Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

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

A Review of Relational Database Management Systems (RDBMS) and Their Modern Applications

Devshree Pathak
Roll No. 241302276

Shatakshi Gupta
Roll No. 241302297

Tanisha Bharti
Roll No. 251302193

Monali Soni
Roll No. 251302298

Dept. of Computer Science & Engineering
School of Engineering and Technology
Shree Guru Gobind Singh Tricentenary University (SGT University), Gurugram, India

Submitted to Ms. Kanika Malik, Assistant Professor, SOET | Subject: Database Management System | Session 2026–2027

***Abstract*—Relational database management systems (RDBMS) remain the usual choice for structured, transaction-oriented data, yet cloud deployment, geographic distribution, and mixed analytical workloads have changed how they are built and selected. This narrative review asks how modern architectures preserve relational correctness while moving computation, logging, and storage; which isolation, recovery, and security duties remain after that move; and how application requirements can become evaluation criteria. Twenty-two scholarly and technical sources are examined; each source carries an evidence grade (passages examined versus abstract only), and every conclusion inherits the lowest grade among its sources. MySQL, PostgreSQL, Oracle Database, and SQL Server are compared on identical criteria: default isolation level, concurrency mechanism, and backup with point-in-time recovery. A secondary analysis of one published Aurora experiment shows database-node network I/Os per transaction falling from 7.4 to 0.95 in a single historical configuration; it is not a product ranking. The review finds that storage elasticity does not establish write scalability, replica consistency does not define application-level isolation, and managed services do not remove retry, secure-query, or backup responsibilities. A requirements-first workflow with a worked example is proposed but not validated. The paper offers an integrated comparison, not a new algorithm or an independent benchmark.**

***Index Terms*—relational databases, RDBMS, cloud-native databases, transaction isolation, point-in-time recovery, database security, narrative review.**

I. I RODUCTION

A. Background

A relational database management system (RDBMS) stores data as related tables, uses keys and constraints to express relationships and business rules, and provides SQL for querying and updating data. Codd's relational model separated the logical description of data from its physical representation, so applications can work with meaningful structures without depending on low-level storage details [1]. Work on the transaction concept described atomicity, consistency, and durability as central properties of reliable state changes, with isolation commonly treated as the fourth ACID property [2]. These ideas explain why MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server remain widely used in banking, healthcare, education, e-commerce, government, and business systems.

Modern applications add requirements: elastic storage, high availability, geographic distribution, analytics beside transactions, and stronger security. Cloud-native database research responds with compute–storage disaggregation, distributed data placement, multi-layer recovery, and hybrid transactional/analytical processing [3]. The relational model has not been replaced; its physical realization has been reorganized.

B. Problem Statement

Organizations must keep growing volumes of data accurate, consistent, secure, and available. The phrase "cloud database" hides large differences: a conventional engine on a virtual machine, a storage-disaggregated engine, and a geographically distributed SQL system solve different problems. Without a clear account of what each architecture and platform changes, and what it leaves to the application, selection can rest on feature lists and popularity instead of requirements.

C. Objectives and Research Questions

Three research questions guide the review. RQ1: how do modern architectures preserve relational correctness while changing the placement of computation and storage? RQ2: which isolation, recovery, and security obligations remain with the platform configuration and the application, and how do the four platforms differ? RQ3: how can application requirements be translated into defensible evaluation criteria? The objectives follow from them: O1, explain relational foundations and transaction correctness (RQ1); O2, compare the four platforms on identical criteria (RQ2); O3, analyze cloud-native and distributed architectures through attributed evidence (RQ1, RQ2); O4, derive a requirements-first workflow (RQ3). Societal importance is treated briefly in Section VII. Table VI gives a separate answer to each question with its evidence grade.

D. Scope and Contributions

Relative to the surveys in Table I, this review adds (i) a grade for the evidence behind every claim; (ii) a platform

(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

comparison on identical criteria, namely default isolation, concurrency mechanism, and backup with point-in-time recovery; (iii) a bounded secondary reading of one published experiment; and (iv) a worked example of a requirements-first workflow. It claims no new algorithm and no independent benchmark. The workflow is not validated, and the security discussion is limited to query safety, access control, and a threat-model framing; a platform-by-platform comparison of encryption and auditing is outside scope.

