International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Centralized vs. Decentralized Databases: A Technical Synthesis

Prepared by Himanshu, Harshkant, and Devanshi
Department of Computer Science & Engineering, School of Engineering and Technology, SGT University, Gurugram, Haryana, India
Supervisor: **Ms. Kanika Malik,** Assistant Professor

Abstract - A Database Management System (DBMS) can run on a single site under one administrative authority (centralized), or it can be spread across multiple sites with data partitioned, replicated, and sometimes controlled by independent parties (decentralized). This choice affects consistency, availability, latency, scalability, security, cost, and governance. This report is a literature-based synthesis. It defines the terminology, explains the mechanisms behind each approach (fragmentation, replication, distributed transactions, consensus, and consistency models), connects them to application settings, and examines the trade-offs. It also proposes a layered decision framework for selecting an architecture.

The report finds that neither approach is universally superior. Centralized systems offer simplicity, strong consistency, and unified control, but they are limited by a single point of failure and the capacity of one site. Decentralized systems offer scale, fault tolerance, and locality, but at the price of coordination overhead, weaker or costlier consistency, and a harder governance problem. Limitations and future directions are stated explicitly. No experimental measurements are reported.

Index Terms: Centralized Database, Decentralized Database, Distributed DBMS, Replication, Sharding, CAP Theorem, Consensus, Distributed Transactions, NoSQL, Access Control, Data Governance.

I. INTRODUCTION

A DBMS stores data persistently, enforces integrity constraints, coordinates concurrent access, and answers declarative queries [3]. The relational model [1] and the transaction concept [2] shaped early systems, which were largely centralized: one machine, one copy of the data, one administrator. As data volumes, user populations, and availability expectations grew, organizations began placing data on multiple machines and in multiple locations. This led to distributed database theory and practice [4], [5].

The vocabulary is often used loosely, so this report fixes three terms.

Centralized database: All data and DBMS processing reside at one site (one node, or one tightly coupled server), managed under a single authority.

Distributed database: Data is stored at multiple sites connected by a network but presented to users as one logical database [4].

Decentralized database: Control as well as data is spread out. Sites may be autonomous, peer-to-peer, or owned by different units or organizations, with no single node or party governing the whole system.

These terms overlap. Most decentralized systems are distributed, but a distributed system can still be centrally administered. This report uses "decentralized" for architectures where both storage and coordination or authority are spread out, and notes where the distinction matters.

The choice has practical consequences. A single-node database gives simple semantics, but one failure can stop service. A distributed database can survive failures and serve users near their location, but it must solve coordination problems that a single node never faces [4], [7]. Like earlier work in this series, this study is literature-based and reports no experiments.

Objectives. The report has five objectives: (1) to define centralized, distributed, and decentralized databases precisely enough to compare them; (2) to explain the mechanisms (fragmentation, replication, distributed commit, consensus, and consistency models) that give each approach its properties; (3) to relate those mechanisms to consistency, availability, latency, scalability, cost, security,

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

and governance; (4) to describe application settings in which each approach is typically used; and (5) to propose a conceptual decision framework that links requirements to architecture.

Scope and method. The study draws on foundational textbooks and papers in database theory [1]–[8], on system papers that describe production architectures [17]–[20], and on analyses of the consistency-availability trade-off [12]–[15]. Sources were chosen because they are widely cited, describe mechanisms precisely, or document the design of real systems. Claims about performance are qualitative, and no figures are quoted where results depend on workload and platform.

Organization. Section II sets out the dependency chain from topology to business outcome. Section III analyzes the two architectures in depth. Section IV considers application settings, and Section V treats security, privacy, and governance. Section VI discusses trade-offs, Section VII proposes the decision framework, and Section VIII lists research-based observations. Sections IX to XII cover future directions, limitations, a final discussion, and the conclusion.

Figure 1 shows the three architectures side by side. Moving from left to right, first the data and then the control over it spread across more sites.

II. RESEARCH SYNTHESIS

The effect of the architectural choice can be read as a chain of dependencies in which each stage constrains the next:

Topology → Data Placement → Coordination → Consistency Model → Availability and Latency → Application Behavior → Business Outcome

A. Topology

Whether the system is one node, a cluster under one authority, or a federation of autonomous sites fixes which failures are possible (node, network partition, whole-site loss) and who can change the system.

B. Data Placement

Fragmentation (partitioning), replication, and allocation decide where each piece of data lives and how many copies exist [4], [5]. Placement determines how much of a query can be answered locally.

C. Coordination

When data or control is spread out, protocols are needed for concurrency control, atomic commitment, and agreement among replicas [7], [8], [10], [11]. Coordination is where most of the cost of distribution appears.

D. Consistency, Availability, and Latency

The consistency model offered to applications results from the coordination choices and trades against availability under network partitions and against latency in normal operation [12]–[14].

E. Application Behavior and Business Outcome

Applications inherit these properties. A payments system needs a single agreed account balance. A product catalogue or social feed can tolerate brief disagreement between copies. The architecture must therefore be chosen from the application's requirements and not from fashion.

Two conclusions follow. First, the chain is cumulative: a highly available store with weak consistency can be unsuitable for financial correctness, and a strongly consistent single node can be unsuitable for global availability. Second, most of the important properties (consistency, availability, recovery, security) are determined by decisions at the DBMS layer.

A conceptual illustration shows how the chain operates. Consider a funds-transfer service and a page-view counter. The transfer debits one account and credits another. If the two accounts sit on different partitions, the transfer needs an atomic commit across them, and a failure midway must not leave the money in neither account or in both [8], [9]. The page-view counter can tolerate a lost or delayed update. Both may be called "distributed," but they sit at very different points on the chain: the first needs coordination and a strong consistency model, and the second gains availability and speed by relaxing both. The same architecture can therefore be right for one workload and wrong for another.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

III. ADVANCED ANALYSIS

A. Centralized Architecture

In a centralized DBMS, one engine controls storage, the buffer cache, the lock manager, and the log. Because all state is in one place, ACID transactions [2] are comparatively simple to provide: locking or multiversion techniques serialize conflicting operations [7], the write-ahead log supports recovery, and a query optimizer sees all statistics and plans over local data [3].

