System |
| DDBMS | Distributed Database Management System |
| CPU | Central Processing Unit |
| ACID | Atomicity, Consistency, Isolation and Durability) |
| 2PL | Two Phase Locking |
| LM | Lock Manager |
| TM | Transaction Manager |
| ROWA | Read one, write all |
| WW | Wound-Wait |
| BTO | Basic Timestamp ordering |
| OPT | Distributed Optimistic |
| TM | Transaction Manager |
| DM | Data Manager |
| CSS | Network concurrency control scheduler |
| TC | Transactional client |
| ACP | Atomic commit protocol |
| DTP | Distributed transaction processing |
| RDBMS | Relational Database Management System |
| CAP | Consistency, Partition tolerance and Availability |

# Glossary

Table 2 - Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Distributed database | Consists of a set of interrelated databases stored on several computers distributed over a network wherein the data can be concurrently accessed and altered |
| DB server | Software that administers the database |
| Client | Application that requests information and seek services from the server |
| Distributed databases | Data are stored across computer systems. |
| Distributed Processing | The use of more than one computer (or CPU) to run an application for an individual task. |
| Parallel processing | A subset of distributed computing where a single computer uses more than one CPU to execute programs (CPUs share information with each other). |
| Distributed system | Network of independent computers that interact with each other in order to achieve a goal and do not share memory of processors. Communicate via messages over a network. |
| Parallel database management | Management of data in tightly coupled multi-processor computer done by a DBMS data are partitioned across multiple nodes |
| DDBMS | Software system that authorizes the management of the distributed DB and makes the distribution transparent to the users. |
| Fragmentation | Technique of dividing relations of a DB into a number of pieces, called fragments, which are then distributed and can be stored in various computers located at different sites. |
| Allocation | Each fragment is stored at the site with optimal distribution |
| Replication | A copy/replica of a fragment is maintained at several different sites |
| Distributed transaction | DB transaction in which two or more network hosts are involved. The transaction is composed of several sub transactions, each running on a different site. |
| Transaction Manager (TM) | Manages distributed computation for that transaction. |
| Atomic action | Unitary action or object that is essentially indivisible, unchangeable, whole and irreducible. |
| Nested transaction | Transaction that is initiated by an instruction within the scope of an already started transaction. |
| Atomic action | Unitary action or object that is essentially indivisible, unchangeable, whole and irreducible |
| Non-relational database | Database that does not incorporate the table/key model as that of RDBMS promote |

# Distributed Transaction Processing

## Introduction to Distributed Database

* Expansion of huge quantity of real time data, the dimensions of data are escalating exponentially.
* Hard to find centralized repository that can efficiently store the data, which needs to be:
  + Retrieved
  + Manipulated
  + Updated using some sort of management system
* DBMS are managing DB activity as Transactions
  + This guarantees consistency when multiple users perform concurrent operations on them
    - Data independence (transparency)
  + The development of database management system (DBMS) helps to fully achieve data independence (transparency) providing centralized controlled data maintenance and access
* The size of data is too large to do this centralised and use computing power optimally.
* The necessity to balance the computer’s workload to *avoid peak-load problems* when people across an organization want to use it, *limits users flexibility* when *using a Centralized computing systems*.
* A distributed database consists of a set of interrelated databases stored on several computers distributed over a network wherein the data can be concurrently accessed and altered.
* Components of distributed database:
  + Database Server
  + Client(s)
* DB server is the software that administers the database, and a client is application that requests information and seek services from the server.
* Each computer is a node (client, server, or both)
* Each server is managed by Its local DBMS an cooperates to preserve the consistency of the global DB.
* A DDBMS makes distribution transparent to users. Logical DB partitioned into sections stored on one or more computers

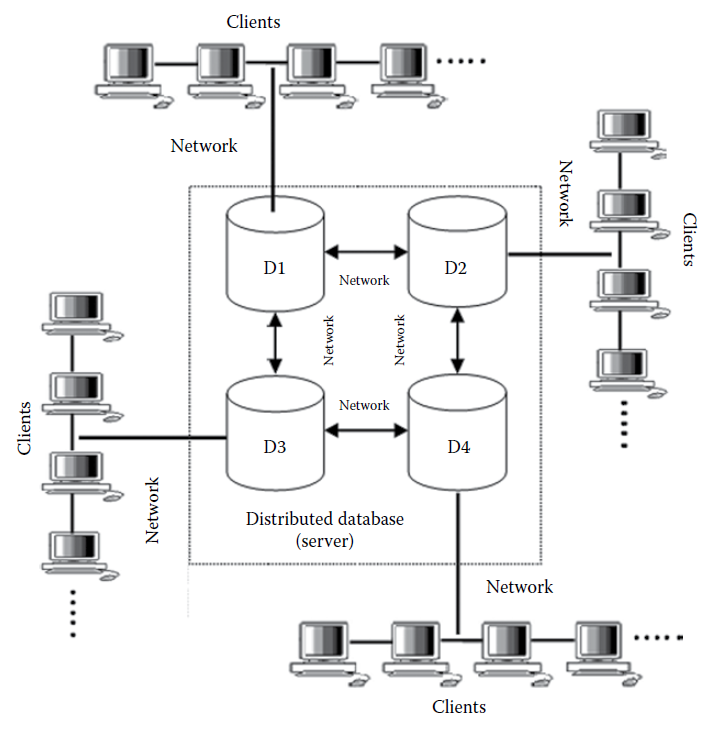


Figure 1 - Components of Distributed Database

* In Figure 1 D1, D2, D3 and D4 can have its own DBMS like oracle or Mongo DB.
* Each computer is a node