II. RELATED LITERATURE AND PREVIOUS WORK

Cloud-native designs are described in several system papers. Amazon Aurora moves redo processing and durable storage functions into a distributed storage service while the database engine keeps query processing and transaction management [4]. Spanner presents a globally distributed database with externally consistent transactions, using synchronous replication and a time interface that exposes clock uncertainty [5]; later work describes its development into a SQL system [6]. F1 shows distributed SQL over Spanner-backed storage and the commit-latency cost of synchronous cross-data-center replication [7]. Research on hybrid transactional/analytical processing (HTAP) classifies systems as monolithic or hybrid by data format, divides hybrid designs into four storage architectures, and treats data freshness as a design goal measured, for example, by freshness-score and visibility delay [8].

Earlier surveys address neighboring questions. Han et al. surveyed NoSQL databases [9]. Iqbal et al. carried out a systematic mapping study of database-security challenges that kept 100 of 4,827 retrieved documents and identified twenty challenges; in their Table 10, inadequate access control (63 studies), poor authentication (59), data leakage and privacy (59), and inadequate network, operating-system, and physical protection (58) were cited most often [10]. Alam et al. describe a layered architecture for cloud database management [11], and Kaur and Sharma discuss architectures for scalable cloud databases [12]. Official documentation describes product behavior: PostgreSQL documents isolation levels and serialization failures [13], SQL Server documents locking and row versioning [14], MySQL documents the InnoDB transaction model [15], and Oracle documents its concurrency model [16]. OWASP gives guidance on preventing SQL injection [17], and Brewer explains how Spanner relates to the CAP theorem [18]. Table I compares the earlier surveys with the present review by stated focus; it does not rank them.

TABLE I
PREVIOUS WORK VERSUS THE PRESENT REVIEW

Work	Stated focus	What this review adds
Dong et al. [3]	Cloud-native architectures: disaggregation, recovery, HTAP.	Ties architecture to transaction semantics, platform isolation and recovery, and a workflow.
Han et al. [9]	Survey of NoSQL data stores.	Concentrates on relational systems and how they adapt; no relational-versus-NoSQL ranking.
Iqbal et al. [10]	Systematic mapping of database-security challenges.	Connects security duties to architecture, recovery, and query construction.
Alam et al. [11]	Layered architecture for cloud database management.	Compares named systems (Aurora, Spanner, F1) with graded, attributed evidence.
Kaur and Sharma [12]	Architectures for scalable cloud databases.	Adds isolation behavior, recovery mechanisms, and a bounded quantitative reading.

III. RESEARCH METHODOLOGY AND EVIDENCE GRADING

A. Search and Selection

This is a focused narrative review, not a systematic review or meta-analysis. A general web search engine was used to locate scholarly papers, publisher and repository pages, and official documentation in November 2026. No database-by-database search was run and the number of records screened was not recorded, so the search cannot be reproduced exactly; this is a limitation of the review. For contrast, the mapping study in the previous section reports its counts at each screening stage [10]. Search terms included relational database, transaction isolation, cloud-native database survey, Aurora, Spanner, F1 distributed SQL, HTAP survey, database security mapping study, and each platform name with isolation levels and point-in-time recovery. Sources were retained if they addressed a review dimension and were a peer-reviewed paper, a widely cited system paper, or official vendor or community documentation. Blogs, tutorials, forum posts, and marketing pages were used only to locate primary sources and are not cited. Twenty-two sources were retained: twelve scholarly papers or surveys, one research technical note, and nine official documentation pages. Older papers are kept for foundational or architectural importance, not as descriptions of current product limits.

B. Evidence Grading

Each source receives a grade. Grade A means the passages supporting the specific claim were examined (full text or the relevant documentation section). Grade B means only the abstract or bibliographic record was examined. Conclusions in Section X inherit the lowest grade among their supporting sources, and platform statements are valid only for the documentation release named in the reference. Table II lists the grade and the boundary of each claim. No source graded B is presented as fully appraised. No database engine was installed and no workload was run, so no latency, throughput, cost, or availability figure was independently measured; the only calculation is the attributed one in Section VI.

(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 II
EVIDENCE-TO-CLAIM MATRIX WITH GRADES