Strengths: simple semantics, strong consistency, straightforward backup and security administration, and no network cost between data and processing. Work on in-memory single-node transaction processing argued that a well-engineered single node can serve many transactional workloads efficiently [36].

Limitations: the node is a single point of failure unless standby copies are added, capacity is bounded by the largest machine that can be obtained (vertical scaling), and users far from the site experience network latency. Adding a hot standby with replication moves the design partly toward a distributed one.

B. Distributed and Decentralized Architectures

Distributed systems are commonly classified by what the nodes share. Shared-nothing designs give each node its own memory and disk and communicate only over the network, which scales well for partitioned workloads [6]. Shared-disk and shared-memory designs trade simpler data access for contention. Cloud warehouses such as the one in [24] separate storage from compute so that each scales independently.

Decentralization adds a further dimension: autonomy. In a multidatabase or federated setting, sites keep their own schemas and administrators and cooperate through agreed interfaces [4]. In peer-to-peer and leaderless stores, no node is privileged, and any node may accept requests [17], [18]. Ledger-style systems extend this to mutually distrusting parties [34]; this report treats them only as a boundary case and makes no claim about their suitability for general database workloads.

C. Fragmentation, Replication, and Allocation

Fragmentation splits a relation horizontally (by rows, for example by region or key range), vertically (by columns), or in mixed form. Hash and range partitioning are common strategies [4]. Good partitioning keeps frequent queries on one site and spreads load evenly. Poor partitioning creates hot spots and expensive cross-site operations.

Replication keeps several copies for availability and read performance. Synchronous replication keeps copies identical at the cost of latency. Asynchronous replication is faster but allows copies to diverge for a time. Replication that allows updates at any copy leads to conflicts, which grow with scale unless handled explicitly [16]. Techniques include primary-copy designs, quorum reads and writes, version vectors, and conflict-free replicated data types [17], [23].

Allocation decides which fragments and replicas go to which site. It balances locality, load, failure domains, and legal constraints on where data may be stored.

D. Transaction Management

A transaction that touches several sites must still be atomic. The classical two-phase commit protocol coordinates this, but a failed coordinator can leave participants blocked, which motivated non-blocking designs [9]. Modern systems combine atomic commitment with replicated logs maintained by consensus protocols such as Paxos [10] and Raft [11], and the relationship between commit and consensus has been analyzed formally [8]. Globally distributed systems such as Spanner provide strongly consistent transactions across regions by combining replication with carefully bounded clock uncertainty [20]. This shows that strong consistency at scale is possible, but it depends on specialized infrastructure and adds latency for cross-region writes.

Figure 3 shows the two phases of atomic commit. In the first phase, the coordinator asks every participant to prepare, and each answers with its vote. In the second phase, the coordinator announces the decision. A participant that has voted yes must wait for that decision, and if the coordinator fails in this window, the participant stays blocked while holding its locks. Non-blocking protocols and coordinators replicated by consensus address this weakness [8], [9].

Concurrency control in distributed settings (distributed locking, timestamp ordering, multiversion methods) must also deal with message delay and partial failure [7]. Deadlock detection becomes harder because wait-for relationships span sites.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

E. Consistency and the CAP Trade-off

The CAP argument states that when a network partition occurs, a replicated service cannot be both fully consistent and fully available [12]. Later discussion stresses that the choice is not binary and that it applies only during partitions [13]. A fuller view is that even without partitions, systems trade consistency against latency [14]. Between strict (linearizable) consistency and weak consistency lies a range of models, including eventual consistency, where replicas converge if updates stop [15], [22].

Centralized systems largely avoid this trade-off because there is one authoritative copy. Decentralized systems must make the choice explicitly, and applications must be designed to tolerate the model chosen.

F. Query Processing

In a distributed DBMS, the optimizer must consider the network cost of moving data, the location of fragments, and the availability of local indexes, in addition to the usual CPU and I/O costs [4]. Operations such as distributed joins and global aggregations can be expensive unless data is co-located or reduced before transfer. In centralized systems, these costs do not arise, so a centralized optimizer is simpler and has more complete information.

G. Fault Tolerance and Availability

A centralized system recovers from crashes using its log [3], but remains unavailable until it restarts or a standby takes over. Distributed systems replicate data and use failure detection and leader election to continue serving requests despite node loss [10], [11]. The price is operational complexity: more components to configure, monitor, and secure, and failure modes such as partial network partitions that are hard to test.

H. Scalability

Centralized systems scale vertically, which is simple but bounded. Distributed systems scale horizontally by adding nodes, which can raise capacity but introduces coordination overhead, rebalancing, and the need for sound partitioning [4], [6], [21]. Scalability is therefore not free: the gain depends on workload patterns and on how much cross-partition work is required.

I. Security, Privacy, and Governance

Security is discussed in Section V. Governance, meaning ownership, definitions, quality rules, and accountability, is easier to enforce when data is held centrally and becomes harder as authority is spread out [29]. Without shared definitions, the same metric can be calculated differently at different sites.

J. Cloud and NoSQL Realizations

Cloud platforms offer managed replication and elasticity, which lowers the effort of running distributed systems, though usage-based billing makes cost depend on workload behavior [24]. NoSQL systems relax parts of the relational model to gain horizontal scalability, high write throughput, or flexible schemas [21]. Examples include the key-value store Dynamo [17], the wide-column systems Bigtable [19] and Cassandra [18]. Their consistency models vary and should be examined before adoption [22]. Many production systems are hybrids that keep a strongly consistent relational core and add distributed components for scale.

K. Data Integration and Heterogeneity

Decentralized settings usually involve sources whose schemas, formats, and semantics differ. Federated database systems address this by letting each component database keep its autonomy while presenting a unified view [38]. Autonomy has several aspects: design autonomy (each site chooses its own schema), communication autonomy (each site decides whether to take part), and execution autonomy (each site controls its own transactions). Higher autonomy makes global transactions harder, because a component cannot be forced to hold locks or wait on behalf of others.

