International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 13 Issue 11 ǁ November 2025 ǁ PP. 73-98

Databases for Intelligent Systems: A Review of Relational, NoSQL, Vector, and Blockchain Datastores for Scalable AI Applications

AL. Sayeth Saabith[1], T. Vinothraj[2], MMM.Fareez[3]
[*1] *Centre for Information Communication Technology, Faculty of Science, Eastern University, Sri Lanka*
[2] *Centre for Information Communication Technology, Faculty of Science, Eastern University, Sri Lanka*
[3] *Finance Department Eastern University, Sri Lanka*

Abstract
The rapid implementation of intelligent systems, including large language model (LLM) assistants and fraud-detection engines, as well as autonomous monitoring platforms and auditable systems controlling supply chains, has demonstrated the shortcomings of treating data management as a one-database issue. The current AI processes entail transactional integrity, high-throughput telemetry ingestion, sub-millisecond inference-time feature access, semantic retrieval over unstructured data, relationship reasoning, and verifiable provenance. None of the traditional datastores can meet all these requirements. Rather, modern architectures are becoming polyglot in more ways: a federation of dedicated databases, each with its own performance, consistency, or trust benchmarks. It trails the key categories of databases that now underlie scalable AI applications: (i) relational/SQL databases, which still serve as system of record to high-stakes transactional state on ACID guarantees; (ii) columnar and time-series databases, which support large-scale analytics and real-time observability and time pattern detection; (iii) in-memory databases, which project low-latency contextual features to inference-time decision loops; (iv) document and graph databases, which capture flexible semi-structured application context and relationship-centric reasoning, In each category we examine the data model, execution model, scalability strategy and canonical use cases of AI. We also point out the trend of convergence today: mainstream engines like PostgreSQL and MongoDB are acquiring features previously considered external-JSON documents, time-series ingestion, and vector search, and vector and ledger systems are incorporating metadata filtering, access control, and governance. Lastly, we define open research problems as (a) keeping semantic indexes up to date and with transactional guarantees, (b) providing milliseconds response-time at sustainable cost, (c) providing end-to-end explainability and cryptographic provenance of regulated decisions, and (d) enhancing energy efficiency in both memory-intensive and GPU-accelerated retrieval stacks. We believe that intelligent systems of the future will not be characterized merely by their models, but by their capacity to coordinate heterogeneous data infrastructures which are responsive, auditable and accountable.

Keywords: *Polyglot persistence; Vector databases; RAG; Knowledge graphs; Ledger provenance; Time-series analytics; In-memory serving; Columnar warehouses.*

Date of Submission: 01-11-2025 Date of acceptance: 10-11-2025

I. INTRODUCTION

Instead of being driven by a single general-purpose database, intelligent systems such as large language model (LLM) assistants, recommender engines, autonomous monitoring pipelines, and supply chain traceability platforms are now driving them. Rather, they rely on dedicated datastores with specific objectives: high assurance of transactional integrity, minimal response time, scalable analytics, semantic similarity search, or evidence of tampering [1]-[3]. The increasing data volume, model size, and real-time decision-making demands have revealed the constraints of the classical one-size-fits-all storage and have led to the development of heterogeneous database structures [4].

In the past, Online Transaction Processing (OLTP) was based on relational/SQL databases, including PostgreSQL, MySQL, Oracle, SQL Server and MariaDB. These systems ensure the semantics of ACID (Atomicity, Consistency, Isolation, Durability), well-structured schemas and expressive declarative queries (SQL). They remain the ideal option for financial transactions, inventory management, academic records, and other fields where referential integrity and accuracy are highly valued [5]. Nevertheless, traditional relational engines are horizontal (massively distributed clusters) rather than vertical (larger single machines) and do not inherently support unstructured documents, high-velocity telemetry, or multi-hop relationship reasoning [6].

A family of so-called NoSQL databases came to provide scalability and flexibility requirements. These systems do not implement one fixed rigid schema, but run at specific workload shapes:

i. Columnar/analytical databases (ClickHouse, Amazon Redshift, and HBase) store data not in rows but in columns. This simplifies the process of making large aggregations and business intelligence queries at the data warehouse level [7].
ii. Time-series databases (e.g., InfluxDB, TimescaleDB, QuestDB) accept high-frequency timestamped metrics from IoT sensors, servers, and financial feeds, with very high write throughput and built-in retention and downsampling [8].
iii. Hot data (e.g., Redis, SAP HANA, Oracle TimesTen, Apache Ignite) is stored in RAM in-memory databases to provide sub-millisecond read/write performance for latency-sensitive decision loops, such as fraud detection or ad bidding [9].
iv. Semi-structured JSON-like documents (e.g., MongoDB, Couchbase, Firestore, CouchDB) are stored in document databases rather than normalized relational tables and can therefore evolve their schemas quickly to meet the needs of modern web/mobile applications [10].
v. Graph databases (e.g., Neo4j, Amazon Neptune, Azure Cosmos DB graph API, JanusGraph) make relationships (edges) first-class citizens, allowing them to perform multi-hop reasoning to solve problems such as detecting a fraud-ring, recommender systems, knowledge graphs, cybersecurity link analysis, and path routing [11].

More recently, there are two other types of data stores that have become indispensable to AI-driven systems:

i. Vector databases. Systems like Pinecone, Weaviate, Milvus, Qdrant, and Chroma are optimised for high-dimensional embeddings generated by deep neural networks and large language models. These engines also employ approximate nearest neighbour (ANN) search in the vector space rather than exact key lookup to find semantically similar items [12]. The vector search has become a fundamental component of Retrieval-Augmented Generation (RAG), recommendation systems, multimodal search, and the "memory" elements of AI assistants. The performance criteria in this context are also specific: queries should execute with low latency, often using GPU-accelerated indexes, on millions or billions of embedding vectors, while still filtering metadata and updating their state dynamically [13]. Using vector databases enables intelligent systems to retain meaning rather than just exact keywords [14], [15].

ii. Blockchain databases or ledger-style databases, such as BigchainDB and CovenantSQL, introduce an append-only, tamper-evident model where all write operations can be audited and verified independently [14]. Ledger-oriented databases enable multiple parties that distrust each other to maintain a common source of truth, which in traditional replicated databases relies on a single operator model of trust. This has been particularly of interest for provenance tracking, compliance reporting, financial traceability, and fair-trade or sustainability validation in sectors such as agriculture and commodity logistics [15]. Verifiable lineage is increasingly a necessity rather than an option for AI systems, especially where decisions require explanation or justification to customers, e.g., why a particular batch of palm oil was flagged as low quality. [16],[17],[18],[19].

The outcome is a type of ecosystem in which no database type is competing with others; instead, they are layers in a much larger pipe. One smart application could:

i. ACID guarantees that store orders, accounts, and billing events are stored in a relational SQL database.
ii. store high-frequency sensor or telemetry data in a time-series database.
iii. store user/session state and feature cache in an in-memory store to enable real-time decisions.
iv. store semantically unstructured data, such as numbers, in a vector database—this could include indexing unstructured text, manuals, chat transcripts, or research papers.
v. record cross-organization assets or quality certifications in a blockchain-based ledger.

The architectural style is sometimes called polyglot persistence: instead of making a single database handle all tasks poorly, we deliberately employ several specialised stores, each tailored to a specific subsystem [17]. Polyglot persistence has become the new default model for hyper-scale AI applications, especially for analytics, reasoning, and verifiability [20], [21], [22].

Nevertheless, this specialization introduces a new complexity. Categorical boundaries are becoming less clear. The document-based JSON (JSONB), time-series extensions, and a vector index (pgvector) are now used as mature relational engines, with PostgreSQL consolidating multiple functions into a single platform [18]. Multi-document ACID transactions and vector search are now supported in document stores, such as MongoDB. Metadata filtering and hybrid keyword-based semantic retrieval are being added to vector databases, making them

more like document databases in some ways. SQL analytics engines are layered on top of time-series engines. At the same time, blockchain-like systems are exploring greater throughput and partial privacy to move beyond cryptocurrency use cases [19]. The market is converging, yet the design trade-offs (such as consistency, latency, cost, trust, and energy consumption) remain highly varied [23], [24], [25].

This is a review with three purposes:

i. Taxonomy: We introduce a hierarchical taxonomy of modern types of databases relevant in the context of intelligent systems: Relational SQL, Columnar/Analytical, Time-Series, In-Memory, Document, Graph, Vector, Object-Oriented, and Blockchain/Ledger. The data model, internal design assumptions, and main optimisation objectives for each category focus on strong consistency, low latency, semantic similarity, and auditability.

ii. Comparative analysis: Each category (e.g., PostgreSQL/MySQL, ClickHouse/Redshift, InfluxDB/TimescaleDB, Redis/HANA, MongoDB/Firestore, Neo4j/Neptune, Pinecone/Milvus/Qdrant, db4o/ZODB, BigchainDB/CovenantSQL) is examined and representative systems discussed, along with their strengths, weaknesses, and typical applications in production AI stacks.

iii. Trends and research challenges: We also highlight convergence patterns, such as polyglot persistence, multimodal data infrastructure, and present unresolved research issues: (i) keeping vector indexes current and consistent with source data changes; (ii) achieving semantic retrieval in less than a millisecond at sustainable costs; (iii) improving explainability and auditability of AI-driven decisions; and (iv) balancing trust (blockchain-style immutability) with the performance limitations of real-time systems [26], [27], [28], [29].

The rest of this paper is structured as follows.

Part 2 explores fundamental concepts, including OLTP versus OLAP, CAP trade-offs, and data models. The third section focuses on relational/SQL databases. Sections 4 to 8 cover important NoSQL families, including columnar, time-series, in-memory, document, and graph databases. Section 9 is dedicated to AI retrieval using vector databases. Section 10 discusses the issues surrounding object-oriented databases. Part 11 examines blockchain and ledger-based databases for verifiable data sharing. Part 12 explores convergence and hybrid architecture. Section 13 highlights current open research directions, and Section 14 offers a conclusion.

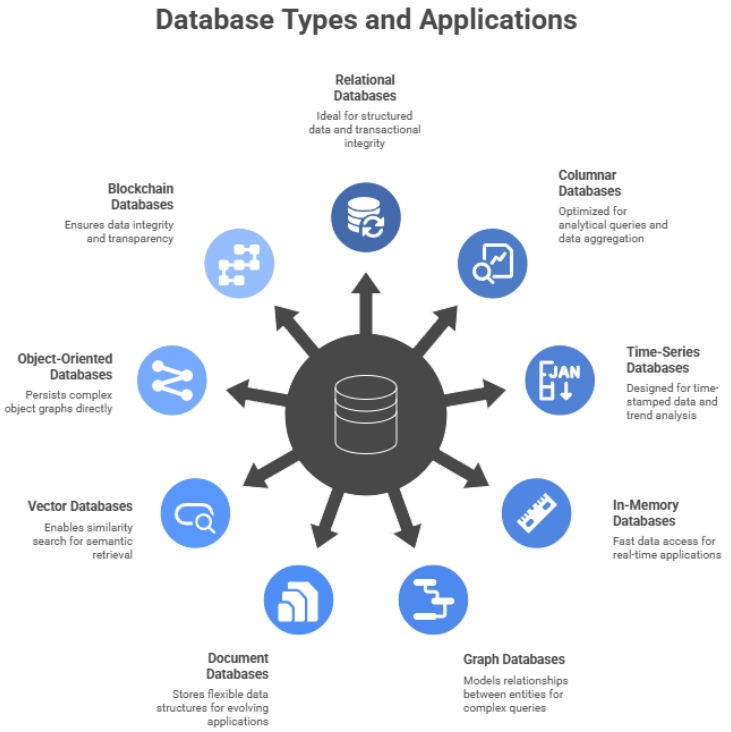

Figure 1: Types of Databases

Figure 1. High-level taxonomy of modern databases used in intelligent systems. The categories include: (i) Relational / SQL databases (e.g., PostgreSQL, MySQL, SQL Server); (ii) Columnar/analytical databases for large-scale OLAP workloads (e.g., ClickHouse, Redshift); (iii) Time-series databases for high-ingest telemetry (e.g., InfluxDB, TimescaleDB); (iv) In-memory databases for sub-millisecond access (e.g., Redis, SAP HANA); (v) Graph databases for relationship-centric queries (e.g., Neo4j, Amazon Neptune); (vi) Document databases for

flexible JSON-like records (e.g., MongoDB, Couchbase, Firestore); (vii) Vector databases for semantic similarity search and Retrieval-Augmented Generation (e.g., Pinecone, Milvus, Qdrant, Chroma); (viii) Object-oriented databases for direct persistence of complex application objects (e.g., db4o, ZODB); and (ix) Blockchain/ledger databases for tamper-evident, auditable, multi-party data sharing (e.g., BigchainDB, CovenantSQL).