Source	Grade	Claim and boundary
[1] Codd	B	Data independence and relations; not a modern implementation assessment.
[2] Gray	B	ACID concepts; not evidence about current engines.
[3] Dong et al.	A	Disaggregation taxonomy and layer-specific challenges; qualitative.
[4] Verbitski et al.	A	Redo offloading and historical I/O evidence; configuration-specific.
[5] Corbett et al.	B	External consistency and clock uncertainty; no reproduced evaluation.
[6] Bacon et al.	B	Evolution to distributed SQL; no new performance comparison.
[7] Shute et al.	B	F1 application and replication trade-off; developer-reported.
[8] Song et al.	A	Full text: monolithic versus hybrid taxonomy, freshness metrics, isolation level of hybrid systems.
[9] Han et al.	B	NoSQL survey context; no relational ranking drawn.
[10] Iqbal et al.	A	Full text: 100 of 4,827 documents kept; challenge counts in Table 10; a mapping study, not a product test.
[11] Alam et al.	B	Layered cloud-DBMS concept; not evaluated here.
[12] Kaur & Sharma	B	Scalable cloud-database architecture context; not evaluated here.
[13] PostgreSQL	A	PostgreSQL 18 isolation levels and retry duties.
[14] SQL Server	A	SQL Server isolation levels, row versioning, deadlock handling; page does not state one release.
[15] MySQL	A	MySQL 8.4 InnoDB isolation and consistent reads.
[16] Oracle	A	Oracle 21c ch. 11: default and offered isolation levels, multiversioning, ORA-08177, deadlocks.
[17] OWASP	A	Parameter binding and least privilege; guidance, not a trial.
[18] Brewer	B	CAP-compatible framing from the abstract; no availability estimate.
[19] PostgreSQL PITR	A	PostgreSQL 18 base backup, WAL archiving, point-in-time recovery.
[20] MySQL PITR	A	MySQL 8.4 binary-log point-in-time recovery.
[21] SQL Server recovery	A	SQL Server recovery models and point-in-time restore.
[22] Oracle RMAN	A	Oracle 21c backup guide ch. 1: RMAN, incremental backups, point-in-time recovery, Flashback.

IV. RELATIONAL FOUNDATIONS, TRANSACTIONS, AND PLATFORMS

A. Logical Organization Is Not Physical Placement

Primary keys, foreign keys, uniqueness rules, and check constraints let the database enforce relationships and business rules instead of leaving them to application code. Normalization reduces redundancy and update anomalies, but design must balance integrity, query needs, and maintainability, and controlled denormalization for read-heavy reporting needs explicit synchronization rules. Codd's data independence is the conceptual bridge to modern systems [1]. Disaggregating compute from storage changes the physical boundary, and distributing SQL and transactions adds coordination boundaries [3], [5], [6], but neither change alone shows that every workload scales equally well.

B. Isolation Anomalies and the Enrollment Case

A transaction is a logical unit of work that either completes as a whole or leaves no partial effect; atomicity concerns this all-or-nothing property, whereas isolation concerns what concurrent transactions may observe [2]. The standard anomalies are dirty reads, nonrepeatable reads, and phantom reads; PostgreSQL also documents serialization anomalies, in which committed outcomes match no serial order [13]. Two further problems matter for application design: a lost update, where one write silently overwrites another, and write skew, where two transactions each read a consistent snapshot, write different rows, and together break an invariant. Concurrency control may be pessimistic, blocking conflicting access with locks, or optimistic, using versions and aborting a transaction when a conflict is detected; the platforms in Table III mix both.

Consider an illustrative university enrollment with a capacity of one remaining seat. Two transactions each read that a seat is free and each insert an enrollment. Under snapshot-style isolation both may commit unless a constraint, a lock on the course row, an atomic conditional update, or Serializable execution prevents it. PostgreSQL states that Serializable requires outcomes equivalent to some serial order, may abort a transaction, and then requires the application to retry the whole transaction [13]. The example is illustrative, not an observed deployment, but it shows that correctness depends on engine guarantees and application control flow together.

C. Platform Comparison on Identical Criteria

MySQL, PostgreSQL, Oracle Database, and SQL Server are compared on the same criteria in Tables III and IV: default isolation level and levels offered, concurrency mechanism, and backup with point-in-time recovery. The comparison is at documentation level and does not rank the products. Versions are those named in the references: PostgreSQL 18, MySQL 8.4, and Oracle 21c for both concurrency and backup. The SQL Server pages do not state one release and are cited as current documentation, so they should be rechecked against the edition used.

(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 III
ISOLATION AND CONCURRENCY MECHANISMS

Platform	Default level; levels offered	Documented mechanism
MySQL (InnoDB) [15]	Repeatable Read; all four SQL levels offered.	Multiversioned consistent reads plus locking reads; Serializable turns plain reads into shared-lock reads when autocommit is off.
PostgreSQL [13]	Read Committed; Read Uncommitted behaves as Read Committed; Repeatable Read allows no phantoms.	Multiversion snapshots; Serializable detects dangerous dependencies and may abort a transaction.
Oracle [16]	Read Committed (default); Serializable and Read-only also offered.	Multiversion read consistency from undo data; a serializable transaction that changes a row committed after it began fails with ORA-08177; a deadlock rolls back one statement.
SQL Server [14]	Read Committed (locking); row-versioning variant via READ_COMMITTED_SNAPSHOT (off by default for user databases) and Snapshot.	Locking by default; row versioning when enabled; a deadlock victim is rolled back.