Data integration theory describes how sources are related to a global schema, either by defining the global schema as views over the sources or the sources as views over the global schema [39]. Two practical responses exist. The centralizing response copies data into one store through extraction, transformation, and loading, which gives consistent definitions at the cost of latency and pipeline maintenance. The decentralizing response queries data where it lives, which gives fresher results but makes each answer depend on the availability and speed of every participating source.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

L. Data Quality and Integrity across Sites

Data quality is multidimensional and covers accuracy, completeness, consistency, and timeliness as perceived by data consumers [30]. Integrity constraints such as primary keys, foreign keys, CHECK, and NOT NULL are simple to enforce on a single node [3]. Across sites they become harder. A foreign key may reference a row held at another site, so checking it needs a remote lookup or a distributed transaction. Uniqueness across partitions needs either the partition key inside the unique constraint or a global coordination step. For this reason, many partitioned systems restrict cross-partition constraints or move them into the application.

Autonomous sites also create duplicate and conflicting records for the same real-world entity, which calls for entity resolution and agreed identifiers. Constraints cannot enforce semantic correctness in any architecture, since a value can satisfy every rule and still be wrong. Shared definitions and validation before data is merged are therefore needed in centralized and decentralized designs alike.

M. Backup, Recovery, and Disaster Recovery

A centralized DBMS recovers from crashes through write-ahead logging and checkpoints [3]. Protection against loss of the site depends on backups and standby copies. Two quantities describe the result: how much recent work may be lost (the recovery point) and how long service is down (the recovery time). Asynchronous replication to a standby keeps latency low but can lose the latest transactions on failover. Synchronous replication avoids that loss at the cost of write latency.

Distributed systems replicate data across failure domains such as nodes, racks, zones, and regions. Consensus-based replication continues to serve requests while a majority of replicas is reachable [10], [11], and a failed replica is rebuilt by replaying the log or copying a snapshot. Replication is not a substitute for backup, because it copies mistakes such as accidental deletions as faithfully as it copies data. Restoring many partitions to one consistent point also needs coordinated snapshots, and decentralized sites with independent backup schedules may restore to different points in time.

N. Performance Considerations: Latency, Locality, and Contention

In a centralized system, latency is governed by CPU, memory, and storage, with no network hop between the engine and its data. In a distributed system, network round trips become a major cost. An operation that spans partitions needs coordination messages, and a commit across regions must wait for at least one inter-site round trip [20]. Placing data that is used together on the same site is therefore one of the most effective design choices [4].

Skew and contention limit the benefit of adding nodes. If many requests target the same key, one partition becomes a hot spot whatever the cluster size. A request that fans out to many partitions completes only when the slowest responds, so tail latency can grow with the number of participants. Weaker consistency can lower latency, but it moves reconciliation work to the application and to users who read stale data [14], [40]. This report quotes no speed-up figures, because outcomes depend on workload and system.

O. Cost and Operational Considerations

Cost has several parts: hardware or cloud capacity, licenses, network transfer, storage for replicas, tooling, and staff time. A centralized system is usually cheapest to run at small scale, and high-end single machines are often priced at a premium. Distributed systems can use commodity nodes [6], but replication multiplies storage, cross-region traffic is often billed, and operations become more demanding: monitoring, rolling upgrades, schema changes across nodes, and incident response all take more effort. Managed cloud services shift some of that effort to the provider, while usage-based billing makes cost depend on query and workload behavior [24]. Team skill is part of the cost, since diagnosing failures across many nodes is a specialized task [31].

P. Migration from Centralized to Distributed Designs

Organizations rarely adopt a distributed design on day one. A common path has five steps: (1) tune the single node with indexes, query changes, and hardware; (2) add read replicas; (3) add caching for hot data; (4) partition by a key such as tenant or region; and (5) adopt a distributed database for multi-region or very large workloads. Partitioning by entity keeps most transactions inside one partition, which avoids distributed commit for them [41].

The main risks are re-partitioning when data grows unevenly, queries that cross partitions, and application changes to handle weaker guarantees. Migration should be triggered by measured limits and not by anticipation, because each step adds complexity that a smaller system does not need.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Q. Concurrency Control and Isolation in Distributed Settings

On a single node, serializable execution is obtained by well-understood techniques. Two-phase locking requires a transaction to acquire all its locks before releasing any, which guarantees serializability but can block and deadlock. Multiversion concurrency control lets readers see a consistent snapshot of the data without blocking writers, which suits read-heavy workloads [7]. Many systems offer isolation levels weaker than serializable for speed, and these admit anomalies that application designers must understand [44].

In a distributed system, the same ideas must work without shared memory. Locks are held at several sites, and each lock request and release is a network message. Deadlocks can span sites, so detection needs either a global wait-for graph assembled from local ones or time-outs that abort suspected waiters [7]. Ordering transactions by timestamp requires a way to order events across nodes. Logical clocks give an order consistent with causality without synchronized physical clocks [46], while systems that need real-time ordering of transactions must bound clock uncertainty, as Spanner does [20]. Weaker isolation levels such as snapshot isolation remain attractive in distributed settings because they avoid cross-site read locks, but they permit anomalies such as write skew that serializable execution would prevent [44].

IV. REPLICATION PROTOCOLS

Replication protocols differ in where updates may be accepted and when a write counts as complete.

Primary-backup: One replica orders all writes and ships its log to the others. Failover promotes a backup. Synchronous shipping avoids loss at the price of latency, and asynchronous shipping does the reverse.

Multi-primary (update-anywhere): Several replicas accept writes, which improves availability and locality but creates conflicts when two sites update the same item. Analysis shows that coordination and reconciliation problems grow much faster than linearly as nodes and traffic increase [16]. Conflicts are resolved by rules such as last-writer-wins, which can silently discard an update, by version vectors that expose concurrent updates to the application, or by data types designed to merge automatically [17], [23].

Quorum replication: A write is acknowledged by W of N replicas and a read consults R replicas. If the quorums overlap, a read sees at least one up-to-date copy [17]. The condition is:

R + W > N

Dynamo adds a sloppy quorum with hinted handoff, so that writes continue when some replicas are unreachable, and uses Merkle trees to repair divergence between replicas in the background [17].