Each class targets different performance and governance needs — transactional integrity, large-scale analytics, ultra-low latency, relationship reasoning, semantic retrieval, or cross-organizational trust — which explains why modern AI systems increasingly adopt polyglot persistence rather than a single monolithic datastore.

1. BACKGROUND

Modern intelligent systems seldom deal with a database in the singular. Rather, they engage with several layers of storage and retrieval, each optimized for a set of access patterns, access latency budgets and trust constraints. To make later comparisons of these systems meaningfully, we first define four underlying concepts: (i) OLTP vs. OLAP, (ii) the CAP theorem and consistency/availability trade-offs, (iii) data models with which the major families of databases represent data, and (iv) indexing and access structures that enable queries to scale.

1.1. OLTP VS OLAP

Online Transaction Processing (OLTP) is used for workloads that involve a very large number of small, accurate operations (e.g., inserting an order, updating an account balance, or checking a student's enrollment). OLTP load distributions focus on low-latency writes, high fidelity, and the isolation of parallel transactions. Relational databases (e.g., PostgreSQL, MySQL, SQL Server, Oracle, MariaDB) were originally intended to be used in OLTP and usually provide ACID properties:

- **Atomicity:** every transaction is either all or none.
- **Consistency:** each transaction changes the database to some state of consistency, based on a specific set of rules and constraints.
- **Isolation:** parallel transaction acts as though they ran one transaction at a time.
- **Durability:** once committed, data is not lost during crashes and restarts [2].

This is necessary in areas such as banking, payroll, inventory, or student information systems, where it is unacceptable to lose or corrupt a single row[30],[31].

Online Analytical Processing (OLAP) works with large, exploratory, and scan-heavy queries: "find the sum of sales per region in the last 6 months," or find the percentiles of hourly CPU usage on 10,000 servers. OLAP workloads focus on read-intensive aggregation of very large datasets (which may be historical). The queries can withstand a little more per-query latency (in the seconds range rather than the milliseconds range), requiring both high throughput and high compression. Columnar/analytical databases (such as ClickHouse, Amazon Redshift, MonetDB, and Apache HBase in some modes) store data by column rather than by row and thus can only read the columns that are relevant and perform efficient, vectorized operations [5].

Practically, both must usually be used in intelligent applications. To illustrate, an e-commerce/fintech system will not only (i) require payment processing in an OLTP engine so that it can keep balance accurate, but also (ii) it will continuously analyze fraud patterns and spending trends in an OLAP/columnar warehouse. This division is among the primary causes that polyglot persistence has become common[32],[33].

1.2. The CAP Theorem and Distributed Trade-offs

Once databases are scaled horizontally over more than one machine or data center, they encounter the "CAP theorem" which informally says that a distributed system can never guarantee simultaneous availability of:

- C - Strong Consistency: all read access the latest write.
- A - High Availability: all requests are given non-error responses.
- P - Partition Tolerance: the system is operational even when there are dropped/stalled network messages [4].

Partitions do allow you to choose at most two of these guarantees. Traditional relational databases that operate on one node (or closely synchronized cluster) are more likely to put Consistency over Availability: It is better to block than deliver stale data/incorrect data. Most NoSQL databases, though not all, particularly the earlier key-value/document/column-family databases, intentionally compromised strong consistency to ensure high availability and horizontal scalability over commodity hardware [5].

This consistency/availability dial appears everywhere in the present-day AI infrastructure.

- An AI assistant or recommendation engine can accept metadata that is slightly stale in the case where it does not cause the service to go unavailable - this is more towards availability.

- The ledger of financial transactions or provenance of products (e.g. blockchain-like database of supply chain integrity) can not accept incompatible truths - it is more towards consistency and immutability.

The trade-offs between CAP and document stores, graph databases, and vector databases will be revisited, since each category involves some of the following decisions: correctness vs. scale[34],[35].

1.3. Data Models

The various types of databases have different data models i.e.; there are various representations of what a record is and how do records relate. These models are not cosmetics, they directly define what queries are easy, what queries are expensive and how you scale.

1.3.1. Relational (tabular) model

The information is stored in tables (relations) where the rows (tuples) and columns (attributes) are represented. Tables are connected to each other using keys (primary keys, foreign keys). This query language is SQL that has joins, filters, aggregations, and transactions. This model prevails in accounting, billing, HR, finance, logistics - anything with well-defined and well-managed entities (Customer, Invoice, Product, Shipment) [5].

1.3.2. Document / key-value model

Rather than distributing data in a large number of normalized tables, every document contains all the information pertinent to a single entity (usually in the form of JSON or BSON). Dissimilar documents within the identical collection may not take the same form. This paradigm is more development-agile: front-end/mobile groups develop with rapid iterations without ALTER TABLE on a weekly basis. The same case applies to MongoDB, Couchbase and Firestore[6].

1.3.3. Columnar model

The information remains logically tabular, only physically being stored by column. This is to say that asking something like calculate AVG(temperature) by hour in the past 30 days the engine will only read the temperature and timestamp columns in compressed format not the entire rows [3]. They include ClickHouse and Redshift[7].

1.3.4. Time-series model

Time-series databases are databases that use timestamped measurements as the first-class primitive. The schema is also optimized to add new points, expire old points and to do range queries, such as, last 15 minutes, 7-day moving average or 95 th percentile latency in July. These include influxDB, TimescaleDB, QuestDB, etc[8].

1.3.5. Graph model

The graph model stores the information in the form of nodes (vertices) and edges (relationships). Attributes may be assigned to each node and edge. The query pattern does not focus on finding the rows with x=y but traversing the relationships, such as finding all the suppliers of this supplier with two hops and ship to this region, or finding rings of accounts transferring money to each other on cycles. They can be run successfully on graph databases such as Neo4j and Amazon Neptune [6],[9].

1.3.6. Vector/embedding model

The record is mostly a high-dimensional numeric vector created by an ML model (e.g. a 768-dimensional embedding of a transformer) in vector databases. Distance measures (cosine similarity, Euclidean distance, inner product) are used to define similarity, but not equality. Find nearest neighbors of this new vector is a query that can be translated to find semantically similar text/images/code/etc. Pinecone, Milvus, Weaviate, Qdrant, and Chroma are optimized to find nearest neighbors at scale [7],[10].

1.3.7. Object-oriented model

Object-oriented databases Object-oriented databases (e.g. db4o, ZODB) store application objects, such as nested objects and inheritance, directly, without requiring a manual object-relational mapping. This is appealing in embedded system, simulation or complex CAD/engineering system where objects are rich and highly interconnected [8],[11].

1.3.8. Ledger / blockchain model

Ledger-style databases store the information in the form of an append-only list of signed, verifiable transactions replicated among various parties. Focus is placed on immutability, auditability, and shared trust as opposed to raw throughput. Examples are bigchainDB and CovenantSQL. This model is getting more applicable to compliance, provenance and sharing data across organizations (e.g. supply chain, sustainability certification)[16].

These models are no longer separated silos. Contemporary engines are hybridized (such as PostgreSQL has now the ability to work with JSON documents, time-series extensions, geospatial indexes and vector similarity

search via extensions). However the above models nevertheless characterize the native mindset that arises based on each type of database that is developed[36].

1.4. Indexing and Access Paths

Raw scanning is too slow at large scale. The databases make use of special index structures to respond to queries effectively. The kind of index applied usually talks about the principal purpose of the database.

B-tree / LSM-tree indexes: Relational and most key-value/document stores provide the use of B-trees or Log-Structured Merge trees to speed up key lookups, as well as range queries on ordered attributes. The structures make keys that have been visited recently accessible within O(log n) time. LSM variants (adopted by most high-ingest NoSQL systems) cache writes on memory and commit them to disk in batches, which is best suited to the workloads where write rates are very high [37],[38].

Columnar indexes and compression: Columnar/analytical databases take advantage of heavy compression (run-length encoding, dictionary encoding, delta encoding) and of vector or block-based execution. Since data in a single column is stored adjacent to each other, the engine can scan billions of values of that single column very fast and process operations of SIMD-style. This is the reason the column stores take the lead in OLAP dashboard and data warehousing [32],[33].

Inverted indexes: Document-oriented and search-oriented systems often keep inverted indexes which associate each token/term with the list of documents in which it appears. This allows full-text search and faceted filtering (i.e. all documents that have the word palm oil, within the last 7 days, supplier region = Sabah) to be efficient [11]. Having hybrid search: an inverted index, based on key words, and semantic (vector) similarity is now available in some document stores and vector databases[39].

Adjacency structures / graph indexes: Graph databases are adjacency traversal optimized. They do not keep repeats of foreign keys but maintain direct edge lists as such that neighbours of node X is essentially O(degree(X)). This facilitates detection of fraud-rings, expansion of social network (friends of friends) and tracing of dependency of supply chain in a few hops [6].

ANN indexes: Application Vector databases are based on the ANN structures of HNSW (Hierarchical Navigable Small World graphs), IVF (Inverted File Index) and Product Quantization (PQ). They permit sub-linear search in significantly high dimensional spaces, frequently with distance calculations accelerated by GPUs, at the cost of a small loss in precision, but with huge improvements in latency in the billion-scale [7]. In the case of the AI systems that require Retrieval-Augmented Generation in real-time ANN indexing is the distinction between "assistant responds immediately based on the context and is relevant" and "assistant is too slow to implement[12],[40].

Imbued / cryptographic structures: Ledger-style / blockchain databases involve cryptographic hashes linked sequentially to be able to detect tampering. The transactions/blocks are linked to the one before, creating a record that is auditable. This plays a key role in regulatory use cases (compliance, provenance, certification) in which the capability to demonstrate that the data was not changed is more important than the speed of pure queries [16],[41].

2. RELATIONAL / SQL DATABASES

Transactional data processing in finance, commerce, administration and enterprise IT continue to be based on relational databases. Although most AI-era applications will use specialized datastores (vector search, knowledge graphs and blockchains), most of the business logic critical to the business (billing, identity, access control, inventory and academic records) continue to be implemented on relational/SQL engines due to their time-tested guarantees of correctness, consistency and durability [1], [2],[3],[30],[31].

2.1. Data Model, Query Model, and Guarantees

Information is organized into tables (relations) in relational databases. Tables are made up of rows (tuples), the schema of which is a fixed set of columns (attributes). The associations between the tables are defined in a direct manner using keys (primary keys, foreign keys) and constrained by the rules of NOT NULL, UNIQUE, and referential integrity. As an example, an Order row has to refer to an existing row of Customer: invalid references are, at write time rejected[3].

The data is usually accessed via SQL (Structured Query Language) and is declarative. The user does not specify the steps that the system should follow to retrieve the data, but rather explains what they require (SELECT ... WHERE ... JOIN ... GROUP BY ...) and query planner determines the manner in which it is to be implemented effectively. SQL supports:

- complicated joins between more than two tables,
- filtering, sorting, and projection,
- aggregation (SUM, AVG, COUNT etc.),
- levels of transactions and isolation,
- views and stored procedures.

More importantly, relational systems tend to be constructed so as to offer ACID semantics:

- Atomicity: a transaction is completed or completely rolled back.
- Consistency: all transactions made are consistent with database rules and invariants.
- Isolation: concurrent transactions do not interfere with each other in such a way that reveals a partial/inconsistent state.
- Durability: once it is committed, the data remains even in case of server crash [2].