TABLE IV
BACKUP AND POINT-IN-TIME RECOVERY

Platform	Documented mechanism	Condition or limit
PostgreSQL [19]	Base backup plus continuous WAL archiving; restore to any time since the base backup.	Logical dumps cannot be part of continuous archiving; a continuous WAL sequence is needed.
MySQL [20]	Full backup plus replay of the binary log with mysqlbinlog.	Needs binary logging; enabled by default in 8.4.
Oracle [22]	RMAN physical backups (incremental supported) with archived redo logs; database point-in-time recovery with RECOVER DATABASE; Flashback Database as an alternative.	Logical Data Pump exports alone are not sufficient; flashback logging is off by default.
SQL Server [21]	Three recovery models; point-in-time restore through log backups under the full model.	Not supported under simple; restricted under bulk-logged.

Beyond these criteria, a fair evaluation specifies a common workload (transaction mix, read/write ratio, data size, concurrency, hardware, edition, isolation level, recovery objectives) and reports latency percentiles, throughput, abort and retry rates, recovery time, and cost. Replication behavior, SQL dialect, licensing, and migration effort also matter and are not compared here.

D. Query Processing and Physical Design

Query plans, statistics, indexes, and partitioning determine how many rows a statement touches and how long a transaction runs, which interacts with the locking and versioning mechanisms in Table III. This is standard database practice and is not drawn from the reviewed sources, so it is kept brief; indexes should follow real query patterns because they add cost to writes.

V. CLOUD-NATIVE AND DISTRIBUTED ARCHITECTURES

A. Disaggregation and Layer-Specific Consistency

Dong et al. distinguish compute–storage, compute–log–storage, and compute–buffer–storage architectures for cloud-native OLTP [3]. They change where durable logs, pages, and shared buffers live, and separating log and page storage adds flexibility but also synchronization delay between services [3]. The survey also separates storage-layer replica consistency from compute-layer update synchronization [3]: a storage protocol can decide whether replicas agree while the engine separately decides visibility, locking, and commit behavior. In this review's interpretation, disaggregation shifts rather than removes bottlenecks, because network bandwidth, tail latency, caching, and acknowledgement paths become design concerns. Fig. 1 shows the boundary.

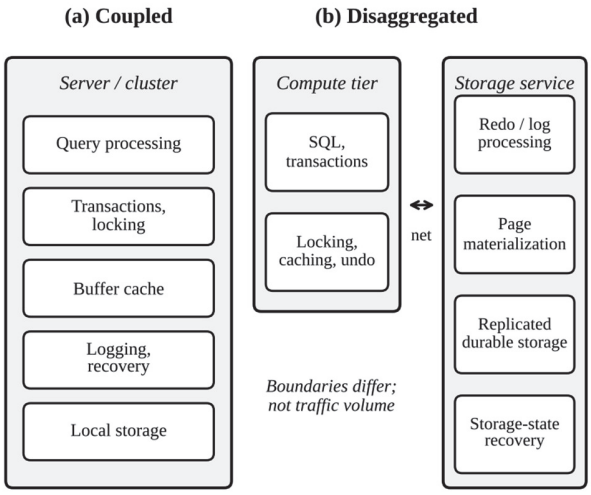


Fig. 1. Conceptual coupled and disaggregated deployments. Original schematic synthesized from the cloud-native survey [3] and the Aurora paper [4]; arrows show responsibility boundaries, not measured traffic.

B. Aurora: Redo Processing as a Storage Responsibility

Aurora keeps query processing, transaction management, locking, caching, and undo management in the engine, and delegates redo processing and durable storage to a distributed storage service [4]. Transmitting several representations of a write amplifies I/O in a replicated environment, so Aurora sends redo records and materializes pages at the storage tier

(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

[4]. The paper describes six storage copies across three Availability Zones with a four-of-six write quorum and a three-of-six read quorum in its published durability model [4]; these are historical design parameters, not a statement of every current configuration. The design redistributes recovery work across layers; it does not remove recovery or transaction logic [4].

C. Spanner and F1: Geographic Coordination and SQL

Spanner combines synchronous replication with externally consistent transactions [5], and its SQL paper addresses relational queries over distributed infrastructure [6]. F1, built for Google's AdWords business on Spanner-backed storage, recognizes higher commit latency from cross-data-center replication and mitigates it through hierarchical schemas and application design [7]. Brewer explains why strong consistency and high availability do not invalidate the CAP theorem [18]: availability claims need defined failure assumptions. Not every application needs global transactions; some can meet their requirements with regional databases, asynchronous replicas, or partitioning.