### Distributed Processing and Distributed Database

* Hadoop
  + A system software that does 2 things:
    - Handles distributed storage of data (Its transparent)
    - Handles the distributed processing of data
* Distributed databases
  + Data are stored across computer systems.
  + DB system keeps track of the location of the data, so the distributed character of the database is not evident to users
* Distributed Processing:
  + The use of more than one computer (or CPU) to run an application for an individual task.
  + Use a LAN and identify idle CPUs and parcels out programs to make use of them
  + Distributed processing is composed of distributed databases
  + Main Goal of Distributed Processing:
    - Connect users and resources in a transparent, open, and scalable way
  + **Benefits of Distributed Processing:**
    - Resource Sharing
      * Share Data
    - Scalability
      * Can add computers to DDB network
      * Computers will share the work automatically
    - Fault Tolerance / Robustness
      * If one goes down the other take over
    - Performance / Speed
      * More computer to do the job quicker

### Parallel DBMS and DDBMS

* Computers are unable to keep up with the scale of data becoming available
  + Multiple Processors
    - This solution outwits the physical and mechanical limitations on speeds of individual processor.
    - One processor is performing a particular aspect of some computation, other computation can be performed by other available processors and the work will advance in parallel.
* **Parallel processing**
  + A subset of distributed computing where a single computer uses more than one CPU to execute programs (CPUs share information with each other).
* **Distributed system**
  + Network of independent computers that interact with each other in order to achieve a goal and do not share memory of processors.
  + Communicate via messages over a network.
    - Messages can be used to execute programs, packets of data and propel signals for behaviour
* Parallel database management
  + Management of data in tightly coupled multi-processor computer done by a DBMS data are partitioned across multiple nodes.
* DDBMS
  + Software system that authorizes the management of the distributed DB and makes the distribution transparent to the users.

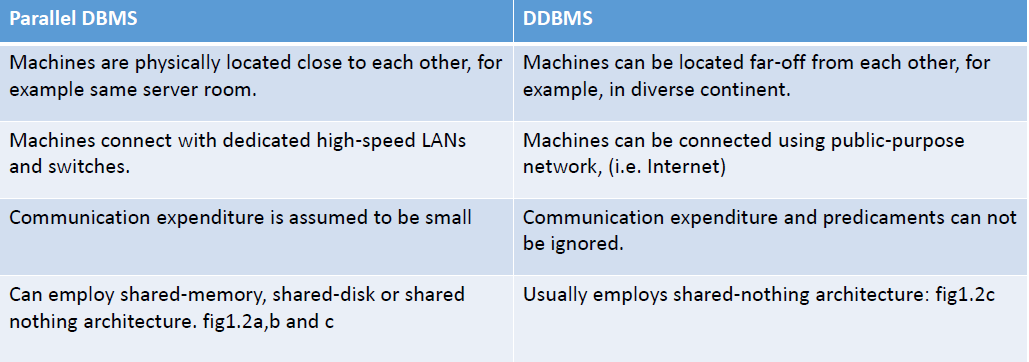


Figure 2 - DBMS vs DDBMS

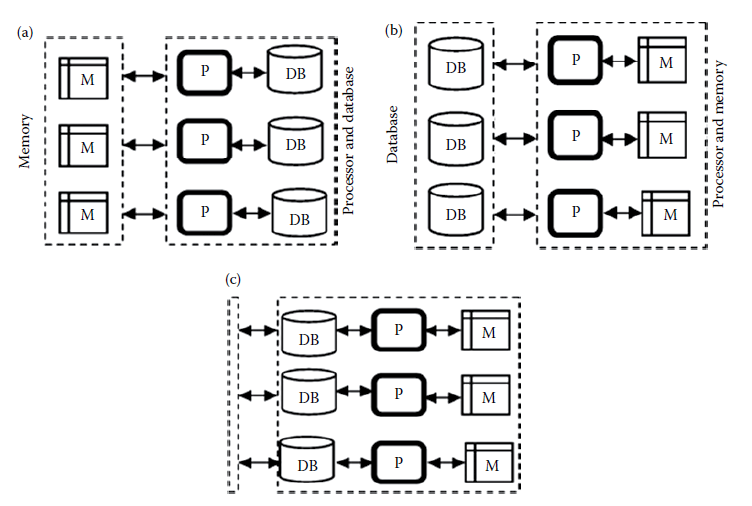


Figure 3 - Different types of architecture: (a) - Shared Memory; (b) - Shared Disk; (c) - Shared Nothing

* Benefits of DDBMS:
  + Placement of data on different sites is not known to the user
  + User does not have to concern about the operational details of the processing
    - Can issue command from any location without affecting the working of the network
  + Replication transparency
    - Multiple copies of data at multiple sites
  + Increases reliability and availability (if one node fails, another copy of data is used)
  + Improvement in performance
    - Keep data close to where it is needed most.
  + Scalability
    - Expand by adding new nodes without making changes to the underlying configuration.

### Distributed Database Techniques

* Distributed Processing on DBMS
  + Is a proficient approach of progressing performance of applications that manipulate outsized dimensions of data.
* Distributed Processing has 2 main objectives:

1. Minimize irrelevant and redundant data accessed during execution
2. Reduce data switching among nodes located at different locations

* Key is to understand how to fragment data and allocation and replication of fragments in different nodes of the distributed system.
* **Fragmentation:** 
  + Technique of dividing relations of a DB into a number of pieces, called fragments, which are then distributed and can be stored in various computers located at different sites.
  + It aims to improve reliability, performance, storage capacity, communication, and security.
* **Rules for Fragmentation:**

1. If a database D, is decomposed into fragments, D1, D2, …, Dn, each data item that can be found in D must in at least one fragment.
   1. No loss of data; consistency of distribution
2. It must be possible to define a relational operation that will reconstruct the database D from fragments, D1, D2, …, Dn.
   1. Perseverance of functional dependencies
3. If a data item Xi appears in fragment Di, then it should not appear in any other fragment.