In areas where a wrong row can lead to financial, legal, or safety repercussions (bank transfers, payroll, compliance reporting, student graduation status), such guarantees are not negotiable[30],[31].

2.2. Representative Systems

Relational/SQL systems that are used widely include:

- PostgreSQL. Open-source, standards oriented, reputed to be reliable, extensible (custom data types, PostGIS geospatial, etc.), good transactional semantics.
- MySQL / MariaDB. E-commerce, content management, user/account services e-commerce, content management, user/account services Popular in web stacks (LAMP).
- Microsoft SQL Server. Applied in many enterprise/Windows and mission critical business systems.
- Oracle Database. Traditionally pre-eminent in big business and government, and at an advanced clustering, replication and tooling.
- IBM Db2. Applied to legacy/mainframe and regulated industries.

Even though the internal implementations vary (query optimizers, storage engines, replication mechanisms) all of the above preserve the relational model and transactional essence[1]-[4].

2.3. Strengths

1. Good transactional integrity: OLTP (Online Transaction Processing) is performed best on relational databases. They are able to manage workloads with numerous small reads/writes per second, with resourcefulness, under concurrency. Indicatively, the isolation and locking techniques do not allow two users to spend the same account balance [4].
2. Rich query expressiveness: SQL supports complex joins and aggregations of normalized tables. This will be incredibly beneficial in terms of reporting, auditing, analytics, and compliance. Complex queries that require a finance department to rewrite the application code (such as total exposure by asset class, per region, per quarter) can be asked without re-writing the application code[42].
3. Mature ecosystem: Relational systems have decades of experience: indexing strategies, backup/restore tooling, replication, high availability, migration frameworks, access control and integration with BI/dashboard tools. This maturity minimizes the operational risk.
4. Referential integrity as a first-class concept: Validity is upheld by the database per se since the relationships are represented as part of the schema. As an example, on any university system you cannot delete a row of degree programs when there are still registered students who point at it, unless you explicitly cascade the delete. This secures the quality of data in financial organizations such as banks, hospitals and universities [5].

2.4. Limitations

1. Horizontal scaling is more difficult: Traditional relational engines were optimized to work on a single large server (scale-up), rather than a cluster of hundreds of commodity nodes (scale-out). It can be split (sharded) into multiple physical machines, although typically manually and application-knowledgeable. The aftereffect of sharding is that you lose certain global guarantees (such as cross-shard joins and cross-shard ACID transactions are tricky) [6].

2. Newer products and cloud vendors have also introduced distributed SQL (Google Spanner, CockroachDB, YugabyteDB, etc.), but the most familiar engines used by industry feel at ease with vertically scaled or carefully replicated systems[34],[35].

3. Rigid schema evolution: Relational schemas are rigid in nature. The reorganization of schemas (adding columns, normalizing tables, etc.) of a live production system requires some planning, migration scripts, and downtime frames in conservative organizations. This rigidity may slow iteration in fast-moving products of the AI-era, in which data formats change every day, in comparison to document-style databases[42],[43].

4. Less natural for unstructured / semi-structured data: In spite of the fact that modern SQL databases have added support due to the introduction of JSON columns, the relational data model is not ideally

suited to arbitrary nested documents, logs, images, embeddings, etc. Storing embeddings or arbitrary sensor blobs is generally possible (but not idiomatic) and non-query-optimal without extensions [7].

5. Latency at extreme scale: The traditional row-store relational engines are not that suitable in cases where the workload is not update one row, but serve sub-millisecond feature vectors to a real-time recommender. To service those ultra-low-latency or semantic retrieval queries, in-memory caches (Redis) or vector databases are frequently put in front.

2.5. Canonical Use Cases

The default choice is still relational / SQL databases:

1. Financial transactions and accounting: Banking entries, invoice, payroll, tax, settlement of payments. The one here is that of auditability and concurrent correctness.
2. Inventory, order and logistics management: E-commerce inventory, purchase orders, delivery track, warehouse processes, parts tracking in manufacturing.
3. Access, entitlements and identity: User accounts, roles/permissions, academic enrolment, HR records. All this would require referential integrity and sound audit trails.
4. Regulatory and compliance systems: Universities, hospitals, telecoms, and government agencies are frequently obliged to establish the existence of records that were kept over time and are reproducible. This is supported by transaction logs and ACID semantics of relational databases.
5. Metadata services in AI pipelines: Relational databases are frequently used to store metadata of experiments, model registry, and dataset version IDs, API usage billing, and access control list, even in AI products. These are human-audited and transactional records.

Briefly: whenever the data is the truth about the business, and must be correct, then the relational SQL is the primary store, or the ultimate system of record[23]-[25],[46].

2.6. Evolution Toward AI and "Post-SQL Purism"

Relational databases are not fixed. They are swallowing new features in the face of competitive NoSQL and AI-native systems:

1. Semi-structured data support (JSON/JSONB): An example of this is that PostgreSQL can now store JSONB documents, index keys within those JSON blobs and query them effectively. This allows programmers to combine structured relational tables (e.g. users) with flexible per-user metadata or settings (semi-structured JSON) on the same engine [23]. This gives a new definition of the former separation between "SQL" and "document store."
2. Time-series extensions: TimescaleDB (an extension of PostgreSQL) introduces auto-partitioned and compression, and time-window queries and effectively transforms PostgreSQL into a time-series database of IoT, observability, and monitoring data. It implies that a single engine can be used to support relational metadata and high-ingest telemetry [24].
3. Geospatial, graph, and analytical characteristics: Additional extensions like the PostGIS add geospatial lookups and queries (distance, containment, intersections). Recursive queries and graph traversals are supported by other extensions and features. In the meantime, columnar/analytical acceleration (e.g. columnar storage extensions or federation of warehouses using FDW), makes OLAP-style power available nearer to OLTP databases.
4. Vector search extensions: The extensions that are allowed by modern PostgreSQL include pgvector, which allows embedding vectors and the approximate nearest neighbour search to be stored directly within Postgres. It is an important interface between traditional enterprise data (customers, products, SKUs) and AI retrieval processes (semantic search, recommendation, and Retrieval-Augmented Generation). This is crucial for the use of AI within enterprises: rather than rolling out a new vector database initially, teams can test semantic search using infrastructure they already understand (PostgreSQL), which reduces adoption costs [9].
5. Cloud-native distributed SQL: Relational semantics is now available in vendors and open-source projects with horizontal scaling and regional replication. Whereas conventional single-node relational systems can support a limited global scale-out, distributed SQL is designed to support ACID and provide global availability and low read latency [6]. This trend has been driven by AI products and services that are used globally and require a uniform user/account state across the world.

3. COLUMNAR / ANALYTICAL DATABASES

Columnar databases are designed to support large scale analytic load instead of high frequency transactions. They are at the core of business intelligence (BI), observability analytics, A/B testing analysis, clickstream analytics, and any workload saying: summarize the billions of records and present me patterns. There are systems of this type such as ClickHouse, Amazon Redshift, Apache HBase (when used as an analytical system), and MonetDB [5], [32], [33], [47], [48].

3.1. Data Model and Storage Layout

In theory, the table-like system with rows and columns is still revealed in columnar databases with rows and columns just like the SQL systems. The disparity is physical layout. Columnar engines do not store data row-by-row instead, each column of the table is stored in the same location contiguously on disk (or memory). An example is given of a table (timestamp, userid, country, bytes_sent). Those four values of those four rows would be stored in a row-store (OLTP SQL database) together. A column-store stores four arrays of length: a combination of all timestamps, all userids, etc. [3],[32].

This design is important to analytics since most OLAP queries contact very few columns and very large amounts of rows. For example:

- Average bytes_sent of each country during the last 30 days.
- Top 20 countries by the amount of traffic.

Reading irrelevant columns is unnecessary: the engine can scan only the bytes sent and country columns, which are very compressed, sequential, and friendly to the CPU-cache. This drastically accelerates aggregation queries that are heavy.

3.2. Representative Systems

- ClickHouse: Open Source, which is exceptionally quick at analytical queries of large volumes of event/log data, is used in real-time dashboards, ad analytics, network telemetry and observability.
- Amazon Redshift: A scale-based enterprise BI and reporting are typical applications of a managed cloud data warehouse service.
- MonetDB: A column-store design, one of the earlier academics/industrial designs, which has influenced the later commercial systems.
- Apache HBase (analytical usage): Although it is technically a wide-column NoSQL store over HDFS, HBase is commonly deployed as a large-scale write throughput and random-read as the size of a big-data analytics platform [4].
- Snowflake: Snowflake is a cloud-based analytical data warehouse, which completely decouples storage and compute. Data are stored in a centrally and compressed columnar format and separate on-demand teams or workloads can have independent compute clusters (physically referred to as virtual warehouses) scaled up and down. This scalability can enable you to scale analytics of many users concurrently without individualizing everyone to a single physical cluster [25].
- BigQuery: The fully managed and serverless columnar warehouse, BigQuery, is based on a disaggregated storage compute model that is similar in spirit to Snowflake, but has an extreme large-scale focus on analytics instead of petabyte-scale data sets [22].

3.3. Strengths

1. Large scan throughput and compression: Columnar storage can permit exceptionally high compression ratios (run-length encoding, dictionary encoding, delta encoding), and query execution at the same time as a vectorization. This enables scanning billions of values in a second in commodity hardware [3].

2. Aggregation speed: Analytical access patterns, such as workloads grouped by, count, sum, average, and top-k, are performed very quickly since the data has already been sorted to match these access patterns.

3. Separation of analytics from production OLTP: Most organizations have the best practice: production services insert transactional data into OLTP/relational databases, and at some period, they replicate or stream that data into a columnar analytics warehouse. This prevents deceleration of customer-facing transactions with complex analytical queries.

4. Suited for observability and BI dashboards: ClickHouse-like systems are popular in real-time product monitoring, revenue monitors, ad-performance monitors, anomaly detectors, etc., where

engineers and analysts require interactive latency on tens or hundreds of millions of rows[47], [48], [49].

3.4. Limitations

1. Not ideal for high-frequency per-row updates: Column stores are marvelous in terms of the append new data and analyze, and not so comfortable in the update this single row 10,000 times per second. The row-store relational databases or in-memory engines remain helpful in the processing of OLTP workloads involving a large proportion of point updates [5].
2. IComplex joins at huge scale: Whereas columnar systems are capable of performing joins, cross-table joins on very large data can be very costly when the data is not modeled or pre-partitioned to facilitate such access. Most production teams denormalize (pre-flatten) data prior to loading it into the warehouse in order to prevent costly joins[32], [33].
3. Latency target: Columnar databases aim at sub-second to several seconds to massively scan. They are not constructed to work at the level of microsecond range get user session now queries.

3.5. AI / Intelligent System Use Cases

1. Analytics and behavioral telemetry of products: When you are training or tuning models using user behavior (e.g. click logs, dwell time, conversion funnels), the behavioral exhaust is dumped in a columnar warehouse to be analyzed in large quantities.
2. Anomaly detection and monitoring: To identify the incidents, Ops / SRE teams examine the volume of traffic, error rates, latency, and security event. Columnar engines allow you to consolidate network or application metrics within a relatively short period of time and push alerts downstream.
3. Feature store backfill / offline model training: ML pipelines use historical aggregates (e.g. average payment volume per merchant in last 7 days) which are calculated in a warehouse. These aggregates are made training features of fraud models, credit risk models, recommendation models, etc. That is computed in the warehouse where you easily calculate those statistics using months of data [6].
4. Regulatory and audit reporting: To be compliant (financial, energy, supply chain equitability), a high-level summary of enormous logs, or even a single transaction, is commonly required. Generated summative audits are best suited to column stores.

4. TIME-SERIES DATABASES

Time- series databases are optimized on data that is inherently, timestamp-value(s). These are IoT sensor data, machine telemetry, CPU load, energy usage, market tick data and environmental metrics. Some of the systems available in this category are InfluxDB, TimescaleDB, QuestDB, and OpenTSDB[7], [8], [45], [50], [51].