The sources describe designs; they do not supply a common comparison of node failure, zone failure, network partition, replica lag, and recovery events. A fair comparison would fix these scenarios and state, for each architecture, what is preserved: atomicity, isolation, and availability. This remains open (Section XI). Table V lists engineering concerns, not scores.

TABLE V
ARCHITECTURAL CAPABILITIES AND EVALUATION BOUNDARIES

Pattern	Supported interpretation	What remains to test
Compute–storage separation [3], [4]	Compute and storage can be managed separately.	Cache misses, network saturation, engine bottlenecks.
Separate log/page services [3]	Durability and page-serving paths can be specialized.	Lag and synchronization delay between services.
Geographic relational execution [5]–[7]	Transactions and SQL can span distributed infrastructure.	Commit latency, locality, failure and partition behavior.
HTAP [8]	Transactions and analytics can share a system.	Interference, synchronization, and data freshness.

VI. SECONDARY ANALYSIS OF PUBLISHED I/O EVIDENCE

This section reanalyzes one published experiment; the authors did not run it. Table 1 of the Aurora paper reports a SysBench write-only test with a 100-GB data set, a 30-minute run, and engines on r3.8xlarge instances, comparing a synchronous mirrored MySQL configuration with Aurora with replicas [4]. The reported database-node network I/Os per transaction were 7.4 and 0.95, respectively [4], redrawn in Fig. 2. From the rounded values, the ratio is $7.4/0.95 = 7.79$ and the relative reduction is $(1 - 0.95/7.4) \times 100 = 87.16\%$. These are secondary calculations; the paper itself describes a 7.7-fold reduction, so rounding matters [4].

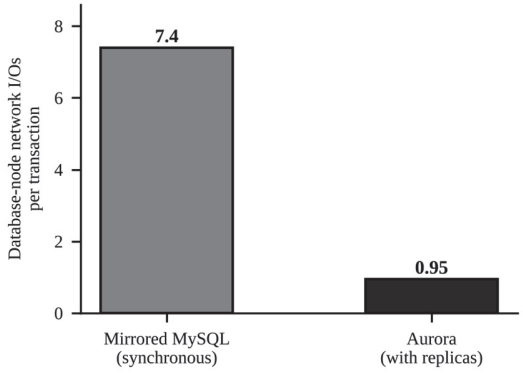


Fig. 2. Redrawn reported database-node network I/Os per transaction from Table 1 of the Aurora paper [4]. Historical write-only configuration; not an experiment conducted for this review.

The reported transaction totals were 780,000 and 27,378,000 [4], a ratio of about 35.1, but this is not a universal speedup: the architectures and replication paths differ, the source notes exclusions in its I/O accounting, and the experiment does not isolate redo offloading as the only cause. It is one historical experiment with no second independent study, and it says nothing about read-heavy, mixed, or other write workloads. The defensible conclusion is narrow: in this configuration, a design built to reduce database-node network work reported much lower I/O per transaction. It supports investigating network amplification, not claims of current product leadership or cost superiority.

VII. MODERN APPLICATIONS AND SOCIETAL IMPACT

A. Hybrid Workloads and Deployment Forms

OLTP emphasizes short, concurrent operations with predictable latency, whereas analytics emphasizes scans and aggregations. HTAP aims to serve both, and freshness should be stated as a requirement: an inventory report may tolerate delay, while a purchase decision may need an authoritative check [3], [8]. Song et al. report that most hybrid HTAP systems they reviewed provide snapshot isolation even when transaction processing promises stronger guarantees, so the isolation seen by analytical reads should be checked separately [8]. One possible design keeps authoritative updates in a transactional RDBMS and moves analytics to replicas or specialized services; evaluation should measure both analytical time and interference with transactions. A cloud-hosted database may run a conventional engine on virtual machines, whereas a cloud-native one reorganizes storage, logging, recovery, and compute placement [3]; the choice depends on latency, data residency, cost, and operational capability.

B. Sector Requirements and Societal Impact

Relational systems hold records on which people rely. In healthcare, referential integrity and controlled access reduce the risk of conflicting or duplicated patient records, which

(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

can matter for patient safety. In banking, atomic updates and traceable records protect balances. In education and government, correct enrollment and service records affect individuals' access to services. In e-commerce, consistent order and inventory data decide whether transactions complete correctly. The sectors differ in which requirement dominates, so the same database choice can be right for one and wrong for another. These are design scenarios, not case studies or prevalence statistics, and choosing a product does not by itself guarantee regulatory compliance.