Consensus-based state machine replication: Protocols such as Paxos [10] and Raft [11] elect a leader that appends each command to a replicated log and treats it as committed once a majority has stored it. A group of 2f + 1 replicas keeps working while f of them fail, and no two replicas disagree about the order of committed commands.

V. PARTITIONING STRATEGIES

Partitioning decides which node owns which data, and each strategy favors some queries over others [4].

Range partitioning assigns contiguous key ranges to nodes. It supports efficient range scans, but monotonic keys such as timestamps send all new writes to one partition.

Hash partitioning spreads keys evenly but gives up key order, so range queries must visit every partition.

Consistent hashing places nodes and keys on a ring so that adding or removing a node moves only a fraction of the keys [47]. Dynamo uses it for incremental scaling [17].

Directory-based partitioning keeps a lookup service that maps keys to nodes. It is flexible, but the directory becomes a component that must itself be replicated and protected.

Geographic partitioning places data near its users or inside a legal region, which lowers latency and supports residency requirements.

Two operational issues recur. First, skew: a few very popular keys can overload one node whatever the hashing scheme. Second, secondary indexes: a local index on each partition is cheap to update but needs a query to visit every partition, while a global index

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

answers queries from one place but needs a distributed update on every write. Rebalancing partitions while the system stays online is a significant operational task.

VI. THE SPECTRUM OF CONSISTENCY MODELS

Consistency is a spectrum and not a switch. **Table I** lists the models most relevant to database design, ordered from strongest to weakest.

Table I. Consistency models and their trade-offs.

Model	Guarantee	Typical cost	Typical use
Linearizable [45]	Each operation appears to take effect at one instant, and reads return the latest write	Coordination latency, reduced availability during partitions [12]	Account balances, stock reservation
Serializable transactions [7]	Outcome equals some serial order of the transactions	Locking or validation across sites	Funds transfer, order placement
Causal [46]	Operations that depend on each other are seen in order	Metadata to track dependencies	Comments and replies
Read-your-writes (session) [15]	A client always sees its own updates	Session affinity or version tracking	Profile edits
Eventual [15], [40]	Replicas converge if updates stop	Temporary anomalies, conflict resolution	Counters, feeds, catalogues

The consistency-availability argument [12] applies during a network partition. The PACELC formulation extends it: during a partition a system chooses availability or consistency, and otherwise it chooses between latency and consistency [14]. Dynamo-style stores favor availability and low latency, whereas Spanner-style systems favor consistency and accept higher latency [14], [17], [20]. A single-node database does not face the choice, because it has only one copy to keep consistent.

VII. SUMMARY COMPARISON

Table II. Comparison of architectural approaches.

Dimension	Centralized	Distributed (single authority)	Decentralized (autonomous or peer-to-peer)
Data location	One site	Several sites, one logical database [4]	Several sites, each with its own control
Consistency	Strong and simple [2]	Strong or tunable, at a cost [20]	Often eventual or application-managed [15], [17]
Availability	Single point of failure unless standby	High through replication [11]	High, no single failure point
Scalability	Vertical, bounded	Horizontal [6]	Horizontal, can be very large

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Dimension	Centralized	Distributed (single authority)	Decentralized (autonomous or peer-to-peer)
Latency	Higher	Lower with local replicas	Lower with local replicas

The entries are qualitative generalizations; real systems vary.

VIII. PRACTICAL APPLICATION PERSPECTIVE

A. Application Settings

The examples describe use cases conceptually, without performance figures.

Banking and payments: Account balances require strict ACID guarantees [2]. Core ledgers are often centralized or use strongly consistent replicated designs [20], while channels such as reporting and mobile access are served from replicas.

Retail and e-commerce: Catalogues, carts, and recommendations can tolerate brief inconsistency and benefit from geographic distribution [17], while order and payment records need stronger guarantees.

Healthcare: Patient records are sensitive and held by many independent providers. Federation with strict access control and de-identification for analytics is common, and privacy obligations constrain replication across sites [26].

Telecommunications and IoT: Large volumes of events arrive continuously from many locations. Partitioned, write-optimized stores suit ingestion [18], [19], with downstream aggregation.

Government and public services: Multiple departments maintain separate systems. Federated access with shared identifiers and defined data-sharing rules is a typical pattern, with data-protection law shaping what may move [27], [28].

Education: A single institution with moderate data volume is often well served by a centralized relational database, with backup replication. Multi-campus systems may partition by campus.

Global SaaS applications: Users in several regions benefit from regional replicas and partitioning by tenant or geography [4], [20].

The settings differ mainly in what they cannot compromise on. **Table III** summarizes the dominant requirement, the usual architectural pattern, and the main risk in each.

Table III. Application settings and architectural patterns (conceptual; no performance figures).

Domain	Dominant requirement	Typical pattern	Main risk
Banking and payments	Strict correctness	Centralized or strongly consistent replicated core, with replicas for reporting [2], [20]	Cost and latency of cross-site commit
Retail and e-commerce	Availability and scale	Partitioned, replicated stores for catalogue and carts, stronger store for orders [17]	Stale carts, overselling
Healthcare	Privacy across many owners	Federation with strict access control and de-identification [26]	Identity mismatch, privacy exposure
Telecommunications and IoT	Write throughput	Partitioned wide-column or key-value stores [18], [19]	Hot partitions

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Domain	Dominant requirement	Typical pattern	Main risk
Government	Autonomous departments	Federated access with shared identifiers and sharing rules [27], [28]	Inconsistent definitions
Education	Modest scale, low cost	Centralized relational database with backup replication	Single point of failure
Global SaaS	Low latency in many regions	Regional replicas, partitioning by tenant or geography [20]	Cross-region cost

A common pattern across the table is to place the strongest guarantee where an error is most costly and to relax it elsewhere. An organization may therefore run a centralized core for ledgers and orders, distributed stores for high-volume events, and federated access for data owned by partners. The choice is made per data set and not once for the whole organization.

B. Reference Architectures from the Literature

Published system papers show how the mechanisms of Section III are combined in practice. The descriptions below summarize the cited papers, and details may have changed in current products.

Centralized relational engine. A single DBMS process manages storage, buffering, locking, logging, and query optimization over one copy of the data [3]. Availability is improved by shipping the log to a standby. This remains the reference design for transactional correctness and the baseline against which distributed designs are judged.