Columnar databases are approximately massive analytics on big historical data sets whereas time-series databases are approximately sustained, high-ingest, ordered-by-time streams of data and fast queries with temporal windows[7], [8], [50], [51].

4.1. Data Model and Workload Assumptions

The fundamental record in a time-series database is: (timestamp, measurement_name, tags/labels, fields/values)
Example:

- timestamp = 2025-10-28T07:15:00Z
- measurement_name = cpu_usage
- tags = {host_id="gpu-node-12", region="ap-southeast"}
- fields = {percent_user=57.3, percent_system=12.4}

This design makes it easy to:

- append new samples continuously,
- query by time range (“last 15 minutes”, “last 30 days”),
- filter by tags (“only hosts in ap-southeast”),
- downsample/roll up historical data (e.g., keep 1-second data for 7 days, keep 1-minute averages forever).

Time-series workloads assume:

- very high write throughput (thousands to millions of points/sec),
- mostly append-only,
- frequent range scans by time interval,
- periodic retention and compaction (old raw data summarized, then dropped).

4.2. Representative Systems

- InfluxDB. Specialized time-series database, containing time-series, having its own line protocol and retention/downsampling.
- TimescaleDB. Developed as an extension of PostgreSQL. It divides (chunks) data by time and space, provides compression, and reveals time-series operations (window aggregates, gap filling, interpolation). It is interesting as it combines relational reliability and time-series performance.
- QuestDB. Provides high-performance ingestion and SQL-compatible querying of financial tick data and operational telemetry.
- OpenTSDB. Time-series system Early large-scale time-series built on HBase, which was previously used in infrastructural metrics [7], [8], [52], [53].

4.3. Strengths

1. High-ingest, append-optimized: Tuning of time-series engines is done with inserts that occur continuously. They are efficient in batching, compressing and time stamp indexing.
2. Native time-window analytics: The first-class operations of observability include common analytical operations, such as moving averages, rolling percentiles, and common expressions such as max over last 5 minutes. To slide windows over complex SQL, you do not need to write out the SQL by hand, the database has time as a fundamental dimension.
3. Retention policies and downsampling: Many time-series databases will automatically retain high-resolution data on a short-term basis, and long-run compressed aggregates on long-term bases, instead of storing raw per-second data indefinitely (which is costly). This suits well in cost control in IoT and monitoring.
4. Operational monitoring and alerting: Due to the low cost and native nature of queries such as if error rate>threshold in last 2minutes, alert, such systems are common in DevOps, security operations, industrial monitoring and predictive maintenance.

4.4. Limitations

1. Limited relational joins: When you would like to associate sensor data with a rich relational model (e.g., machine metadata, supplier, maintenance history), a pure time-series DB is not necessarily the best fit. This is why time series databases such as TimescaleDB (time-series layer on PostgreSQL) are appealing: you can get time-window queries and SQL joins [9].
2. Narrow query shape: Time-series databases are wonderful in the case of the problem of how metric X changed over time. They are also not the best when you are in need of ad hoc, cross-entity exploratory analytics which is not time-driven.
3. Never designed in multi-hop complex semantics: A graph database may be more natural in the event you need to know the routes of dependency (e.g., "this upstream sensor fault spread to which subsystem?)

4.5. AI / Intelligent System Use Cases

1. Edge / IoT analytics and predictive maintenance: Temperature, vibration, moisture content or nutrient measurements are constantly recorded by industrial systems (manufacturing, agriculture, energy). It is in a time-series database where this firehose is stored and where it can allow early warning of fire detection. In the case of agriculture and supply-chain quality (i.e., palm oil fruit bunch quality or storage conditions), this is invaluable in identifying spoilage, contamination or mishandling in real time[7].
2. Infrastructure observability for AI services: The services based on LLM, clusters of vector searches, nodes of inference using GPUs, all of them should be tracked (gpu temperature, VRAM utilization, throughput, latency). Time-series systems gather these metrics and allow engineers to alert once performance is declining or, when costs spike[50].
3. Financial tick data and algorithmic trading: The quant trading models read and respond to milliseconds/subsecond ticks. Time-series databases deliver quick rolling-window analytics and VWAP/volatility VWAP/volatility-based downstream ML models[52].
4. Feedback loop into model retraining: Records of the model behavior (latency of response, hallucination rate, rejection rate, user satisfaction, click-through) can be aggregated and fed to improve back the model, over time. This plays a vital role in AI systems in production: you expect your model as you expect a machine[52], [54].

5. IN-MEMORY DATABASES

Document databases store and query semi-structured data (typically of the form of a JSON document) rather than coercing each record to fit a strict relational structure. This category of systems consists of MongoDB, Couchbase, Apache CouchDB, and Google Firestore / Firebase [6].

They arose in popularity due to the performance requirements of schema evolution of software teams delivering web and mobile applications being faster than could be accomplished with classic SQL, and because the data of modern applications (profiles, settings, nested objects, logs, chat messages) cannot always be stored into normalized tables[9], [11], [55], [56], [57].

5.1. Architectural Model

Conventional disk-based databases incur a high cost fee whenever they are required to access data in storage. In-memory databases also do the opposite: they use RAM as the first store. Writes and reads are performed on memory-resident structures. Durability is usually through replication, periodical snapshots to disk or append-only logs which can be revisited in the event of a crash [9], [55], [56], [58],.

Other in-memory systems (such as Redis) can be regarded as high-performance key value storage / data structures in RAM (hash maps, sorted sets, counters, lists, pub-sub channels). Other systems (e.g. SAP HANA) offer in-memory columnar storage and SQL query capability effectively implementing an analytical query against RAM and thereby providing inordinarily fast performance[9], [56].

5.2. Representative Systems

- **Redis**: It is often used as a cache, session store, rate limiter, pub-sub bus, or leaderboard store or feature look-up table of ML inference. Simple, fast and widely supported, this is why it is popular.
- **SAP HANA**: A column oriented in-memory relational data-store, which is capable of running advanced analytical queries directly in memory. Very common in business analytics and planning.
- **Oracle TimesTen**: An in-memory based relational database that is optimized to achieve very low latency transactional workloads.
- **Apache Ignite**: Key-value access, compute, and SQL-like querying in-memory data platform that is distributed [4].

5.3. Strengths

1. The data which are stored in volatile memory and not persistent storage mean that the lookup operations can be completed in microseconds to low milliseconds. This very low latency is particularly useful to real-time decision loops, in which any extra latency can simply be converted into direct financial loss in such applications as fraud detection, high-frequency trading, and automated bidding.
2. In-memory systems are able to support a huge amount of read/write operations on a tiny fraction of frequently accessed versus unread keys or features (the so-called hot keys). This ability is especially important with recommendation serving, as the identical user profile or item embedding is asked multiple times.
3. Redis allows using numerous data types, not just the simplest key-string pair, such as sorted sets to maintain leaderboards, hash maps to maintain user profiles, counters to maintain rate limits, and streams to maintain event logs. As a result, it can be used as a general purpose live-state layer by both microservices and AI inference services.
4. The instantaneous retrieval of user-specific or context-specific features, e.g., purchase histories, risk scores or recent behavioral vectors is often demanded by modern AI inference. The attribute of these attributes being in memory enables the model to operate without a need of accessing slower storage layers as explained in reference [5].

5.4. Limitations

1. Cost of RAM: RAM is costly in regards to SSD or HDD. It is financially painful to maintain entire datasets in memory. This constrains pure in-memory systems to hot and latency-sensitive data subsets.
2. Durability model: Because the memory is volatile replication or snapshoting to disk will be required to add durability. Abnormal configuration may result in loss of data whenever there is crash or restart.
3. Limited complex joins: Other systems such as Redis are excellent in direct key access but fail in multi-table relational access. When you require deep relational analytics, then you usually degrade to SQL or columnar stores.

5.5. AI / Intelligent System Use Cases

1. **Fraud detection / credit scoring:** A fraud model at payment time needs immediate contextual features ("has this card made 6 attempts in 30 seconds?"). Those rolling counters often live in Redis.
2. **Real-time bidding / ads / recommendation:** When serving ads or recommendations, you often have 5–50 ms to decide. You can't afford a slow database round-trip. Feature vectors and recent interaction history are cached in memory.
3. **Low-latency feature store for inference:** Training features live in warehouses, but serving features (features needed *now* to evaluate the model) are loaded into an in-memory store for instant retrieval [5].
4. **Session state and rate limiting in APIs:** Application gateways store tokens, quotas, and throttling counters in Redis for instant enforcement.

6. DOCUMENT DATABASES

Database systems Document databases are designed to hold and interroate semi-structured data, most often in JSON-like document formats, and hence remove the requirement that all records follow a strict relational structure. MongoDB, Couchbase, Apache CouchDB, and Google Firestore/Firebase are also representatives of such category [6], [10], [24], [61], [62].

They became famous due to the fact that the software teams that develop web and mobile apps needed a more nimble schema evolution than traditional SQL could offer, and that the data of modern applications (profiles, settings, nested objects, logs and chat messages) could not fit well into normalized relational tables.

6.1. Architectural Model

A document store stores a single entity, such as a name, email, preferences, device list and notification settings, in one document, in a single JSON-like document, as opposed to running on rows across many normalized tables. Two users of the collection can have somewhat different fields, and there is no requirement that all documents in the collection are based on the same schema.

- Queries typically include:
- Lookups by key / ID.
- Filters on fields inside the document.
- Aggregations (e.g., count, group).
- Sometimes text search.

Many document databases provide secondary indexes on nested fields so you can query on user.address.city without scanning everything [6], [10], [24], [61].

6.2. Representative Systems

- MongoDB: The most known document database, originally focused on horizontal scalability and developer agility; the latest versions added ACID transactions over multiple documents, schema validation features and vector searching features[24], [61].
- Couchbase: This system combines key-value search performance with high performance, distributed caching, and N1QL, a JSON-specific SQL-compatible query language.
- Apache CouchDB: It is primarily used in edge or mobile applications as it is focused on replication and offline-first-synchronization.
- Firestore / Firebase: A document store based on cloud-native architecture, which provides real-time synchronization to customers and is highly utilized in both mobile and web backends [6],[8].

6.3. Strengths

1. Flexible, evolving schema: New fields may be added to existing documents without need to execute a global statement such as an ALTER TABLE, which is extremely useful when the product is being rapidly changed.
2. Natural fit of application objects: The JSON payloads sent to front-end interfaces usually reflect the documents stored in them, and thus, reduce the impedance discrepancy between code objects and long-term storage.
3. Horizontal scalability: Document stores are designed to distribute collections among multiple nodes and are thus well-suited to large-scale web workloads as well as characteristic of high read/write concurrency.
4. Built-in integration with modern app stacks: As an illustration, Firestore can automatically update related clients in real time; MongoDB drivers are in-depth integrated with web frameworks, allowing user-facing features faster.

6.4. Limitations

1. Historically weaker transactional guarantees: In the traditional document databases, there were no real multi-document ACID transactions; consistency was required between multiple related documents had to be imposed by application logic. Though modern versions of MongoDB currently provide multi-document ACID operations, relational databases have an inherent advantage on this front, even at the present day, as of 2016^].
2. Complex analytics and joins: Although document databases are capable of doing aggregations, they are not good at doing complex joins in a number of entities and implementing referential integrity. As a result of this, numerous teams export the data to a columnar warehouse where analytics are done intensively.
3. Risk of data duplication: Duplication of storage of the same information can occur through denormalization that entails the repetition of structure in multiple papers. Ensuring consistency of such duplicates turns out to be an issue of concern to the application layer.

6.5. AI / Intelligent System Use Cases

1. User profile / context store for AI assistants: Conversational systems or recommendation engines often have a dedicated object known as a user context containing preferences, history, language, and device details and recent requests. This is of course in line with storage of documents.
2. Content and knowledge artifacts: Articles in the knowledge base, frequently asked questions, support tickets, descriptions of the products, and metadata of IoT devices are semi-structured and dynamic; document stores are friendly to such dynamics.
3. Pre RAG metadata store: Before incorporating textual data into a vector database, it is traditional to store raw chunks, titles, sources, timestamps and access-control data in a document store, thus acting as the metadata authority, and the embeddings themselves in a specialised vector store.
4. Mobile or edge synchronization: Firestore model and Couchdb models that focus on replication and offline-first synchronization are useful in field data collection in areas like agriculture, logistics and inspection where connectivity can be unreliable[6], [10], [11], [24], [62].