* Difference between **Fragmentation** and **Replication**
  + **Fragmentation**
    - Part of DB
    - Each fragment can be replicated
  + **Replication**
    - Copy of the fragment:
* **Types of Fragmentation:** 
  + Horizontal (a)
    - Different rows are stored on different computers, depending on a condition
  + Vertical (b)
    - Some of the columns are store in one computer, and the rest are stored in other computers
    - No selection condition is used
  + Mixed / Hybrid (c)
    - Combination of Vertical and Horizontal
* The company’s needs will determine whether you use column based, row based or mixed

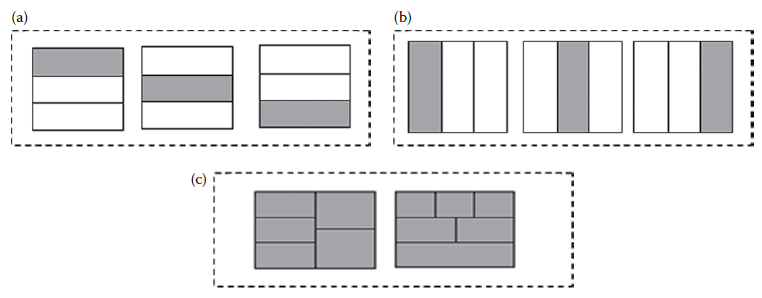
****

Figure 4 - Different Types of Fragmentation

* Allocation
  + Each fragment is stored at the site with optimal distribution
* Replication
  + A copy/replica of a fragment is maintained at several different sites
* Strategies for replacement of data
  + Centralized
    - Single DB stored at one site with use distributed over the network
  + Fragmented
    - Partitions of the DB into disjoint fragments, with each fragment assigned to one site
  + Complete Replication (combination of fragmentation, replication, and centralized)
    - Maintaining a complete copy of DB at each site and is a combination of fragmentation, replication and centralized.
  + Selective Replication (combination of fragmentation, replication, and centralized)
    - Some data item are fragmented to achieve high locality of reference, and others that are used at many sites and are not frequently updated are replicated and others are centralized.

### Concurrency Control in Distributed Database

* Aim:
  + The DBMSs control the concurrent execution of user transactions so that the overall correction and update of the database are maintained allowing the user to access the database in a multiprogramming approach safeguarding the misapprehension that each user is working single-handedly on a dedicated system.

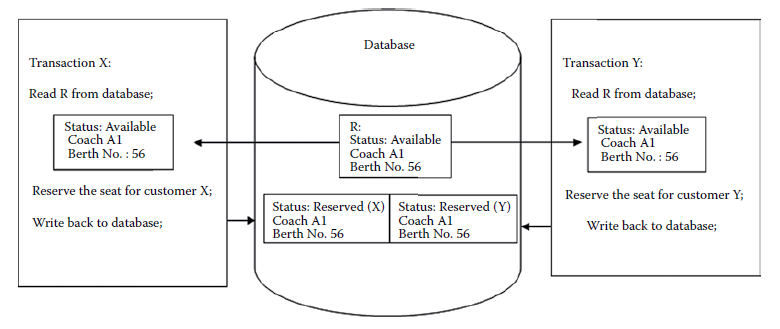


Figure 5 - Absence of concurrency control scenario

* Centralised DBMSs implement Two Phase locking (2PL)
  + Too much emphasis on communication in distributed system
* ACID ( atomicity, consistency, isolation, and durability) property is maintained in distributed transaction processing using:
  + Recoverable processes
  + Commit protocol

**Algorithms for concurrency control in Distributed systems:**

1. Extensions to 2PL to DDBMS:
   1. Centralized 2PL
      1. Single site responsible for lock management (LM) for the whole DDBMS.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Easy to implement | Bottlenecks and lower reliability |
|  | Replica control protocol is additionally needed if data are replicated. |

* + 1. Coordinating Transaction Manager (TM at site where transaction is initiated) can make all locking requests on behalf of local TMs.

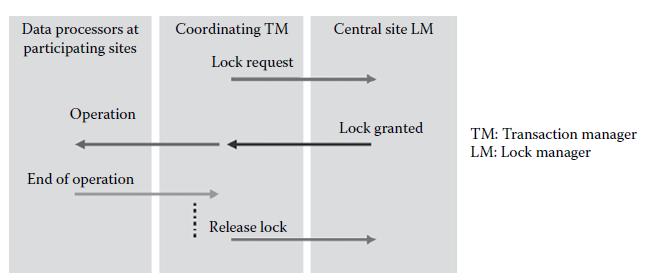


Figure 6 - Centralized 2PL communication

* 1. Primary Copy 2PL:
     1. Several LMs are distributed to a number of sites and each LM is responsible for managing the locks for a set of data items.
     2. For replica data items, one copy is chosen as the primary copy, and others are slave copies.
     3. Only the primary copy of a data item that is updated needs to be write-locked.
     4. Once the primary copy is updated the change is propagated to the slaves

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Lower communication costs are better performance than centralised 2PL | Deadlock handling is more complex in this algorithm |