VIII. SECURITY, RECOVERY, AND EVALUATION WORKFLOW

A. Threat Model and Query Safety

A minimal threat model lists assets (records, credentials, backups), actors (external attackers, compromised application accounts, careless or malicious administrators), attack surfaces (application queries, network connections, replicas and backups, administrative interfaces), and objectives (confidentiality, integrity, availability, accountability). Mapping each control to this list shows what it does and does not cover. This framing is the authors' own. The mapping study found that 52 of its 100 studies addressed both internal and external attackers and that inadequate access control and poor authentication were among the most cited challenges [10], which supports reviewing identities and privileges in any evaluation.

OWASP identifies separation of SQL structure from bound data values as the central defense against SQL injection; dynamic identifiers need allow-lists, least privilege limits damage but does not prevent injection, and stored procedures are not safe if they build dynamic SQL by concatenation [17]. These points do not depend on whether the engine runs locally or as a managed service, but they cover only query construction, not backups or administrative access. Other commonly recommended controls, such as encrypted connections, secrets kept outside source code, audit logging, and network segmentation, are general good practice and are not drawn from the reviewed sources.

B. Recovery Responsibility

Recovery design should define the recovery point objective (RPO) and recovery time objective (RTO) and then check that the platform's mechanism can meet them. Table IV shows that all four platforms support point-in-time recovery only under stated conditions: archived WAL in PostgreSQL, binary logging in MySQL, ARCHIVELOG mode in Oracle, and the full recovery model in SQL Server. Replication improves availability but is not a backup, because accidental deletion or corruption can be replicated faithfully, and a backup is incomplete until restoration is tested. Storage-layer designs such as Aurora's redistribute recovery work across layers [4], so responsibility for each step should be documented. Failover, failback, replication lag, and managed-service responsibility boundaries are not compared here.

C. Requirements-First Workflow and Worked Example

Fig. 3 shows five stages: specify invariants, transaction semantics, and recovery objectives; characterize contention, volume, locality, and freshness; shortlist deployment candidates using Tables III–V; pilot with failure, restore, and security scenarios; and compare observed correctness, cost, and operational demand. When requirements conflict, invariants and recovery objectives take priority over cost and convenience.

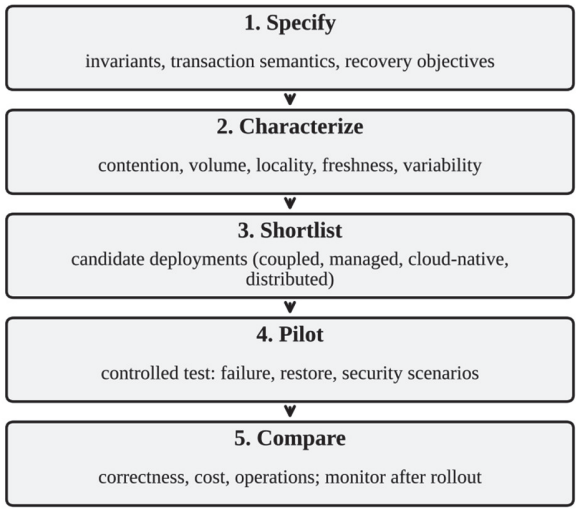


Fig. 3. Proposed requirements-first evaluation workflow. Conceptual synthesis of the reviewed sources; not an experimentally validated selection algorithm.

Worked example (hypothetical). A university must accept enrollments with a hard seat limit. Specify: the capacity invariant, retry of failed transactions, and, for illustration, an RPO of 15 minutes and an RTO of one hour. Characterize: a short registration peak with many writers on few rows, mostly regional users, modest data. Shortlist: a conventional engine with locking or Serializable isolation; a managed single-region service; a distributed SQL system. Illustrative decision rule, not a validated threshold: choose distributed SQL only if writes from several regions must commit with strong consistency; otherwise prefer the first two. Pilot: run many concurrent attempts on the last seat and confirm the limit holds; terminate the primary during load and time the failover; restore to a chosen time and compare data. Compare: invariant violations (target zero), abort and retry rates, restore time against the RTO, and cost. The workflow has not been shown to improve selection, and two independent evaluators were not compared.

IX. INNOVATION AND EMERGING DIRECTIONS

- **Disaggregated, log-centric storage:** moving redo processing into a storage service changes the I/O and recovery path [3], [4]; whether it helps other workloads needs testing.

(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

- **Distributed SQL:** externally consistent, geographically replicated execution trades commit latency for coordination [5]–[7].
- **AI-assisted management:** the cloud-native survey lists machine-learning techniques, serverless processing, and multi-writer designs as research directions [3]. They are not analyzed here; automation should be judged on reversibility and recovery behavior.