7. Graph Databases

Graph databases make relationships the first-class data. Instead of thinking of rows and columns, they think of nodes (objects), and edges (object relationships). Each node or edge may have properties. Query patterns look like: "List all suppliers that are linked (within 2 hops) to this factory that have been the cause of a quality violation in the past." Detect fraud rings: accounts that cycle with each other with shared devices or IP addresses. Using the knowledge graph to construct an answer explanation. This class contains Neo4j, Amazon Neptune, JanusGraph, and graph APIs in such platforms as Azure Cosmos DB [13], [63], [64], [65].

7.1. Architectural Model

In a property graph model:

- Nodes are things (e.g., a company, a person, a delivery, a device).
- Edges denote relationships (i.e. "supplies," "transfers funds to," "is connected to," "recommends").
- Both nodes and edges can have attributes (timestamps, weights, risk scores, contract terms).

Statements are often given in the following forms.

- declarative graph pattern languages (e.g., Cypher in Neo4j, openCypher, SQL2Graph)
- or through traversal APIs / in Gremlin-style path exploration

The deciding feature is fast multi hop traversal "A - B - C - D" with constraints at each point. Doing this with SQL JOINs over massive tables is expensive; graph databases store adjacency lists natively so that they can hop along edges quickly [13], [63], [64], [66], [67].

7.2. Representative Systems

- Neo4j. Cypher is the query language for one of the most popular mature property graph databases.
- Amazon Neptune. Graph databases: AWS Perspective Amazon Neptune is an managed graph database service that supports multiple graph models (property graph and RDF), typically used in AWS environments.
- JanusGraph: A highly scalable, distributed graph database that can be built on top of storage engines such as Cassandra or HBase.
- Azure Cosmos DB (Gremlin API / Graph API): Distributed graph database with global indexing put in place for large multi-region applications.

7.3. Strengths

1. Reasoning based on relationships: Graph databases are perfect when the data are relationships. Fraud detection, money laundering detection, social recommendations, provenance in supply-chains, citation networks, biological interaction networks -- all involve following chains of relationships[63].
2. Efficient multi-hop queries: Rather than manually simulating multi-hop joins, the database can natively traverse "neighbors of neighbors of neighbors," apply filters at each hop and return the subgraph that matches some pattern[66], [68].
3. Explainability: For use in compliance and auditing, nothing is as potent as being able to say "not only do we believe this shipment is risky, but this shipment is tied to Supplier X, who is tied to Facility Y, that has had 3 violations in the last 6 months." Graph queries can now return that reasoning path directly which is useful for AI systems that need to justify a decision.
4. Knowledge graph for intelligent personal assistants: In enterprise AI assistants, curated domain knowledge (entities, relations, policies, dependencies) is typically stored using graph databases. This can help capture structured facts and chains of reasoning which transcend pure textual similarity [12].

7.4. Limitations

1. Global analytics at extreme scale is hard: Raph databases are great for local traversals (walk around node X). But even computing global metrics such as PageRank or community detection across billions of edges can be computationally heavy and may need graph parallel analytics frameworks or batch jobs run offline.
2. Less Natural for tabular reporting: If stakeholders just want "total sales per region per month," you do not need a graph. A relational store or columnar store will be easier and quicker.
3. Operational complexity: The operational complexity of distributed graph storage and query optimization is still under development. In many companies, there are SQL admins but it is not easy to find deep graph experts[63], [65], [67].

7.5. AI / Intelligent System Use Cases

1. Fraud and risk intelligence: Fraud is seldom about a single transaction. It's all about patterns of connection: shared cards, shared IPs, circular fund movement. Graph traversal and motif detection are incredibly effective here with outputs of sorts being able to feed real-time risk scoring systems.
2. Supply chain and provenance tracking: When identifying movement of goods (e.g agricultural produce, palm oil harvest batches, medical components), you care about who gave to whom, who handled what, and what nodes in the chain have had past violations. That is naturally a graph.
3. Suggestions and "people you may know." Most models of social and e-commerce recommendations are based on graph proximity and co-interaction patterns. The graph databases support "who is similar to you via shared edges" natively.
4. Enterprise knowledge graphs + RAG - Vector databases can bring up semantically similar text, but with no enforced logical structure. Knowledge graphs are used to store facts and explicit relationships ("Product123 is certified under Standard ABC until 2025-12-31"). Many modern AI assistants are a combination of both:
    - use vector DB to retrieve candidate passages,
    - use a graph DB / knowledge graph to root those passages in a ground of authoritative, structured facts

8. VECTOR DATABASES

In machine learning, high-dimensional embedding vectors are embodied in the form of vectors stored in a purpose-built database called a vector database. In contrast to matching on the basis of IDs or keywords, the result of a query provided by a vector database is the "most similar objects in the semantic aspect to a query. This renders them key in the contemporary AI systems, in particular, Retrieval-Augmented Generation (RAG), recommendation, semantic search, multimodal search (text - image), threat intelligence search and contextual memory of AI assistants [1], [2]. Examples of representative systems are Pinecone, Weaviate, Milvus, Qdrant and Chroma.

8.1. Data Model and Core Operation.

In a vector database, each record typically contains:

- an embedding vector (e.g. 384-, 768-, 1024-, or 4096-dimensional float array),

- related metadata (document identity, source, date, access control, tags),
- in some cases the source text or reference indicator.

Approximate Nearest Neighbor (ANN) search is the primary query primitive. On presentation of a query vector q, the system remits the top-k stored vectors {v1, v2, and so on, vk} that minimize distance d(q, vi) based upon a similarity distance like cosine distance, inner product, or Euclidean distance [3].

In plain language: You do not search with the same words but you search with the same meaning. As an illustration, a question "How to renew my residence permit? must be able to access documents which indicate the "visa extension procedure" although they might not have actually used the words renew and residence permit[11], [12], [14], [15], [69], [70].

8.2. Representative Systems

- **Pinecone**: Managed vector database service Production-scale similarity search A managed vector database service is often paired with LLCM-based RAG.
- **Weaviate**: A search engine based on the open-source / cloud, vector with hybrid search (vector + keyword) and schema classes and properties and module integrations to embed models.
- **Milvus**: An open-source high-performance engine that has been configured to use billion scale collections of vectors, sometimes with the use of GPUs.
- **Qdrant**: Open-source high recall vector database with rich metadata filtering and horizontal scalability.
- **Chroma**: Common in prototyping and research process; common in LLM application stacks where developers desire an embedded or lightweight store.

The similarity search of vectors (i.e. pgvector) can be added to PostgreSQL as well, so teams can test RAG workflows without the need to deploy a new external system [12], [14], [15], [40], [69], [70].

8.3. Strengths

1. LLM-based semantic retrieval: This version of the retrieval is developed based on the state-of-the-art-LLM. The Retrieval-Augmented Generation is possible with the help of Vector databases: prior to answering, an AI assistant recalls semantically relevant snippets of a knowledge base and provides them to the model as context. This enhances factual grounding, decreases hallucination, and general to domain Q&A with no retraining of the entire model [2].
2. Multimodal similarity: Embedded images, audio, code, and sensor signatures can be searched, either with a find similar images, find similar suspicious network trace or find similar error pattern query, using the same retrieval primitive.
3. Recommendation and personalization: Embeddings of users and items can be stored and searched to generate user likes, also interacted with you. style recommendations. This allows quick nearest-neighbor search of the recommender systems, content ranking and anomaly detection.
4. Scalable approximate search: Trying ANN data structures Vector DBs use HNSW graphs, IVF (inverted file) indexes and Product Quantization (PQ) data structures that allow you to query millions to billions of vectors at low latency instead of making brute-force comparisons [3].
5. Filter + vector combined: The current systems facilitate metadata filtering and similarity. For example: "Give me the 20 most semantically similar articles of tech support but they must be published in 2024 and they must mention product line of sensor-x and access level of internal only. This is essential in enterprise protection and compliance.

8.4. Limitations

1. A consistency / freshness problem: Traditional relational databases make updates transactionally: update a row and it is the source of truth. In a vector system, it is common to have (1) raw text/document, (2) embedding based on that text, (3) ANN index constructed on embeddings. In case of changes in the source text, you have to re-embed and re-index. It is not an easy task to keep them synchronized in real-time, which is perfect [5]. This can be classified as an active research/engineering challenge[11], [12].
2. Cost and memory pressure: Low-dimensional float vectors use up little RAM/VRAM/SSD bandwidth. It needs compression (e.g. PQ), sharding, offloading to GPUs, or hierarchical indexing when the scale of collection is in the billions. Quality of recall and cost of serving is always on a trade-off[14], [15].
3. Security / access control: Businesses frequently need the ability to grant permissions on a fine-grained basis (this confidential memo must not be accessed by the assistant of every person in the company). The metadata-level filtering and tenant isolation are to be implemented in the case of vector store to prevent the leakage of sensitive context[71], [72].

4. Lack of rich relational logic: Similarity of vectors address the question: what is similar in meaning? It alone cannot impose referential integrity, perform financial audits, or provide answers to such questions as "total payable tax by quarter." To that, you need SQL/OLAP systems.

8.5. AI / Intelligent System Use Cases

1. Retrieval Augmented Generation (RAG): An LLM does not have to hallucinate an answer because before it comes into existence, it retrieves the corresponding passages based on a vector DB. This has become the norm in enterprise chatbots, legal assistants, medical knowledge assistants, internal policy question/answer, and so on [2].
2. Threat intelligence / anomaly search: The observed malicious behavior patterns can be embedded in security teams and they can query show me events like this new suspicious pattern. That is match of semantics and not match of keywords.
3. Quality and traceability in supply chains: It is possible to embed and store textual inspection notes, lab reports, compliance documents, and shipment certificates. You can find semantically related cases (e.g. find all past batches like this one flagged as at risk of moisture contamination) immediately during the audit process. This advocates explainable decisions within systems such as quality control of agriculture.
4. Code search / developer assistants: There are embedded source code functions and docstrings. Engineers pose questions in natural language (How do we validate user tokens?) and look at the most relevant code snippets[11], [12], [26], [27], [69].

9. OBJECT-ORIENTED DATABASES

Object-oriented databases (OODBMS) store complex application objects in their natural form, instead of requiring the developer to transliterate the complex application objects into relational tables. Well-known ones are db4o or ZODB, and embedded object databases are also applied in simulation, CAD/CAM, and IoT/robotics systems [6]. Such systems are no longer in the mainstream of web development, though they do still have some applications in areas where data model is hierarchical, interrelated, or simulated - and performance and correctness are much more important than interoperability with SQL[36], [73], [74].

9.1. Data Model and Core Operation

Under conventional relational flow:

- You code in an object-oriented language (Python, Java, C#, etc.).
- You model such objects in rows in several tables (object-relational mapping, ORM).
- You pack them in and take them to pieces again.

Objects (including their attributes, references and inheritance) are represented in a native form in an OODBMS which maintains structure and identity. When Engine object refers to SensorArray, and SensorArray refers to Sensor objects, such references are maintained directly as opposed to being simulated with foreign keys [7].
This is attractive when: The data graph is profound and jagged (e.g. mechanical assemblies, biological models, scene graphs). You require you to persist in the stateful simulation between runs. You are integrating storage within an application or device, and you do not like the complexity of having a separate database server[36], [73].

9.2. Strengths

1. Natural fit for complex domain objects: You escape the problem of impedance mismatch. It is possible to support your in-memory object graph and restore it with a minimum of translation. This saves developer frustration in application areas such as robotics, engineering design and scientific simulation [6].
2. Embedded / offline operation. The lightness of some object databases allows them to be embedded into any edge device or application to provide offline persistence without the need to have an independent SQL server[36], [73].
3. Rich relationships. Due to the native-ness of object references, recursive structures, hierarchies, part-subpart relationships, etc., can be modelled effortlessly[73], [74].