Dynamo. Amazon's key-value store is designed so that writes are always accepted. It partitions data by consistent hashing, replicates each item on several nodes, tracks concurrent updates with version vectors, uses a sloppy quorum with hinted handoff, and repairs replicas with Merkle trees. Conflicts are resolved by the application or by simple rules, so the system provides eventual consistency [17].

Bigtable. Google's system is a distributed, sorted, sparse map. Tables are split into row ranges that are served by many nodes, the data is kept in a distributed file system, and a lock service supports coordination. It offers atomic operations on a single row and not general multi-row transactions [19].

Cassandra. Cassandra combines Dynamo-style peer-to-peer distribution, in which every node has the same role, with a Bigtable-style data model [18]. It illustrates a decentralized design with no single point of failure.

Spanner. Spanner shards data across many sites and replicates each shard with Paxos. Transactions that span shards use two-phase commit across the replicated groups, and TrueTime, an interface that exposes bounded clock uncertainty, lets transactions be ordered consistently with real time [20]. It shows that strong consistency and global distribution can coexist, at the cost of specialized infrastructure and commit latency.

Cloud data warehouse. Snowflake separates storage, held in a shared object store, from compute clusters that can be scaled and isolated independently, with a shared metadata layer [24]. This is a different form of decoupling: the data is shared, while the processing is elastic.

Federated multidatabase. Component databases stay autonomous while a mediator layer presents a global schema and routes queries [38]. It sits at the autonomy end of the spectrum, where global transactions are the hardest to support.

IX. SECURITY, PRIVACY, AND GOVERNANCE

Centralization concentrates risk: a breach of one system can expose everything, but there is one place to defend, audit, and patch. Decentralization reduces the loss from any single compromise but enlarges the attack surface, since more nodes, links, and administrators must be secured.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

A. Authentication, Authorization, and RBAC

Control catalogues list authentication and authorization as basic controls [26]. Role-based access control assigns permissions to roles and roles to users [25]. In a decentralized setting, identity must be trusted across sites, which requires agreed identity management, and local role definitions can drift apart.

B. Encryption and Audit Logging

Data must be protected at rest and in transit [26]. In distributed systems, replication and inter-node traffic multiply the channels to protect. Audit logs record access and changes, but in decentralized designs logs are spread across sites and need consolidation to be useful.

C. Privacy and Regulation

The EU GDPR [27] and India's Digital Personal Data Protection Act, 2023 [28] set obligations for personal data. Replication across regions can raise questions about where data is stored and processed. Techniques such as federated learning aim to analyze data without centralizing it [33]. This report does not interpret these laws, and applicable requirements should be confirmed with qualified advisers.

TABLE IV. SECURITY AND GOVERNANCE IMPLICATIONS.

Concern	Centralized	Decentralized
Attack surface	Smaller	Larger: more nodes and links
Impact of one breach	Potentially total	Often limited to the affected site
Access policy	One policy, easy to apply [25]	Must be coordinated across sites
Audit	Single log source [26]	Logs must be consolidated
Data residency	Single location	Possible per-region control, but more complex
Governance	Single owner [29]	Shared ownership, agreed standards [32]

D. Threats Specific to Distributed Systems

Spreading data across sites introduces threats that a single node does not face. More nodes and links mean more points to defend, and traffic between nodes (replication streams, coordination messages) must be authenticated and encrypted [26]. Encryption keys have to be managed and rotated across many nodes. Replicas, backups, and snapshots are further copies of the data and must receive the same protection as the primary. Privileged administrators exist at every site, so separation of duties and audit logging of administrative actions matter more as the number of administrators grows [25], [26].

E. Governance Models

Centralized governance assigns ownership, definitions, and quality rules to one body, which gives consistency and a single point of accountability but can become a bottleneck. Federated governance assigns ownership to domains and sets shared standards for identifiers, definitions, security, and quality [29], [32]. Hybrid models keep the standards central and delegate operation. In each model, metadata, a data catalogue, and agreed definitions of key measures are what keep different sites from computing the same figure in different ways [29].

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

F. Data Sovereignty and Residency

Laws may restrict where personal data is stored or transferred [27], [28]. Geographic partitioning and region-bound replicas allow a decentralized system to keep each region's data inside its borders. Backups, logs, and analytical copies are also data and fall within the same constraint. Residency requirements limit how far data can be replicated for availability, so a design that must keep data inside one region cannot use cross-border replicas as protection against regional loss. This tension between residency and availability should be settled early in design. This report does not interpret the law, and applicable requirements should be confirmed with qualified advisers.

X. CRITICAL DISCUSSION

A. Consistency versus Availability and Latency

Strong consistency across sites requires coordination, which costs time and reduces availability during partitions [12]–[14]. Relaxed consistency improves both but moves the burden of handling anomalies to applications.

B. Simplicity versus Scale

A single node is far easier to build, operate, and reason about. Distribution is justified when requirements exceed what one node can offer. For modest data volumes and tolerable downtime, a well-indexed centralized database may be sufficient [3], [36].

C. Fault Tolerance versus Operational Complexity

Replication removes the single point of failure but adds components and failure modes, so a poorly operated distributed system can be less reliable than a well-run centralized one [31].

D. Locality versus Global Coordination

Placing data near users reduces latency, but operations spanning regions need cross-site coordination [20]. Good partitioning reduces such operations.

E. Autonomy versus Governance

Decentralized ownership lets domains move independently, but consistent definitions and quality rules need shared standards. Approaches such as data mesh propose federated governance with domain-owned data products [32]; their effectiveness at scale remains a subject of ongoing practice and study.

F. Cost

Centralization can reduce infrastructure and administration cost for small systems. Distribution can reduce cost at large scale through commodity hardware [6], but network transfer, replication storage, and operations effort add expense. Usage-based cloud pricing links cost to workload behavior [24].

G. Skills and Operational Maturity

A distributed design is only as reliable as the team that runs it. Failure modes such as partial network partitions, clock drift, and replica divergence are hard to reproduce and diagnose [31]. A team without that experience may achieve better availability from a simple, well-monitored centralized system with a tested standby than from a distributed one it cannot fully understand.