* 1. Distributed 2PL:
     1. 1c. Distributed 2PL: LMs are distributed to all sites.
     2. Each LM is responsible for locks for data at that site.
     3. Same as primary 2PL if data is not replicated.
     4. ROWA (read one, write all) is implemented if data are replicated.
     5. Read (x): Any copy of a replicated item x can be read by obtaining a read-lock on that copy
     6. Write(x): All copies of x must be write-locked before x can be updated
     7. Communication: The coordinating TM sends the lock request to the LMs of all participating sites, and the LMs pass the operations to the data processors. End of operation is signalled to coordinating TM.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Better than primary copy 2PL | More complex and more communication |

A picture containing timeline

Description automatically generated

Figure 7 - Communication in distributed 2PL in distributed environment

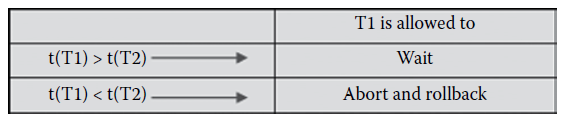
1. Wound – Wait
   1. Also “Read-any, Write all” strategy. 2PL waits for data, WW avoids deadlocks with timestamps.
   2. Each transaction is numbered according to its initial start-up time, and younger transactions are prevented from making older ones wait.
   3. If an older transaction requests a lock, and if the request would direct to the older transaction waiting for a younger transaction, the younger transaction is “wounded” and it is restarted unless it is already in the second phase of its commit protocol. Younger transactions can wait for older transactions.
   4. Advantage: No Deadlocks

Figure 8 - Wound-wait Algorithm

1. Basic Timestamp Ordering (BTO)
   1. Use timestamps differently from method of WW.
   2. Rather than using a locking approach, BTO correlates timestamps with all currently accessed data items and requires that conflicting data accesses by transactions be performed in timestamp order. Transactions that attempt to perform out-of-order accesses are restarted.
   3. When a read request is received for an item, it is permitted if the timestamp of the requester exceeds the item’s write timestamp.
   4. When a write request is received, it is permitted if the timestamp of the requester exceeds the item’s read timestamp.
   5. If the timestamp is less than the current timestamp. The request is ignored.
   6. For replicated data: read any, write all is implemented
2. Distributed Optimistic
   1. Timestamped based operating by exchanging certification information during the commit protocol
   2. For each data item a read and write timestamp is maintained. Read and writes are done freely and stored in a local workspace until commit time.
   3. When all the transaction’s cohorts have completed their work, and have reported back to master, the transaction is assigned a globally unique timestamp. This timestamp is sent to each cohort in the “prepare to commit” message, and is used locally to certified read and writes:
   4. A read is certified if:
      1. (a) the version that was read is still the current version of the item and
      2. (b) no write with a newer timestamp has already been certified.
   5. A write is certified if:
      1. (a) No later reads have been certified and subsequently committed, and
      2. (b) no later reads have been locally certified already

### Promises of DDBMS

* DDBMS
  + Makes distribution transparent to user
  + Facilitates quick and easy access of data for users who are located at different locations.
  + Every site is considered as a database system of its own and supervised autonomously
  + Local data are stored on a local computer DB
  + Different sites are connected to each other using high speed network connections.
* **Promises of DDBMS:**

1. ***Transparent management if distributed , fragmented, and replicated data.***
   1. Transparency refers to separation of higher-level semantics of a system from lower-level implementation details, from data independence in centralized DBMS to fragmentation transparency in DDBMS
2. ***Improved reliability and availability through distributed transactions.***
   1. Increases reliability and availability since DDBMS has multiple nodes, and if one fails, other nodes are available to perform the desired task.
3. ***Improved performance.***
   1. Since each site handles only a portion of a database, the contention for CPU and I/O resources is not that severe. Data localization reduces communication overheads.
4. ***Higher system extensibility.***
   1. Ability to add new sites, data, and users over time without major restructuring.
5. ***Sustained performance***
   1. Single-site failure does not affect performance of system

## Introduction to Distributed Transaction Processing

* Distributed transaction
  + DB transaction in which two or more network hosts are involved. The transaction is composed of several sub transactions, each running on a different site.
* Host provide transactional resources, while the TM is responsible for creating and managing a global transaction that encompasses all operations against such resources

### Background of Distributed Transaction Processing

* DDBMS 5 Components:
  + Transactions
  + Transaction Manager
  + Data Manager (DM)
  + Network concurrency control scheduler (css)
  + Data
* Each transaction is supervised by a single TM, which manages distributed computation for that transaction.
* Operations for Transaction:
  + READ(X)
    - Returns the value of X in the current logical database state
  + WRITE(X, new value)
    - Creates a new logical database state in which X has the specified new value,
  + BEGIN
    - Bracket transaction executions
  + END
    - Bracket transaction executions
* DMs manages the stored DB functioning as back-end DB processors.
* In response to commands from transactions, TMs issue commands to DMs specifying stored data items to be read or written.
* Components in a system can take the role of a **client, server, or coordinator**
  + Transactional client (TC)
    - Only sees transaction through the transaction coordinator
    - TC invoke services: begin, commit, abort.
  + Transactional Server
    - Registers its participation in a transaction with the coordinator and implements 2 phase transactional protocol (2 phase commit).
  + *Transactional Coordinator*
    - Manages transactions;
    - Handles begin/commit/send calls;
    - Allocated system-wide unique ID;
    - Different transactions has different coordinators.