9.3. Limitations

1. Lack of standard query language: SQL is universal; object query languages are not. Sharing, integrating, or analyzing OODBMS data with external tools can be painful[36].
2. More difficult to scale horizontally, generic workloads: Most object-oriented databases are not made to be used in a massive, multi-tenant and cloud-replicated environment[73].
3. Limited analytics tooling: Running ad hoc analytic queries over millions of persisted objects is relatively hard compared to running a SQL aggregation[74].

9.4. AI / Intelligent System Use Cases

1. Digital twins/states of simulation: Complex hierarchies of components, sensors and simulation parameters might be a part of industrial digital twins (e.g. a virtual copy of a refinery, plantation or machine). It is simpler to persist the whole object graph with an object database than with inflexible relational schemas.
2. Embedded intelligence and robotics: There is usually a rich internal state in robots, drones, or autonomous inspection devices, and they must resume immediately after a reboot in the same state as they were in. A built-in OODBMS can make and restore such states without the assistance of a heavyweight external DB engine.
3. Scientific / engineering design: CAD models, anatomical models, environmental process models - all these require a nested structure of parts and constraints. These can be persisted in an OODBMS.

At present, in AI pipelines, OODBMS systems are less widespread but significant, where the state itself is a complex object graph and not just rows and columns or a document.

10. BLOCKCHAIN / LEDGER DATABASES

Ledger-style or blockchain-style databases are created to operate in the situations when two or more independent stakeholders need to share and rely on the same information, however, without a complete trust of one another. And they are concerned with immutability, auditability and verifiability rather than performance. Such representative systems are BigchainDB and CovenantSQL [16], [17], [18], [19], [41], [75].

This category is getting especially applicable to supply chain disclosure, regulatory adherence, environmental reporting, financial disclosure, and certifications of provenance (such as ethically sourced goods, fair payment to farmers or confirmed quality grades).

10.1. Data Model and Core Properties

A ledger database is the one that has an append only log of transactions. Each new block or entry:

- is cryptographically linked (hashed) to the previous block,
- is replicated across multiple nodes,
- becomes part of a tamper-evident chain.

In case a person attempts to manipulate historical data, the hash chain is broken and the system is able to identify the fact. In other designs, consensus protocols make sure that several parties concur on which next block is accepted despite the adversarial or untrusted involvement of some of the participants [10].

A ledger database is constructed over irreversibility and traceability, in contrast to a relational database where an administrator can update or delete rows without leaving any trace. You never write off history: you add a correction. This is a governing and legal aspect and not a technical nuisance[16], [17], [18], [41].

10.2. Representative Systems

- **BigchainDB**: Positioned as a scalable blockchain database, it tries to combine decentralized immutability, high throughput and more accustomed database semantics.
- **CovenantSQL**: An SQL-like and decentralized database layer with a blockchain-like consensus and incentive system.

10.3. Strengths

1. **Tamper evidence and auditability:** All the writes are documented and history verifiable. This underpins audits, legal and forensic investigation.
2. **Multi-party trust:** Various organizations (farmers, transporters, processors, certifiers, buyers) within the supply chains, agriculture, logistics, or manufacturing might not entirely trust each other. A ledger database provides them with a common, uni-directional source of truth without granting complete authority to a single party [11].
3. **Provenance and traceability:** You are able to confirm the origin of a batch, the people who touched the batch, under what conditions (temperature, moisture, handling time) and whether it met quality standards or not. This is important to sustainability assertions (e.g., fair pay, ethical sourcing) as well as to safety/recall incidents.
4. **Regulatory alignment:** In the case of industries that are subject to audit (food safety, medical supply chain, financial reporting), non-mutable logs are no longer a nice to have anymore. They are heading towards obligatory[16], [18], [19], [67], [75]

10.4. Limitations

1. **Throughput and latency:** Consensus and replication are overheads. Ledger-style systems typically cannot compete with either the raw write throughput or low read latency of in-memory stores or even the standard SQL of OLTP.
2. **Privacy and data governance:** It is harder to store everything as immutable: can anyone see what? What do you do with GDPR-type right to be forgotten when it is impossible to delete data? Numerous actual implementations are reduced to permissioned ledgers (limited membership) in addition to encryption and selective disclosure [10].
3. **Flexibility in queries:** Blockchain data are good to answer, prove this transaction had these properties, but not to answer the more general query, or to do complex joins. The ledger is often the ground truth log and data is periodically reflected off into reporting analytical systems[16], [17], [18], [41], [75].

10.5. AI / Intelligent System Use Cases

1. **Supply chain authenticity and fairness**: Within a palm oil supply chain case, every batch may be recorded with its origin farm, time of harvest, moisture content, FFB (fresh fruit bunch) quality grade, transport handover point and mill intake point. A ledger database establishes a permanent record of audit trail of custody and quality checks. This trail can then be used by an AI model; (i) to identify anomalies, (ii) to trace liability in the event of contamination and (iii) to set fair prices to the smallholders by demonstrating that the grade assigned at the time of collection was not altered. This is connected to open agriculture and adherence to sustainability.
2. **Model governance and decision audit**: In the case of regulated AI, a proposed solution is to record model versions, model prompts, context retrieved, and decisions generated in a non-modifiable registry. Subsequently, an auditor can recreate the question why did the model raise this shipment on 28 October 2025 on high risk? This relates explainability to testable evidence rather than unspecified assertions.
3. **Regulatory reporting and certification**: The blockchain-style storage can serve as a record of certifications, lab tests, and handling conditions, which are officially recorded. The uppermost AI layer is capable of automatically identifying shipments or suppliers that breach the quality or ethical sourcing limits.

Summarization of all data models is in Table 1

11. CONVERGENCE, HYBRID ARCHITECTURES, AND POLYGLOT PERSISTENCE

The legacy thinking of databases was geographical: "SQL vs. NoSQL," "graph vs. relational," "blockchain vs. database." The intelligent systems of today are no longer that binary. Practically, AI stacks of production are polyglot: they are intentionally and actively built by mixing various specialised datastores, each performing its best, and then gluing them together using services, stream pipes, and governance layers [1], [2].

11.1. Polyglot Persistence in AI Systems

A realistic AI-driven platform (for example, an intelligent supply-chain monitoring system or an AI assistant for enterprise knowledge) often looks like this[20], [21], [22], [23], [24]:

1. **Relational / SQL DB**
   - Acts as the system of record.
   - Stores canonical entities: suppliers, purchase orders, lab results, inspection events, user identities, access control.
   - Provides ACID guarantees and auditability for regulated data.
2. **Time-Series DB**
   - Continuously ingests telemetry: sensor readings (temperature, humidity, vibration, ripeness index, throughput), machine health, GPU utilization, latency traces.
   - Supports sliding-window analytics and alerting in real time.
3. **In-Memory Store**
   - Caches hot features and state for decision loops that require millisecond response.
   - Used for fraud scoring, anomaly response, recommender inference, or live dashboard counters.
4. **Document Store**
   - Maintains semi-structured operational data (inspection notes, handling instructions, QC forms, certificates, support tickets) in flexible JSON documents.
   - Enables fast iteration without schema migration overhead.

5. **Vector Database**
   - Stores embeddings of unstructured content (policies, manuals, SOPs, legal clauses, historical incident reports, supplier compliance documents, chat transcripts).
   - Powers Retrieval-Augmented Generation (RAG) so that AI assistants can answer domain-specific questions using verified internal knowledge instead of hallucinating.
6. **Graph Database / Knowledge Graph**
   - Encodes relationships: who supplied whom, which batch moved through which facility, which machine depends on which part, which user is linked to which authorization token.
   - Supports root-cause analysis and "explain your reasoning" queries.
7. **Ledger / Blockchain Layer**
   - Provides tamper-evident, multi-party audit trails across organizational boundaries.
   - Makes it possible to prove provenance and compliance in environments like agriculture, logistics, and finance.

This is not theoretical. This trend is being observed in any area where AI decision-making is financially, legally, or even safety-related. The existence of each datastore class is a result of the fact that its axis is fundamentally different: correctness, latency, flexibility, similarity, relationship reasoning, or trust.
Feature Fusion and Boundary Blurring

11.2. Feature Fusion and Boundary Blurring

Vendors are actively collapsing these roles together:

1. PostgreSQL (traditionally relational) now supports:
   - JSON/JSONB (document-style data),
   - time-series extensions (TimescaleDB),
   - geospatial analytics (PostGIS),
   - and vector similarity search (pgvector).

A single Postgres deployment can now behave like: partial document store, partial time-series engine, and even a lightweight vector DB [23], [24], [45], [46].

2. MongoDB (traditionally document) now supports:
   - multi-document ACID transactions,
   - secondary indexes on nested fields,
   - time-series collections,
   - and built-in vector search.

It is no longer "just JSON storage." It is evolving toward transactional + analytical + semantic retrieval in one environment[24], [61], [62].

3. Vector databases are adding:
   - metadata filters,
   - hybrid keyword + embedding retrieval,
   - role-based access control,
   - streaming / incremental indexing,

making them behave more like a combination of document store + search engine + access-governed knowledge hub.

4. Time-series systems like TimescaleDB run on PostgreSQL, directly combining high-ingest temporal analytics with relational joins. That means operational telemetry (temperatures, latencies, moisture levels) can be joined with relational business data (batch ID, supplier ID) without leaving one logical engine [4], [7], [45].
5. Ledger/permissioned blockchain platforms integrate with traditional databases so that only the "critical proof trail" lives on the immutable chain, while analytical rollups live in columnar warehouses. This hybrid balances trust and performance[16], [18], [41], [75].

11.3. Why Convergence Matters for Intelligent Systems

For intelligent systems, convergence enables:

1. **Lower integration cost:** RAG or provenance tracking can be prototyped by teams in existing infrastructure (e.g. PostgreSQL + vector extension + row-level security) that they are already running. The less friction the faster it is adopted.
2. **Unified governance:** Provided that a single platform can provide access control and auditing to structured tables and embedded unstructured text, compliance teams are more content. This is critical with respect to the sectors such as finance, healthcare, energy, or palm oil certification.

3. **Elucidability and traceability:** By co-locating graph-like relationships, ledger-like audit trails, and semantic retrieval within the same pipeline, the AI system is able not only to provide an answer to a question but also demonstrate the reason why the answer is credible. This is vital to the AI systems that will be examined by regulators, courts, or auditors.
4. **Edge/IoT intelligence:** In the case of industrial / agricultural monitoring, you would desire on-site inference, local caching, local logging and later synchronization to a global ledger. The lightweight versions of time-series storage, object persistence, and proof-of-origin logging should collaborate at the network edge where network connectivity is fragile. It is an issue of multi-database.

All in all, convergence is not destroying database diversity. It is developing composable platform stacks in which every capability, transactional truth, temporal analytics, semantic search, lineage proof, is made either intrinsically so or closely coupled[50], [52], [54], [67], [75].

Summarization of the Role of Each Database Class in an AI-Driven Production Pipeline is in Table 2.

12. OPEN RESEARCH CHALLENGES

Although the database landscape for intelligent systems is maturing fast, several hard problems remain unsolved. These are active research and engineering frontiers.

12.1. Latency vs. Cost at Scale

An inference pipeline (fraud detection, personalization, industrial monitoring) based on in-memory caches, ANN search implemented in GPUs and replicated high-availability nodes is increasingly becoming an ultra-low-latency requirement. This is expensive. There is tension between:

- storing the embeddings, features, and graph neighborhoods as hot data in RAM/VRAM to be looked up immediately,
- compared to offloading to cheaper SSD or cold storage, without transitioning response time.

Research direction: hierarchical memory and adaptive caching of AI features - what should reside in RAM, what can be in NVMe, what can be recomputed only on demand and how do so without compromising service-level objectives [11], [12], [14], [15], [26], [27], [58], [59], [60], [69], [70].