X. FINDINGS AND RECOMMENDATIONS

A. Answers to the Research Questions

Table VI answers each question separately. "Established" means stated by the cited source; "interpretation" means the authors' synthesis. Each answer carries the lowest grade of its sources.

TABLE VI
RESEARCH QUESTIONS, ANSWERS, AND EVIDENCE BASIS

Item	Answer	Basis and grade
RQ1	Logical relational organization coexists with different placements of compute, logs, pages, and transaction execution; modernization changes responsibility boundaries [3]–[7].	Established in system papers; the synthesis is interpretation. Grade B.
RQ2	Default isolation, retry duties, and point-in-time recovery conditions differ by platform (Tables III–IV) [13]–[16]; query safety is independent of hosting [17].	Established in documentation for the named releases. Grade A.
RQ3	Requirements can become testable criteria through the five-stage workflow and its worked example.	Interpretation; not validated.
Scale	Aurora reported lower network I/O in one historical configuration only [4].	Established, single experiment. Grade A.
Security	Access-control and authentication weaknesses recur in the literature [10].	Established in full text. Grade A.

B. Recommendations

- **Choose by requirements:** select the platform and deployment form from invariants, workload, locality, and recovery needs, not popularity.
- **Match isolation to invariants:** justify the isolation level and retry whole transactions after serialization or deadlock failures [13].
- **Design the schema deliberately:** normalize to remove harmful redundancy; denormalize only with explicit synchronization rules.
- **Prioritize query safety and access control:** use parameter binding and least privilege [17], and review identities and privileges [10].
- **Meet recovery objectives:** enable the log mechanism your platform needs for point-in-time recovery (Table IV), keep backups independent of replication, and test restoration.
- **Evaluate before adopting:** pilot under a stated workload, including failure and restore tests, before choosing a platform or cloud-native service.

These recommendations are the authors' proposals derived from the reviewed evidence, not experimentally validated rules.

XI. RESEARCH GAPS, LIMITATIONS, AND FUTURE DIRECTIONS

A. Research Gaps

- **Performance–security trade-off:** encryption, fine-grained access control, and auditing can add overhead, but the reviewed sources were not used to quantify it for distributed relational systems.
- **Failure behavior:** no common failure-scenario comparison of Aurora, Spanner, and F1 was found in the reviewed sources; claims must state failure assumptions [18].
- **Selection and tuning:** HTAP engines and managed services enlarge the configuration space [8].

B. Limitations and Their Effect on the Conclusions

The search was not reproducible in the systematic sense (no record counts), so the evidence set may omit relevant work. Nine of the twenty-two sources are graded B, which limits the strength of conclusions that rest on them, notably RQ1. Platform statements apply only to the named releases, and the SQL Server pages do not state one. System papers reflect developer-reported designs and publication-era environments, and the Aurora reanalysis rests on one experiment with rounded values. The workflow and recommendations are unvalidated, and the comparison omits encryption, auditing, replication, and failover across platforms. Conclusions are therefore qualitative and conditional.

C. Future Directions

Reproducible follow-ups could compare mixed workloads on matched durability and isolation settings, measure the cost of security controls, compare failure scenarios across architectures, and test whether the workflow improves selection with independent evaluators. These need experiments beyond this review.

XII. CONCLUSION

Established in the reviewed sources: relational organization coexists with several physical architectures; Aurora moves redo processing to storage and Spanner and F1 distribute transactions and SQL; the four platforms differ in default isolation, concurrency mechanism, and point-in-time recovery conditions; and authorization and query construction are recurring security concerns. The authors' interpretation is that modernization shifts responsibility boundaries instead of removing duties, and that platform choice should follow requirements tested in a pilot. Because nine sources were graded B, the search was not reproducible, and the workflow is unvalidated, broader comparative claims need full-text appraisal and independent measurement.

(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

ACKNOWLEDGMENT

The authors thank their course faculty for guidance during the four phases of this term paper.

DECLARATION OF AI USE

AI-based tools (large language model assistants) were used to help find and summarize sources, draft and organize the text, prepare the tables and figures, and perform the secondary calculation in Section VI. The authors are responsible for the accuracy of all content, calculations, citations, and conclusions, and have reviewed and verified them against the original sources.

REFERENCES

[1] E. F. Codd, "A relational model of data for large shared data banks," Commun. ACM, vol. 13, no. 6, pp. 377–387, 1970, doi: 10.1145/362384.362685.

[2] J. Gray, "The transaction concept: Virtues and limitations," in Proc. 7th Int. Conf. Very Large Data Bases (VLDB), 1981, pp. 144–154.