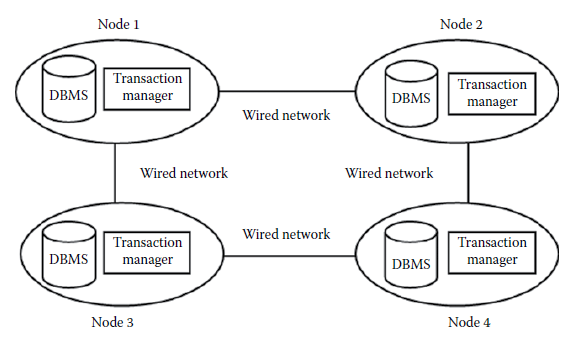


Figure 9 - Transactions in distributed database system

* DM may abort transactions, but an atomic commit protocol (ACP) is run by each DM to ensure that all the sub-transactions are consistently committed or aborted.
* Atomic commit protocol (ACP) ensures:
  + All the DMs reach same decision
  + Decisions are not reversible
  + A commit decision can only be reach if all the DMs voted to commit
  + If there are no failures and all the DMs voted to commit, the discussion will be commit
  + If all failures are repaired, without any new failures, all the DMs will eventually reach a decision

### Introduction to Distributed Transaction Processing Models

1.2.2 is not handled in the slides nor covered in the videos

* Transaction processing models
  + Mechanisms for imposing some structure on transactions
  + Meant for transaction that updates data on two or more computer systems connected via network that must update distrib­uted data avoiding any kind of failure.
* Failures and Problem from designing and developing a transaction model:
  + Failure at client side
  + Server-side failure
  + Connection failure
  + Site failure
  + Network problems
  + Data duplication
  + Distributed deadlocks
  + Distributed transaction at multiple sites
* When computers communicate with each other, a TM propels, sets up, commits, and aborts messages to its entire subordinate TMs. It does so as follows:
  + Client begins the transaction by making a request to the TM and deploys the request
  + TM on receiving the request from the client processes it by saving the updates
  + TM after processing the client request maintains the sequential transaction log, thereby ensuring the durability related to the decision made by it in regard to commit or abort

#### 1.2.2.1. Atomic Actions and Flat Transactions

* Atomic action
  + Unitary action or object that is essentially indivisible, unchangeable, whole, and irreducible.
* Flat transaction consists of atomic actions performed on local variables by accessing single DBMS using call or statement level interface. Indivisible from client perspective.
* Total rollback when aborted, all variables restored to their previous value.
* Entire transaction fails if there is a problem.
* Issues related to flat transactions:
  + Limited access to a single DBMS;
  + Entire transaction takes place at a single point in time;
  + At a particular point, if there is some kind of failure, the entire updates/work is lost, that is, if one action fails, the whole transaction must be aborted

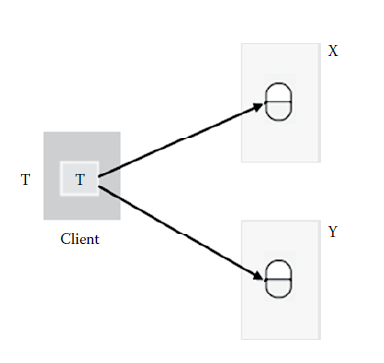


Figure 10 - Flat Transaction Structure

#### 1.2.2.2. Nested Transactions

* Nested transaction
  + Transaction that is initiated by an instruction within the scope of an already started transaction.
* One transaction has many sub-transactions, with their own sub-transactions….parents has children
* Root transaction is called top-level (TL) transaction;
* Nested transactions enable committing and aborting the sub-transactions independently of the larger transactions.
* **Nested Transaction Rules:**
  + - 1. When the child is active, the parent may not perform any operations other than to commit or abort, or to create more sub-transactions.
      2. The commit operation performed on the sub-transactions has no effect on the state of the parent transaction, parent transactions are still uncommitted. The parent can view the modification made by the child transaction, but these modifications will be hidden to all other transactions until the parent transaction also commits.
      3. If the parent transaction is committed or aborted, the same will happen to the child.
      4. The depth of the nested transaction is limited only by memory and cannot be restricted by external mechanisms.

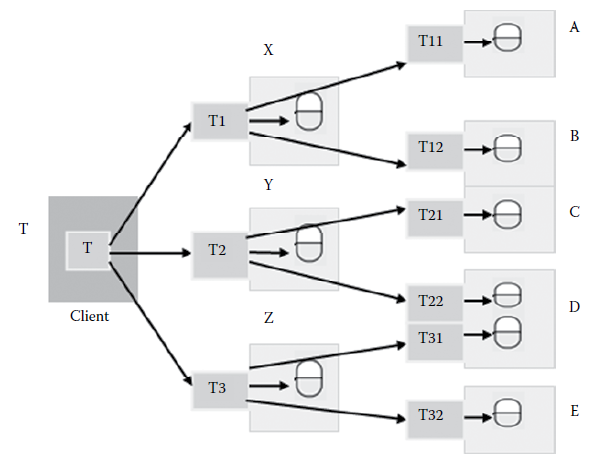


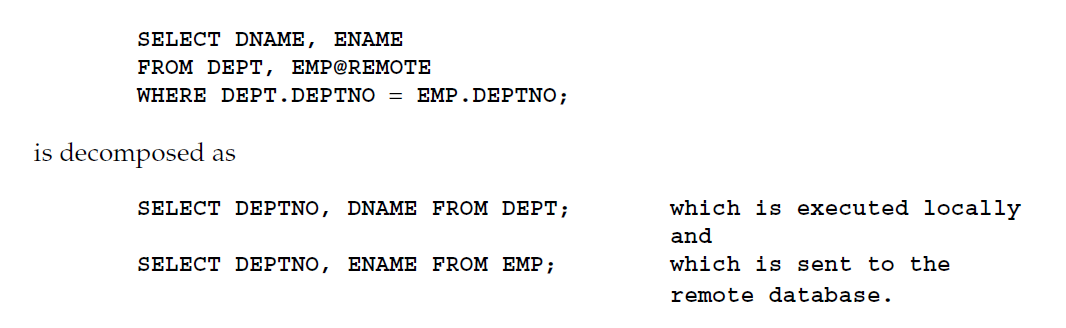
Figure 11 - Nested Transaction Structure