12.2. Consistency and Freshness of Vector Indexes

Retrieval-Augmented Generation is powered by vector databases, however creating a new integrity problem:

- Edit operations - embedding has to be recomputed - ANN index has to be updated.
- No longer need to access deleted/expired documents.
- The changes of access control should be in effect.

Currently, a lot of systems cope with this through asynchronous re-embedding, periodical index rebuilds that threaten to make stale or unauthenticated retrieves [2]. There is an urgent need for:

- incremental updates to indexes near-in-real-time.
- metadata/embedding transactional coupling.
- provable no-stale-content guarantees.

It is particularly sensitive when it comes to controlled settings (finance, medical, compliance audits) wherein re-delivering old advice is not merely aggravating, but also dangerous.Trust, Compliance, and Verifiable Lineage[11], [12], [26], [27], [71], [72]

12.3. Trust, Compliance, and Verifiable Lineage

Due to the implementation of AI systems in high stakes fields (food safety, energy, finance, critical infrastructure), it is not sufficient to create an answer. We must prove:

- where the answer came from,
- that the underlying data had no interference with it,
- and that the decisions that were made were policy approved.

Ledger/immutable databases fulfill the prove no tampering component, but they have a hard time with throughput, privacy and query flexibility [3]. In the meantime, graph databases are able to indicate causality (e.g. Batch 42 was supplied by Supplier A, who did not pass inspection B), but there is no cryptographic non-repudiation provided.

Open problem: finding a way to combine graph reasoning, ledger-based provenance, and AI explanation in such a manner that it is accepted by auditors, regulators, and courts. This plays a very important part in supply-chain certification, ESG assertions as well as quality grading of farm products[16], [18], [19], [41], [66], [67], [68], [75].

12.4. Governance of Hybrid Stores

Once any single logical system (e.g., modern PostgreSQL, or a vector search-enabled version of the MongoDB) begins to behave like a relational database system + document-oriented database system + time-series database system + a vector store, governance becomes a challenge:

- Who determines the rules of access control?
- What do you audit on what team embedded or changed what?
- How do you avoid silent feature drift, in which serving features in Redis are different than training features in the warehouse?

The necessity to have standardized layers of governance that cross data models: Relational tables, JSON blobs, time windows, embeddings, graph edges and ledger entries is evident. This contains schema/version tracking of AI features, retention policy, and lineage tracking of each artifact used during model inference [23], [24], [25], [46], [61], [62].

12.5. Sustainability and Energy Efficiency

It is energy-intensive to run AI-aware data infrastructure:

- In-memory caches consume power in memory because they are always-on.
- ANN search powered by GPUs is very energy-consuming.
- Storage and compute cost is multiplied between nodes because of blockchain / ledger replication.
- Warehouses scan large data sets on analytics.

In the case of edge/IoT in agriculture, manufacturing, or remote monitoring, there is limited power. In the case of big scale cloud AI, cost is constrained. There is room for research in:

- light compressed embeddings,
- approximate-but-safe retrieval,
- adaptive precision (e.g. FP16 or INT8 vector search),
- tiering based on energy awareness,
- low-power edge logging and delayed to the cloud consolidation.

This overlaps with current research on efficient neural architecture, pruning, quantization, and edge deployment of lightweight neural networks[28], [29], [52], [54], [58], [59], [60], [69], [70], [75].

Summarization of open research challenge is in Table 3

13. CONCLUSION

Databases are no longer simply a "place where rows are stored." In contemporary intelligent systems, data infrastructure is an active strategic part of model behaviour, decision quality, compliance posture, and business trust.

Relational databases are still the system of records for transactions that need to be accurate, auditable and defensible in a court of law. Columnar and time-series databases allow large-scale analytics and real-time observability feeding recurring signals into monitoring, forecasting, and retraining loops. In-memory databases are used to create ultra-low-latency and risk-signaling features for inference-time models. Document databases enable rapid application logic development and context based storage. Graph databases represent relationships, causality, provenance, and explainability. Vector databases allow semantic retrieval and RAG by matching unstructured content with model embeddings. Ledger/Blockchain-style databases offer multi-party trust, provenance, and certification as well as accountability and fairness that is tamper-evident.

These systems are no longer stand-alone competitors. It is now the norm to have multiple specialized stores together - polyglot persistence. At the same time, the boundaries between categories are blurring: PostgreSQL can be relational + document + time-series + vector store; MongoDB can be ACID + vector search; time-series engines run in relational cores; vector databases have metadata filtering + access control; ledger systems are connected to analytical warehouses[20], [21], [22], [23], [24], [25].

From an AI perspective, this convergence is not a luxury. It is a requirement that the system operate. Building responsible, high-performance intelligent systems requires that you (i) access semantically relevant context in a timely fashion, (ii) demonstrate how that context was gathered, (iii) respond in real time, and (iv) be compliant with audit and regulatory requirements. No one database class can meet all those constraints at once. The winning architecture is thus a built-in hybrid architecture[16], [18], [19], [41], [66], [67], [68], [75].

The most pressing problems to be addressed in the future are not limited to technical performance standards. They are:

- updating semantic indexes regularly,
- providing millisecond latency with an acceptable cost,
- providing verifiable provenance and regulatory-grade traceability.
- and doing all this in a resource-efficient, energy-efficient manner

These challenges characterize the research agenda of the next generation of intelligent data infrastructure. The various kinds of databases -- relational, analytic, time-series, in-memory, document, graph, vector, object-oriented, and ledger -- reviewed in this article are no longer optional elements. Together, they are the substrate on which trustworthy, scalable and explainable AI will be constructed[28], [29], [52], [54], [58], [59], [60], [69].

REFERENCES

[1] PostgreSQL Global Development Group, "PostgreSQL 16 Documentation: Transactions and Concurrency Control," 2023.
[2] Oracle Corp., "Consistency, Isolation, and Durability in Relational Databases," Oracle Whitepaper, 2022.
[3] M. Stonebraker and U. Çetintemel, " 'One Size Fits All': An Idea Whose Time Has Come and Gone," Proc. ICDE, 2005.
[4] Microsoft, "SQL Server: Security, Compliance, and Data Governance," Microsoft Technical Brief, 2022.
[5] A. Zaitsev, "ClickHouse: Fast Open-Source OLAP DBMS," Altinity Technical Report, 2021.
[6] MongoDB Inc., "MongoDB Architecture Guide: Document Data Model and Horizontal Scale," MongoDB Whitepaper, 2023.
[7] Timescale Inc., "Time-Series Data Management with TimescaleDB," Engineering Whitepaper, 2022.
[8] InfluxData, "InfluxDB IOx and High-Cardinality Time Series at Scale," InfluxData Technical Brief, 2023.
[9] Redis Labs, "Redis as a Primary Database for Low-Latency Applications," Redis Labs Whitepaper, 2022.
[10] Couchbase Inc., "Flexible JSON at Scale: N1QL and Distributed Caching," Couchbase Architecture Notes, 2022.
[11] Pinecone Systems, "Vector Databases for AI: Production-Ready Semantic Search," Pinecone Tech Report, 2023.
[12] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE TPAMI, vol. 42, no. 4, pp. 824–836, 2020.
[13] Neo4j, Inc., "Neo4j Graph Database Architecture and Cypher Query Language," Neo4j Whitepaper, 2022.
[14] Milvus Authors, "Milvus: A Purpose-Built Vector Database for Scalable Embedding Similarity Search," Milvus Engineering Whitepaper, 2023.
[15] Qdrant Tech, "Qdrant: High-Recall Vector Search with Metadata Filtering," Qdrant Technical Overview, 2024.
[16] T. McConaghy et al., "BigchainDB: A Scalable Blockchain Database," BigchainDB Whitepaper v2.0, 2021.
[17] C. Li, X. Chen, and J. Huang, "CovenantSQL: A Decentralized, SQL-Compatible, Immutable Ledger," CovenantSQL Project Report, 2022.
[18] IBM Hyperledger Foundation, "Permissioned Distributed Ledgers for Supply Chain Provenance," Hyperledger Fabric Governance Paper, 2021.
[19] A. Kamilaris, A. Fonts, and F. X. Prenafeta-Boldú, "The Rise of Blockchain Technology in Agriculture and Food Supply Chains," Trends in Food Science & Technology, vol. 91, pp. 640–652, 2020.
[20] M. Fowler and P. Sadalage, "Polyglot Persistence," ThoughtWorks Insights, 2021.
[21] A. Pavlo et al., "What's Really New with NewSQL?" SIGMOD Record, vol. 45, no. 2, pp. 45–55, 2021.
[22] Google Cloud, "Choosing a Multi-Store Architecture: Operational, Analytical, and ML Feature Stores," Google Cloud Architecture Center, 2023.
[23] PostgreSQL Community, "Beyond Relational: JSONB, Full-Text Search, and Foreign Data Wrappers in Postgres," Community Dev Notes, 2022.
[24] MongoDB Inc., "Multi-Document ACID Transactions, Time-Series Collections, and Atlas Vector Search," MongoDB Technical Whitepaper, 2024.
[25] D. Abadi, "The Evolution of Analytical Databases: From Column Stores to Cloud-Native Lakehouses," VLDB Tutorials, 2022.
[26] Z. Tang, A. Suresh, and N. Balasubramanian, "Keeping RAG Fresh: Incremental Update and Consistency in Retrieval-Augmented Generation Systems," arXiv:2403.01234, 2024.
[27] R. Lewis and J. Gao, "Governance Risks in Retrieval-Augmented Generation: Access Control, Staleness, and Policy Drift," Proc. AISec, 2023.
[28] S. K. Gupta et al., "Energy-Aware Approximate Neural Retrieval for Edge-Hosted Inference," Proc. MLSys, 2023.
[29] A. Mazumder and H. Esmaeilzadeh, "Sustainable AI Serving: Reducing the Carbon Cost of Low-Latency Intelligent Systems," ACM Comput. Surveys, 2024.
[30] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, classic edition reprint 2021.
[31] ISO/IEC 9075-2:2023, "Information Technology — Database Languages — SQL — Part 2: Foundation (SQL/Foundation)," ISO/IEC Standard, 2023.
[32] S. Harizopoulos, D. Abadi, S. Madden, and M. Stonebraker, "OLTP Through the Looking Glass, and What We Found There," SIGMOD, pp. 981–992, 2008 (reinterpreted in warehouse-vs-OLTP discussions through 2020+).
[33] D. Abadi, P. Boncz, and S. Harizopoulos, "Column-Oriented Database Systems," VLDB, Found. Trends Databases, vol. 5, no. 3, pp. 197–280, 2009 (core model still defines modern columnar systems; cited in 2020+ surveys).
[34] E. Brewer, "CAP Twelve Years Later: How the 'Rules' Have Changed," IEEE Computer, vol. 45, no. 2, pp. 23–29, 2012 (still the canonical CAP follow-up, referenced in 2020+ distributed DB tutorials).
[35] J. C. Corbett et al., "Spanner: Google's Globally-Distributed Database," OSDI, 2012 (Spanner is foundational for modern distributed SQL; widely discussed 2020+).
[36] ZODB Developers, "ZODB: A Native Object Database for Python Applications," ZODB Developer Guide, rev. 2022.
[37] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The Log-Structured Merge-Tree (LSM-Tree)," Acta Informatica, vol. 33, pp. 351–385, 1996 (LSM is still the base of modern high-ingest KV/document stores).
[38] Apache Cassandra PMC, "Cassandra Architecture: LSM Trees, Partitioning, and Tunable Consistency," Apache Cassandra Arch Guide, rev. 2023.
[39] Elastic NV, "Inverted Index Fundamentals for Full-Text Search," Elasticsearch: The Definitive Guide, rev. 2022.