H. Vendor Dependence and Portability

Managed distributed services reduce operational effort but bind the application to the provider's consistency model, query dialect, and pricing [24]. Portable designs rely on standard interfaces and keep consistency assumptions explicit, so that a later migration is possible. Whether the reduction in effort outweighs the loss of portability depends on the organization's size and strategy.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

XI. RISK SUMMARY

Table V summarizes the main risks and how they differ between the two approaches.

Table V. Risk summary by architecture.

Risk	Centralized	Decentralized	Mitigation
Site outage	Service stops until recovery	Survives if replicas sit elsewhere	Replicas in separate failure domains [11]
Capacity exhaustion	Hard ceiling of one machine	Add nodes, but rebalancing is needed	Partitioning and monitoring [4]
Inconsistent reads	Rare	Possible under weak models	Choose the model per data set [14]
Data breach	One large target	More targets, smaller blast radius	Encryption and access control at each site [25], [26]
Conflicting updates	Not applicable	Divergent replicas	Primary copy, quorums, mergeable types [17], [23]
Metric drift	Low	High	Shared definitions and catalogue [29]
Cost overrun	Over-provisioned hardware	Replication and network charges	Capacity planning and monitoring [24]
Operational error	Simpler procedures	Complex failure modes	Automation, runbooks, training [31]

Because these trade-offs depend on workload, consistency needs, latency targets, regulatory constraints, team capability, and budget, the architecture should be chosen case by case and not by adopting the most recent technology.

XII. INTEGRATED ARCHITECTURAL AND DECISION MODEL

A. Decision Framework

The model below is a design synthesis drawn from the literature. It is not an implemented or measured system.

Requirements Analysis → Consistency and Latency Needs → Scale and Availability Needs → Governance and Regulatory Constraints → Architecture Selection → Data Placement (Partition/Replicate) → Coordination Mechanism → Security and Monitoring → Operation and Review

Security and governance are cross-cutting concerns and apply to every step.

Figure 2 reads from top to bottom. A system starts with the simplest option and moves to a more distributed one only when a requirement rules the simpler one out. The question order is a simplification: in practice cost, team skill, and regulation (**Table VI**) also shape the result.

Worked examples. A university with one campus, moderate data volume, and a few hours of tolerable downtime answers yes to the first question and is served by a centralized relational database with a standby copy. A payments operator that cannot tolerate lost updates but must serve several regions answers no to the first question and yes to the second, which points to consensus-replicated distributed SQL and accepts the latency of cross-region commits [20]. A health information exchange whose data is owned by many hospitals answers yes to the third question and needs federation with shared standards and strict access control [38]. A global product catalogue that tolerates brief divergence reaches the last box, a sharded store with tunable consistency [17]. These examples are illustrations and not validated case studies.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Table VI. Decision framework.

Question	If the answer is yes, favor	Source
Do transactions require strict, global consistency with moderate scale?	Centralized, or strongly consistent replicated system	[2], [20]
Must service continue through node or site loss?	Replication with consensus	[10], [11]
Is the data volume or write rate beyond one node?	Partitioned (sharded) distributed design	[4], [6]
Are users spread across regions needing low latency?	Regional replicas, geo-partitioning	[20]
Can the application tolerate temporary divergence?	Eventually consistent decentralized store	[15], [17]
Do independent organizations or units own the data?	Federation with shared standards	[4], [32]
Is data residency legally restricted?	Region-bound partitions, privacy controls	[27], [28]

A hybrid pattern is common: a centralized, strongly consistent core for critical transactions, surrounded by distributed read replicas, caches, and analytical stores. Analytical systems may follow the data-warehouse pattern of separating analytical from operational workloads [37].

B. Proposed Evaluation Methodology

Because this study is literature-based, the framework has not been tested. This subsection outlines how it could be validated. Standard workloads exist for the purpose: YCSB defines simple read and update mixes for cloud serving systems [42], and TPC-C defines a transactional order-entry workload [43]. The same workload would be run against a centralized configuration and against distributed configurations with increasing node counts, both with and without injected failures. Results would be reported as distributions and not single averages, with repeated runs, warm-up periods, and the hardware recorded. **Table VII** lists the criteria.

Table VII. Evaluation criteria for a future empirical study.

Criterion	Metric	How measured	Question answered
Latency	Median, 95th and 99th percentile response time	Client logs under fixed load	Does distribution add tail latency?
Scalability	Operations per second as nodes are added	Scale nodes and workload together	Does capacity grow with nodes?
Availability	Share of requests served during failures	Fault injection: node, zone, partition	Does service survive failure?
Recovery	Time to resume service, data lost	Stop the primary or a node and observe	Are recovery targets met?
Consistency	Share of stale or conflicting reads	Write, then read from other replicas	Which anomalies reach applications?

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Criterion	Metric	How measured	Question answered
Cost	Cost per unit of work	Resource accounting or billing data	Is distribution economical?
Operability	Effort for upgrades, schema changes, diagnosis	Effort log or user study	Can the team run it?

Threats to validity include differences between products, workloads that do not represent a real application, and configurations tuned unequally. A fair comparison tunes each system to a similar standard and publishes its settings.

XIII. RESEARCH-BASED OBSERVATIONS

1. **Centralization simplifies correctness.** One authoritative copy avoids replica divergence and distributed commit, making ACID semantics easier to provide [2], [7].
2. **Distribution buys scale and availability at the cost of coordination.** Partitioning and replication raise capacity and fault tolerance, but cross-site operations are expensive [4], [6].
3. **Consistency is a design choice in decentralized systems.** The CAP and latency trade-offs mean the model must match application needs [12]–[14].
4. **Consensus makes strong consistency feasible at scale, with latency.** Replicated logs and carefully designed systems demonstrate this, though they require specialized design [10], [11], [20].
5. **Partitioning quality determines performance.** Poor placement produces hot spots and costly distributed operations [4].
6. **Security and governance are harder as authority spreads.** More nodes and owners require explicit shared policy [25], [26], [29].
7. **Hybrid architectures are common in practice.** Systems often combine a consistent core with distributed components [20], [31].