### 1.2.3. Distributed Transaction Processing in Relational and Non-Relational Databases

* Transaction processing basically deals with processing information or data that are divided into individual, indivisible rather atomic operations called transactions
* A transaction generally repre­sents any change in database with two main purposes:
  + Providing reliable units of work that allow correct recovery from failures and keeping a database consistent even in cases of system failure
  + Providing isolation between programs accessing a data­base concurrently
* Transactions compliant with ACID (atomic, consistent, isolated, and durable)

### 1.2.3.1. Distributed Transaction Processing in Relational Database

* Distributed transaction processing (DTP) takes place as follows:
  + The *application requester (AR)* 
    - Accepts requests in the form of SQL Query form application and sends it to appropriate application server.
  + The *application server (AS)* 
    - Processes the section that it can process and forward the remainder to DB servers for subsequent processing, using Application Support Protocol (ASP) which handled data representation conversion.
  + The *database server (DS)* 
    - Supports distributed requests and forward parts of the request to collaborating DSs, using Database Support Protocol (DSP)
* *Decomposition of SQL enables optimization: determine the number of tables to be sent through the network!*
* The following query accesses data from the local database “DEPT” as well as the remote “EMP” database as follows:



* The following PL/SQL program unit updates tables on the local database and the remote “sales” databaseText, letter

  Description automatically generated

#### 1.2.3.2. Distributed Transaction Processing in Non-Relational Database

* Non-relational database
  + Database that does not incorporate the table/key model as that of RDBMS promote
  + Transactions compliant with only 2 of 3 from CAP (Consistency, Partition tolerance and Availability)
* *Non-relational database uses aggregate data models* that enable usage of data struc­ture to solve several problem domains as modelled by developers
* Aggregate
  + Collection of data (unit) that makes it easier for the database to handle data storage over group when the unit of data resides on any machine
* Two Types of Distributing Data:
  + **Sharding**
    - Distributes data across multiple servers, so each server acts and the single source for a subset of data;
  + **Replication**
    - Copies data across multiple servers, so each bit of data can be found at multiple places. Done in two ways:
      * Master-slave replication
        + Make on node the authoritative copy, while slaves synchronise with master and may handle reads;
      * Peer-to-peer replication
        + Allows writes to any node; the nodes coordinate to synchronise their copies of the data.

## 1.3. Return of ACID Property in Distributed Transaction Processing

* Distributed transaction
  + Refers to the processing of data located at different loca­tions connected via a network, that is, database systems that focus on distributed transac­tions as transactions against multiple applications or hosts

### 1.3.1. Introduction to ACID Property

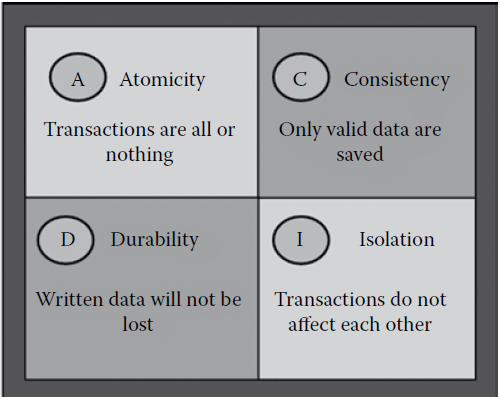


Figure 12 - Significance of ACID Property

* Transaction must follow these four properties:
  + *Atomicity* 
    - Refers to the ability of the DBMS to guarantee that either all of the jobs of a transaction are performed or none of them and database modifications must follow an “all or nothing” rule. If some part of a transaction fails, then the entire transaction fails, and vice versa.
  + *Consistency* 
    - Property ensures that the database remains in a consistent state, despite the transaction succeeding or failing and both before the start of the trans­action and after the transaction is over.
  + *Isolation* 
    - Refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction and helps to retain concurrency of database.
  + *Durability* 
    - States that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. That means when users are notified of success, the transactions will persist, not be undone, and survive from system failure.

### 1.3.2. ACID Property and Non-Relational Database

* As industries become more and more reliant upon a large quantity of unorganized data such as images, text, files, audio, and videos, traditional RDBMS technologies are proving to be a bottleneck in such kind of situations as they feature a very strict schema.
* NoSQL is a relatively new DBMS technology for handling large amounts of unstructured data that does not come in a predefined format.
* Most non-relational databases are incorporated into websites such as Google, Yahoo, Amazon, and Facebook
* The most important feature of a non-relational database is its scalability
* RDBMSs are compliant with the ACID, but NoSQL follows a different approach, that is, CAP theorem laid down by Eric Brewers.
  + Consistency
    - All clients view the same instance of data at the same time.
  + Availability
    - Database can be updated, added, or removed without going offline.
  + Partition tolerance
    - Databases can be distributed across multiple servers, and they are tolerant to network failures.
* This theorem states that it is mathematically impossible for an NoSQL DBMS to guarantee all three features. One must always pick two out of the three (C, A, P), that is, CA or CP or AP. The various databases that come under different categories are:
  + CA category
    - RDBMS (MySQL, Postgres
  + CP category
    - BigTable , HBase , HyperTable , MongoDB, Terrastore , Scalaris , and Redis .
  + AP category
    - Dynamo, Voldemort, CouchDB , SimpleDB , Riak , Tokyo Cabinet, and Cassandra