[3] H. Dong, C. Zhang, G. Li, and H. Zhang, "Cloud-native databases: A survey," IEEE Trans. Knowl. Data Eng., vol. 36, no. 12, pp. 7772–7791, Dec. 2024, doi: 10.1109/TKDE.2024.3397508.

[4] A. Verbitski et al., "Amazon Aurora: Design considerations for high throughput cloud-native relational databases," in Proc. ACM Int. Conf. Management of Data (SIGMOD), 2017, pp. 1041–1052, doi: 10.1145/3035918.3056101.

[5] J. C. Corbett et al., "Spanner: Google's globally-distributed database," in Proc. 10th USENIX Symp. Operating Systems Design and Implementation (OSDI), 2012, pp. 251–264.

[6] D. F. Bacon et al., "Spanner: Becoming a SQL system," in Proc. ACM Int. Conf. Management of Data (SIGMOD), 2017, pp. 331–343, doi: 10.1145/3035918.3056103.

[7] J. Shute et al., "F1: A distributed SQL database that scales," Proc. VLDB Endow., vol. 6, no. 11, pp. 1068–1079, 2013, doi: 10.14778/2536222.2536232.

[8] H. Song, W. Zhou, H. Cui, X. Peng, and F. Li, "A survey on hybrid transactional and analytical processing," VLDB J., vol. 33, no. 5, pp. 1485–1515, Sep. 2024, doi: 10.1007/s00778-024-00858-9.

[9] J. Han, H. E, G. Le, and J. Du, "Survey on NoSQL database," in Proc. 6th Int. Conf. Pervasive Computing and Applications (ICPCA), 2011, pp. 363–366, doi: 10.1109/ICPCA.2011.6106531.

[10] A. Iqbal et al., "Advancing database security: A comprehensive systematic mapping study of potential challenges," Wireless Netw., vol. 30, no. 7, pp. 6399–6426, 2024, doi: 10.1007/s11276-023-03436-z.

[11] B. Alam, M. N. Doja, M. Alam, and S. Mongia, "5-layered architecture of cloud database management system," AASRI Procedia, vol. 5, pp. 194–199, 2013, doi: 10.1016/j.aasri.2013.10.078.

[12] P. D. Kaur and G. Sharma, "Architectures for scalable databases in cloud – and application specifications," Procedia Comput. Sci., vol. 58, pp. 622–634, 2015, doi: 10.1016/j.procs.2015.08.081.

[13] PostgreSQL Global Development Group, "Transaction isolation," PostgreSQL 18 Documentation, sec. 13.2. [Online]. Available: https://www.postgresql.org/docs/18/transaction-iso.html. Accessed: Nov. 2026.

[14] Microsoft, "Transaction locking and row versioning guide," SQL Server Documentation (current release on Microsoft Learn). [Online]. Available: https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide. Accessed: Nov. 2026.

[15] Oracle Corporation, "Transaction isolation levels," MySQL 8.4 Reference Manual, sec. 17.7.2.1. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/innodb-transaction-isolation-levels.html. Accessed: Nov. 2026.

[16] Oracle Corporation, "Data concurrency and consistency," Oracle Database Concepts, 21c, ch. 11. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/data-concurrency-and-consistency.html. Accessed: Nov. 2026.

[17] OWASP Foundation, "SQL injection prevention cheat sheet," OWASP Cheat Sheet Series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. Accessed: Nov. 2026.

[18] E. Brewer, "Spanner, TrueTime and the CAP theorem," Google Research, 2017. [Online]. Available: https://research.google/pubs/spanner-truetime-and-the-cap-theorem/. Accessed: Nov. 2026.

[19] PostgreSQL Global Development Group, "Continuous archiving and point-in-time recovery (PITR)," PostgreSQL 18 Documentation, sec. 25.3. [Online]. Available: https://www.postgresql.org/docs/18/continuous-archiving.html. Accessed: Nov. 2026.

[20] Oracle Corporation, "Point-in-time recovery using binary log," MySQL 8.4 Reference Manual, sec. 9.5.1. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/point-in-time-recovery-binlog.html. Accessed: Nov. 2026.

[21] Microsoft, "Recovery models (SQL Server)," SQL Server Documentation (current release on Microsoft Learn). [Online]. Available: https://learn.microsoft.com/sql/relational-databases/backup-restore/recovery-models-sql-server. Accessed: Nov. 2026.

[22] Oracle Corporation, "Introduction to backup and recovery," Oracle Database Backup and Recovery User's Guide, 21c, ch. 1. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/21/bradv/introduction-backup-recovery.html. Accessed: Nov. 2026.

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