Table VIII. Technical challenge and mitigation.

Challenge	Mitigation	Source
Single point of failure	Standby or replicated cluster	[3], [11]
Capacity ceiling	Partitioning, horizontal scaling	[4], [6]
Replica conflicts	Primary-copy, quorums, CRDTs	[16], [17], [23]
Blocking in distributed commit	Consensus-based replication, non-blocking protocols	[9], [10], [11]
Inconsistent metrics across sites	Shared definitions, federated governance	[29], [32]
Fragmented permissions	Unified identity and RBAC	[25], [26]
Cross-region data restrictions	Region-bound placement	[27], [28]

Three further observations complete the list.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

8. **Failure differs in kind.** A centralized system fails as a whole and recovers from its log. A distributed system fails partially: a node may be down, a link slow, or a network split in two. Partial failure is the harder case to design for and to test [4], [10], [11].
9. **Strong consistency at scale is possible but costly.** Consensus and bounded clock uncertainty allow strongly consistent transactions across regions, at the cost of coordination latency and specialized infrastructure [10], [11], [20].
10. **Governance does not decentralize automatically.** Spreading data across sites does not spread agreement on definitions, identifiers, or access rules. These need explicit shared standards and ownership [29], [32].

XIV. FUTURE SCOPE

The following are research directions, not established facts, and their maturity varies.

Globally distributed SQL: Systems that combine relational semantics with horizontal scale, as pioneered in [20], continue to develop.

Autonomous database management: Research proposes self-driving systems that tune configuration and adapt to workload [35]. Their reliability in production is still developing.

Federated and privacy-preserving analytics: Learning and analysis over data that stays at its source [33] reduces data movement but involves trade-offs in accuracy and complexity.

Data mesh and federated governance: Domain-owned data with shared standards [32] is a proposed organizational pattern that needs further evidence.

Ledger and peer-to-peer databases: Systems for mutually distrusting parties [34] address trust rather than performance, and general suitability for database workloads remains limited.

Conflict-free data types: Convergent replicated structures [23] reduce coordination for suitable data models.

Table IX. Future research directions in centralized and decentralized databases.

Direction	Basis	Status	Open issue
Globally distributed SQL [20]	Production systems and research	Maturing	Cost and latency of cross-region commit
Self-driving DBMS [35]	Research	Developing	Reliability and explainability
Federated learning and analytics [33]	Research and early deployment	Emerging	Accuracy, privacy and data heterogeneity
Data mesh [32]	Practitioner proposal	Emerging	Evidence of effectiveness, governance at scale
Ledger and peer-to-peer databases [34]	Research and niche deployment	Emerging	Throughput and fit for general workloads
Conflict-free replicated data types [23]	Research with some adoption	Developing	Limited to data types with merge rules

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

XV. LIMITATIONS

This study is literature-based. No benchmarking, implementation, or user study was carried out, so statements about performance are qualitative. Results for one platform may not transfer to another, because products differ in design. The terms "distributed" and "decentralized" are used inconsistently in the literature, so classifications in this report are simplifications. Cloud and distributed database technology changes quickly, and some cited work describes systems as they were at publication, so current products may differ in detail. The regulatory discussion is technical in scope and not legal analysis.

XVI. FINAL DISCUSSION

The analysis supports one argument. The placement of data and control is a foundational DBMS decision, not a deployment detail, because it determines consistency, availability, latency, scalability, security, and governance.

Consistency and correctness: Centralization provides them most simply. Distribution can provide them at higher cost [2], [20].

Availability and scale: Replication and partitioning provide them, with added coordination and operational complexity [4], [11].

Security and governance: These are easiest to apply from one place and need explicit shared design as authority spreads [25], [29].

Architecture selection: The right choice depends on workload and context, and hybrids are common.

The evidence therefore does not support declaring either approach superior. It supports matching the architecture to requirements and being explicit about the properties given up.

XVII. CONCLUSION

This report compared centralized and decentralized database architectures through a literature-based synthesis. Centralized DBMSs offer simplicity, strong consistency, and unified control, but are limited by single-site capacity and failure. Decentralized and distributed DBMSs offer scalability, availability, and locality, but require partitioning, replication, consensus, and governance mechanisms, and force explicit choices between consistency, availability, and latency. Security, privacy, and governance become more demanding as control is spread. The layered decision framework proposed here is a conceptual aid for selecting among these options. Its validation through implementation and benchmarking is left for further work.

Appendix A. Glossary

- **ACID:** the transaction properties of atomicity, consistency, isolation, and durability [2].
- **Atomic commit:** a protocol that makes a transaction spanning several sites either commit everywhere or abort everywhere [8].
- **CAP:** the result that, during a network partition, a replicated service cannot be both fully consistent and fully available [12].
- **Consensus:** agreement among replicas on a single value or order of commands despite failures, as in Paxos and Raft [10], [11].
- **Consistent hashing:** a placement scheme in which adding or removing a node moves only a fraction of the keys [47].
- **CRDT:** a conflict-free replicated data type whose concurrent updates merge automatically [23].
- **Eventual consistency:** a guarantee that replicas converge if updates stop [15].
- **Federation:** a set of autonomous databases presented through a common interface [38].
- **Fragmentation:** splitting a relation into pieces stored at different sites [4].
- **Horizontal scaling:** adding nodes to increase capacity. Vertical scaling adds resources to one node.
- **Linearizability:** a strong consistency model in which every operation appears to take effect at one instant [45].
- **MVCC:** multiversion concurrency control, in which readers see a snapshot and do not block writers [7].
- **PACELC:** an extension of CAP that adds the latency versus consistency choice made when there is no partition [14].
- **Quorum:** the number of replicas that must acknowledge a read or write [17].

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

- **RBAC:** role-based access control, which grants permissions to roles and roles to users [25].
- **Shared-nothing:** an architecture in which each node has its own memory and disk [6].
- **Sharding:** horizontal partitioning of data across nodes [4].
- **Two-phase commit:** the classical atomic commit protocol with a prepare phase and a decision phase [8], [9].

Appendix B. Design Checklist

The following questions summarize the framework of Section XII and can be used as a review list.