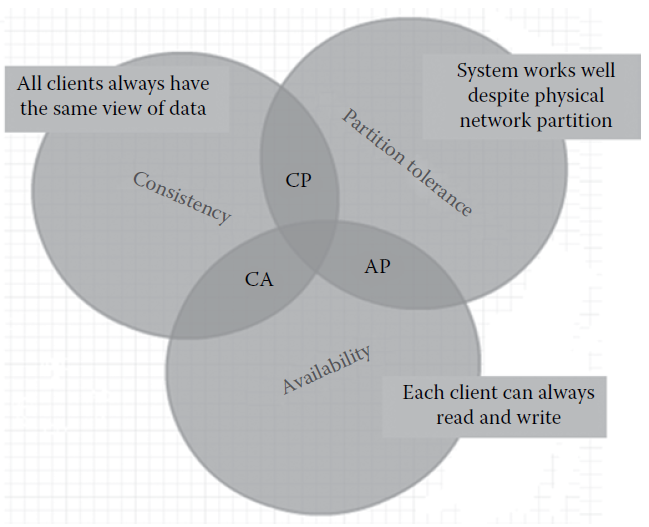


Figure 13 - Significance of CAP theorem

## 1.4. NoSQL in Distributed Transaction Processing

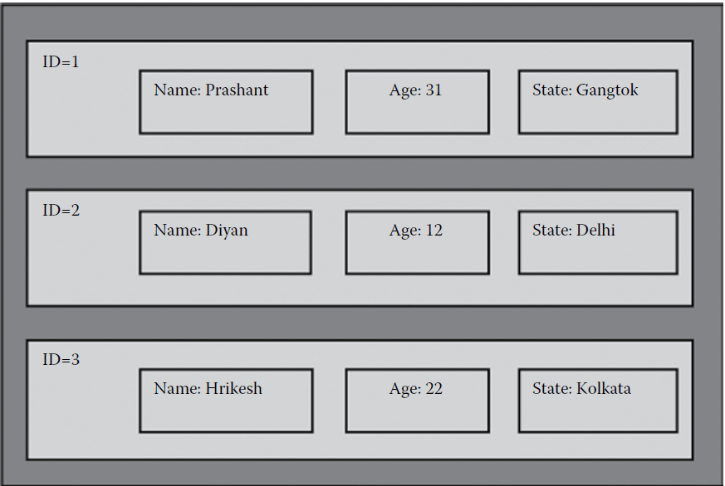
* NoSQL means Not Only SQL, implying that when designing a software solution or product, there is more than one storage mechanism that could be used based on the needs.
* NoSQL does not have a rigid definition, but NoSQL does not employ relational model, is open source, schema less, and meant for unstructured data storage and retrieval.
* There are four types of NoSQL databases:
  + Column family stores
  + Key value pairs
  + Document store
  + Graph databases
* Column family stores
  + Column family databases store data in column families as rows that have many columns associated with a row key.
    - Column families are groups of related data that are often accessed together. Cassandra is one of the popular column family databases; HBase, Hypertable, and Amazon DynamoDBare others

Figure 14 - Column family store database

* *Key–value pairs*
  + Key–value pair database is one of the simplest NoSQL data stores to use from an Application Programming Interface (API) perspective.
    - The client can get the value for the key, put a value for a key, or delete a key from the data store.
    - Some of the popular key–value databases are Riak, Redis, Berkeley DB, Amazon DynamoDB, and Couchbase.

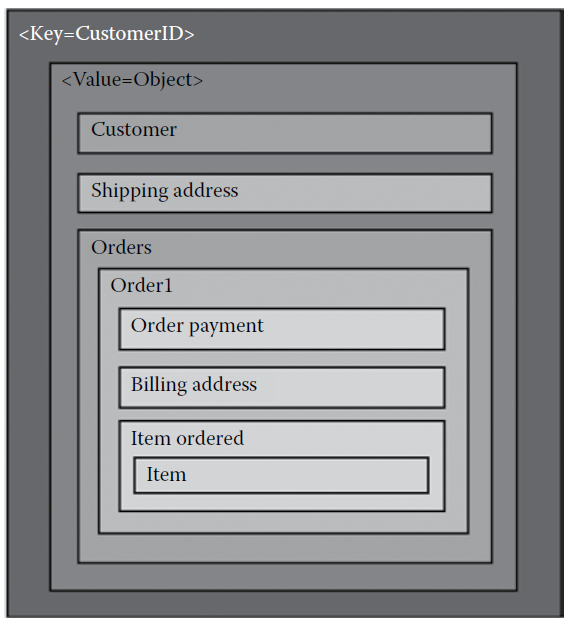


Figure 15 - Key- Value pair database

* Document store
  + Documents are the main concept in document databases.
    - It stores the data in JSON-like documents having key–value pairs.
    - MongoDB is a document database that stores data as a hierarchy of key–value pairs, which allows branching at different levels (a maximum of three levels).
    - Some of the popular document databases are MongoDB, CouchDB, Terrastore, Orient, RavenDB, and Lotus Notes that use document storage.
    - Document database such as MongoDB provide a rich query language and constructs such as database and indexes allowing for easier transition from relational databases.
    - Text, letter

      Description automatically generatedAlso, MongoDB stores data in JSON-like .BSON files.
* *Graph databases*
  + These databases store entities and relationships between these entities.
    - Entities are also known as nodes that have properties, and relations are known as edges that can have properties.
    - There are many graph databases such as Neo4J, Infinite Graph, OrientDB , or FlockDB