1. What are the consistency requirements of each data set, and which operations must never observe stale data?
2. What availability target applies, and what is the cost of an hour of downtime?
3. How much data loss is acceptable after a failure, and how long may recovery take?
4. What data volume and growth rate are expected, and can one node meet them with headroom?
5. Where are the users, and what latency do they need?
6. Which operations touch more than one entity, and can the data be partitioned so that most transactions stay in one partition?
7. Are there legal or contractual limits on where data may be stored or processed?
8. Who owns each data set, and who may change its definitions?
9. How will identity, roles, and encryption keys be managed across sites?
10. How will backups be taken and restored to a consistent point?
11. What monitoring, alerting, and runbooks exist for partial failures?
12. What skills does the team have, and what training is needed?
13. What is the expected total cost, including replication storage, network transfer, and staff time?
14. How portable must the design be if the platform or provider changes?
15. Which measured limit would trigger a move to the next architectural step?

REFERENCES

[1] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.
[2] J. Gray, "The transaction concept: Virtues and limitations," in *Proc. 7th Int. Conf. Very Large Data Bases (VLDB)*, Cannes, France, 1981, pp. 144–154.
[3] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill, 2019.
[4] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 4th ed. Cham, Switzerland: Springer, 2020.
[5] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*. New York, NY, USA: McGraw-Hill, 1984.
[6] M. Stonebraker, "The case for shared nothing," *IEEE Database Eng. Bull.*, vol. 9, no. 1, pp. 4–9, 1986.
[7] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, MA, USA: Addison-Wesley, 1987.
[8] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Trans. Database Syst.*, vol. 31, no. 1, pp. 133–160, Mar. 2006.
[9] D. Skeen, "Nonblocking commit protocols," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Ann Arbor, MI, USA, 1981, pp. 133–142.
[10] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998.
[11] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Philadelphia, PA, USA, 2014, pp. 305–319.
[12] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002.
[13] E. Brewer, "CAP twelve years later: How the 'rules' have changed," *Computer*, vol. 45, no. 2, pp. 23–29, Feb. 2012, doi: 10.1109/MC.2012.37.
[14] D. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, Feb. 2012.
[15] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, Jan. 2009.
[16] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Montreal, QC, Canada, 1996, pp. 173–182.
[17] G. DeCandia et al., "Dynamo: Amazon's highly available key-value store," in *Proc. 21st ACM Symp. Operating Syst. Principles (SOSP)*, Stevenson, WA, USA, 2007, pp. 205–220.
[18] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.
[19] F. Chang et al., "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, Art. 4, Jun. 2008.
[20] J. C. Corbett et al., "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, Art. 8, Aug. 2013.
[21] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, Dec. 2010.
[22] M. Diogo, B. Cabral, and J. Bernardino, "Consistency models of NoSQL databases," *Future Internet*, vol. 11, no. 2, Art. no. 43, 2019, doi: 10.3390/fi11020043.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

[23] M. Shapiro, N. Preguica, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Proc. 13th Int. Symp. Stabilization, Safety, and Security of Distributed Systems (SSS)*, LNCS vol. 6976, 2011, pp. 386–400.
[24] B. Dageville et al., "The Snowflake elastic data warehouse," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, San Francisco, CA, USA, 2016, pp. 215–226.
[25] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996, doi: 10.1109/2.485845.
[26] Joint Task Force, "Security and privacy controls for information systems and organizations," NIST, Gaithersburg, MD, USA, Special Publication 800-53, Rev. 5, Sep. 2020, doi: 10.6028/NIST.SP.800-53r5.
[27] European Parliament and Council, "Regulation (EU) 2016/679 (General Data Protection Regulation)," *Official J. Eur. Union*, vol. L 119, pp. 1–88, May 2016.
[28] Government of India, "The Digital Personal Data Protection Act, 2023," Act No. 22 of 2023, Aug. 2023.
[29] DAMA International, *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd ed. Basking Ridge, NJ, USA: Technics Publications, 2017.
[30] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *J. Manage. Inf. Syst.*, vol. 12, no. 4, pp. 5–33, 1996.
[31] M. Kleppmann, *Designing Data-Intensive Applications*. Sebastopol, CA, USA: O'Reilly Media, 2017.
[32] Z. Dehghani, *Data Mesh: Delivering Data-Driven Value at Scale*. Sebastopol, CA, USA: O'Reilly Media, 2022.
[33] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS)*, PMLR vol. 54, 2017, pp. 1273–1282.
[34] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[35] A. Pavlo et al., "Self-driving database management systems," in *Proc. 8th Conf. Innovative Data Syst. Res. (CIDR)*, Chaminade, CA, USA, 2017.
[36] M. Stonebraker et al., "The end of an architectural era (it's time for a complete rewrite)," in *Proc. 33rd Int. Conf. Very Large Data Bases (VLDB)*, Vienna, Austria, 2007, pp. 1150–1160.
[37] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM SIGMOD Record*, vol. 26, no. 1, pp. 65–74, Mar. 1997, doi: 10.1145/248603.248616.
[38] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Comput. Surveys*, vol. 22, no. 3, pp. 183–236, Sep. 1990.
[39] M. Lenzerini, "Data integration: A theoretical perspective," in *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst. (PODS)*, Madison, WI, USA, 2002, pp. 233–246.
[40] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Commun. ACM*, vol. 56, no. 5, pp. 55–63, May 2013.
[41] P. Helland, "Life beyond distributed transactions: An apostate's opinion," in *Proc. 3rd Biennial Conf. Innovative Data Syst. Res. (CIDR)*, Asilomar, CA, USA, 2007, pp. 132–141.
[42] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Computing (SoCC)*, Indianapolis, IN, USA, 2010, pp. 143–154.
[43] Transaction Processing Performance Council, "TPC Benchmark C Standard Specification," Rev. 5.11, Feb. 2010.
[44] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, San Jose, CA, USA, 1995, pp. 1–10.
[45] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 3, pp. 463–492, Jul. 1990.
[46] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
[47] D. Karger et al., "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proc. 29th ACM Symp. Theory Comput. (STOC)*, El Paso, TX, USA, 1997, pp. 654–663.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)