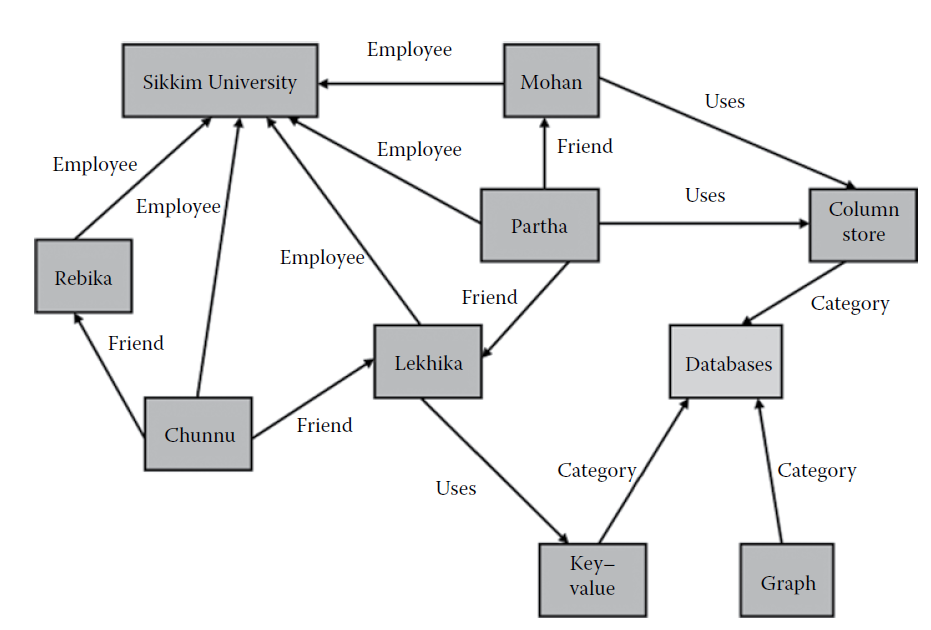


Figure 16 - Graph Database

* NOSQL Disadvantages
  + No standardized query as that of RDBMS;
  + Lacks ACID compliance, which is required in some situations;
  + No evident increase in performance when used for structured data

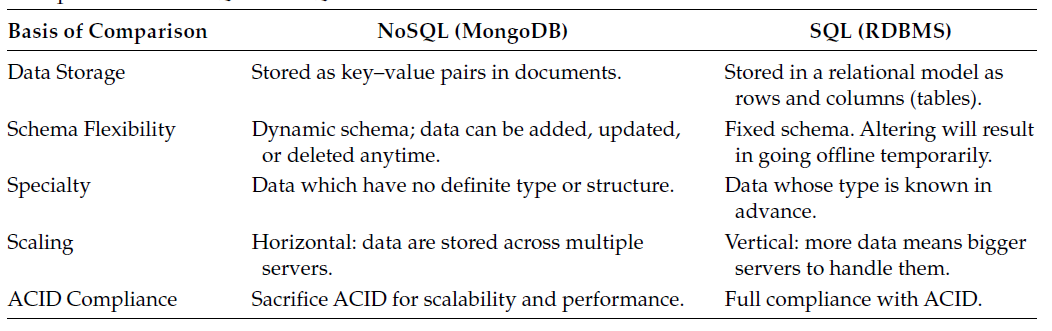


Figure 17 - Comparison of NoSQL and SQL

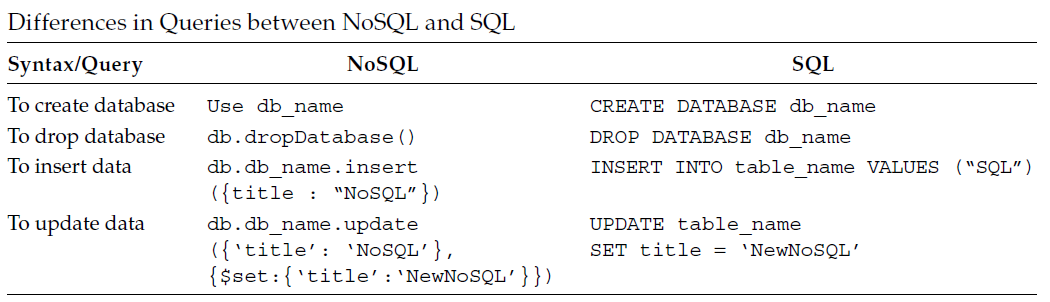


Figure 18 - Difference in Queries between NoSQL and SQL

## 1.5. Security Issues in Distributed Transaction Processing Systems

* A security constraint consists of a data specification (any subset of transaction processing system) and a security value (given by a classification function). The specific values are unclassified, confidential, secret, and top secret
* Two types of security constraints: internal and external constraints.
  + *Internal constraints*
    - Classify the entire Transaction Processing System (TPS) as well as relations, attributes, and tuples within a relation. These constraints can be applied to data as they are actually stored in the TPS.
  + *External constraints* 
    - Classify relationships between data and the results obtained by applying operations on the stored data, such as sum, average, and count. Among these constraints are the functional constraints and the dynamic constraints. These security constraints are subject to inconsistency and conflicting local security constraints. A good global security approach should reject inconsistent security constraints and inconsistent clearance of users.
* Examples of the inconsistencies:
  + Conflicting security constraints:
    - Such constraints classify the same facts into different categories
  + Overlapped security constraints:
    - These constraints cover overlapped data domains;
  + Inconsistent security level of replicated data:
    - Cases where different copies of replicated data may belong to different security cases
  + Access privileges of users to replicated data:
    - Instances where a user may have different access rights on replicated data at different sites.