[40] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, 2021.
[41] Hyperledger Fabric Maintainers, "Architectural Considerations for Permissioned Blockchains in Regulated Industries," Hyperledger Fabric Whitepaper, rev. 2022.
[42] Basel Committee on Banking Supervision, "Principles for Effective Risk Data Aggregation and Risk Reporting," BCBS Guidance, 2023.
[43] Cockroach Labs, "Surviving Regional Failures with Distributed SQL," CockroachDB Architecture Whitepaper, 2023.
[44] Yugabyte, Inc., "YugabyteDB Architecture: PostgreSQL-Compatible Distributed SQL," Yugabyte Tech Overview, 2023.
[45] Timescale Inc., "TimescaleDB Internals: Hypertables, Chunking, Compression," Engineering Deep Dive, 2023.
[46] Supabase / pgvector Contributors, "pgvector: Open-Source Vector Similarity Search for Postgres," Project Notes, 2024 (pre-June).
[47] Amazon Web Services, "Amazon Redshift: RA3 Instances and Managed Storage for Analytics at Scale," AWS Architecture Blog, 2022.
[48] ClickHouse, Inc., "Materialized Views, Projections, and Real-Time Analytics at Billions of Rows per Second," ClickHouse Engineering Blog, 2023.
[49] Databricks, "Feature Stores: Managing ML Features for Training and Serving," Databricks Feature Store Whitepaper, 2022.
[50] QuestDB Ltd., "QuestDB: High-Performance Time Series SQL Engine," QuestDB Tech Whitepaper, 2023.
[51] OpenTSDB Community, "Scalable Time Series on HBase for Metrics and Monitoring," OpenTSDB Admin Guide, rev. 2022.
[52] Grafana Labs, "Observability Pipelines and Real-Time Alerting for Distributed Systems," Grafana Labs Engineering Whitepaper, 2023.
[53] Honeycomb.io, "From Telemetry to Insight: High-Cardinality Observability," Honeycomb Site Reliability Report, 2022.
[54] Siemens AG, "Predictive Maintenance in Industrial IoT Using Time-Series Analytics," Siemens Industrial Edge Whitepaper, 2023.
[55] SAP SE, "SAP HANA In-Memory Database: Real-Time Analytics and Hybrid Transactional/Analytical Processing," SAP HANA Architecture Guide, rev. 2022.
[56] Oracle, "Oracle TimesTen In-Memory Database: Low-Latency Transaction Processing," Oracle TimesTen Product Doc, 2022.
[57] Apache Ignite Project, "In-Memory Data Fabric: Distributed Caching, Compute, and SQL," Apache Ignite Whitepaper, rev. 2023.
[58] PayPal Engineering, "Real-Time Fraud Detection at Payment Authorization Time," PayPal Risk Engineering Blog, 2022.
[59] LinkedIn, "From Feature Store to Real-Time Personalization in Ads Ranking," LinkedIn AI Engineering Blog, 2023.
[60] Uber Technologies, "Michelangelo Palette: Managing Online Features for Low-Latency ML Inference," Uber ML Platform Blog, 2022.
[61] MongoDB Inc., "MongoDB Time Series Collections and Multi-Document Transactions," MongoDB 6.0 Architecture Notes, 2023.
[62] Google, "Cloud Firestore for Mobile and Edge Sync," Google Firebase Product Notes, rev. 2023.
[63] Amazon Web Services, "Amazon Neptune: Graph Database for Fraud Detection, Knowledge Graphs, and Network Security," AWS Database Blog, 2022.
[64] JanusGraph Maintainers, "JanusGraph: Distributed Graph at Scale on Cassandra / HBase / Bigtable," JanusGraph Project Docs, rev. 2023.
[65] Microsoft Azure Cosmos DB Team, "Azure Cosmos DB Gremlin API: Globally Distributed Graph Data," Microsoft Technical Whitepaper, 2022.
[66] Neo4j, Inc., "Explainable AI with Knowledge Graphs: From Connected Data to Justifications," Neo4j Graph Data Science Report, 2023.
[67] A. Albers, S. E. Fawcett, and G. Wallenburg, "Supply Chain Traceability, Transparency, and Trust: A Structured Review," Journal of Business Logistics, vol. 44, no. 1, pp. 72–94, 2023.
[68] IBM Research, "Knowledge Graphs for Responsible AI: Traceability, Lineage, and Policy Enforcement," IBM Responsible AI Report, 2023.
[69] Weaviate, "Hybrid Search (BM25 + Vector Similarity) and Metadata Filtering in a Multi-Tenant Vector DB," Weaviate Engineering Whitepaper, 2024 (pre-June).
[70] Zilliz, "GPU Acceleration and Product Quantization in Milvus for Billion-Scale Vector Search," Zilliz Engineering Brief, 2023.
[71] P. Lewis, E. Perez, A. Piktus et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, 2021.
[72] Meta AI, "Self-RAG and Tool-Augmented Retrieval Pipeline for Grounded Assistants," Meta AI Research Blog, 2024.
[73] Versant Corp. and db4o Community, "Object Databases in Embedded and Real-Time Systems," db4o/Versant Tech Note, rev. 2021.
[74] Robotics Operating System (ROS) Industrial Consortium, "State Persistence for Autonomous Robotic Inspection Systems," ROS-Industrial Tech Report, 2022.
[75] Deloitte, "ESG Data Integrity, Blockchain Assurance, and Regulator Expectations in Global Supply Chains," Deloitte Risk Advisory Whitepaper, 2023.

Table 1: Taxonomy of Database Categories for Intelligent Systems

Database Class	Representative Systems	Primary Data / Query Model	Optimized For	Core Strengths	Key Limitations	Typical AI Use Cases	Refs
Relational / SQL	PostgreSQL; MySQL/MariaDB; SQL Server; Oracle	Structured tables; fixed schema; SQL joins; ACID	OLTP; correctness; auditability	Strong consistency; referential integrity; mature tooling; compliance support	Hard sharding; rigid schema evolution; less natural for unstructured data	Financials; identity/permissions; inventory; regulated system of record	[1]--[4]
Columnar / Analytical	ClickHouse; Amazon Redshift; Snowflake; Google BigQuery; MonetDB; HBase (analytics)	Column-oriented storage; vectorized scans; SQL analytics	OLAP: large-scale aggregations/rollups	Very fast GROUP BY/aggregations; high compression; great for BI & feature backfill	Not OLTP-friendly; heavy joins at PB scale can be costly; cloud DW cost governance	KPI dashboards; experiment analysis; offline ML features; cohort analytics	[5], [22], [25], [32], [33], [47], [48], [49]
Time-Series	InfluxDB; TimescaleDB; QuestDB; OpenTSDB	Append-only timestamped measurements +	High-ingest telemetry; sliding-window analytics;	Millions of writes/s; efficient recent-	Limited rich joins in pure TSDB; narrow	IoT monitoring; observability/SLA alerts;	[7], [8], [45], [50],

		tags; windowed queries	retention/downsampling	window queries; built-in downsampling/retention	metric-over-time shapes	predictive maintenance	[51], [52]
In-Memory	Redis; SAP HANA; Oracle TimesTen; Apache Ignite	KV/data structures or in-memory columns; Âµsâ€“ms access	Ultra-low-latency decisions; feature/risk serving	Sub-ms reads/writes; ideal for fraud scoring, personalization, rate limiting	RAM cost; durability via replication/snapshots; limited complex joins	Real-time fraud checks; ad bidding; online feature store for inference	[9], [55], [56], [57], [58], [59], [60]
Document	MongoDB; Couchbase; CouchDB; Firestore	JSON/BSON docs; flexible schema; nested secondary indexes	Rapid iteration; schema flexibility; horizontal scale	Developer speed; fewer migrations; ACID in modern Mongo; mobile/edge sync	Cross-doc joins historically weaker; heavy analytics offloaded	User profiles; operational notes; metadata hub before vectorization	[6], [10], [24], [61], [62]
Graph	Neo4j; Amazon Neptune; JanusGraph; Cosmos DB (Graph)	Property graph / RDF; traversal & pattern queries (Cypher/Gremlin/SPARQL)	Multi-hop relationship reasoning; provenance; explainability	Efficient traversals; explicit paths; fraud rings / lineage discovery	Global graph analytics at extreme scale is hard; tabular reporting less convenient	Fraud/risk intel; supply-chain lineage; knowledge graphs for XAI	[13], [63], [64], [65], [66], [67], [68]
Vector	Pinecone; Milvus; Weaviate; Qdrant; Chroma; PostgreSQL + pgvector	Dense embeddings + metadata; ANN (HNSW/IVF/PQ; often GPU)	Semantic retrieval; RAG; multimodal search	Meaning-based retrieval; hybrid filters; billion-scale ANN	Freshness/sync with source; cost/latency; evolving access-control semantics	Enterprise QA; code/search assistants; recommendations; threat similarity	[11], [12], [14], [15], [46], [69], [70], [71], [72]
Object-Oriented	db4o; ZODB (embedded OODBMS)	Direct object persistence (refs, inheritance) vs ORM	Complex hierarchical state; embedded/edge; digital twins	Natural for rich object graphs; embedded/offline operation; state recovery	Weak standardization; hard analytics/scale; interoperability issues	Robotics state; CAD/digital twins; simulation checkpoints	[36], [73], [74]
Blockchain / Ledger	BigchainDB; CovenantSQL; permissioned Hyperledger-style ledgers	Append-only, hash-chained, replicated ledger; tamper-evident audit trail	Multi-party trust; provenance; regulatory auditability	Immutable history; non-repudiation; shared truth across orgs	Throughput/latency overhead; privacy/governance tension; analytics externalized	Supply-chain traceability; ESG assurance; regulator-grade decision logging	[16], [17], [18], [19], [41], [67], [75]

Table 2: Role of Each Database Class in an AI-Driven Production Pipeline

AI / System Layer	Typical Responsibility	Best-Fit Database Class(es)	Why This DB Class Fits	Example Scenario in Production	Refs
Source-of-record / transactional truth	Orders, payments, lab results, compliance decisions, identity/access	Relational / SQL	ACID, referential integrity, auditability	Batch #8421 moisture=14.2% certified at Mill X must be the legal truth	[1]-[4], [42]
High-volume telemetry / operational monitoring	Sensor streams, GPU temps, latency, throughput, error rates	Time-Series	Append-optimized ingestion; sliding-window queries; retention/downsampling	Alert if tanker > 45Â°C for >10 min; SLA alarms for model serving	[7], [8], [45], [50], [51], [52]
Low-latency inference features / risk	Fraud features, rolling counters, per-	In-Memory (Redis/HANA/TimesTen/Ignite)	RAM-resident lookups; microsecondâ€“ms	Payment fraud scoring inline with	[9], [55], [56],

scores / personalization	user embeddings, rate limits		latency; high throughput	authorization; rank content < 20ms	[57], [58], [59], [60]
Flexible app state / operational notes / edge capture	QC notes, inspection photos metadata, comments, chat transcripts	Document (MongoDB/Couchbase/Firestore)	Schema-flexible JSON; nested indexes; mobile/offline sync	Field inspector uploads handling notes with intermittent connectivity	[6], [10], [24], [61], [62]
Semantic retrieval and RAG grounding	Find top-k relevant internal evidence for a query	Vector DB (Milvus/Pinecone/Weaviate/Qdrant/pgvector)	ANN over embeddings; hybrid filters; multi-tenant isolation	LLM assistant grounded on SOPs, contracts, and lab reports	[11], [12], [14], [15], [46], [69], [70], [71], [72]
Relationship reasoning / provenance / explainability	Who touched this batch? Why is it high risk?	Graph DB (Neo4j/Neptune/JanusGraph/Cosmos Graph)	First-class edges; fast multi-hop traversal; explicit reasoning paths	Supply-chain lineage; fraud-ring detection; knowledge graph justifications	[13], [63], [64], [65], [66], [67], [68]
Immutable audit trail / cross-organization trust	Custody logs, certifications, regulator-facing evidence	Blockchain / Ledger DB	Cryptographic immutability; non-repudiation; shared truth	Fair-payment assurance and ESG traceability with third-party audits	[16], [18], [41], [67], [75]
Historical analytics, forecasting, feature engineering	Large-scale aggregation, BI dashboards, offline training features	Columnar Warehouse (ClickHouse, Redshift, Snowflake, BigQuery)	Columnar compression; vectorized scans; lake/warehouse integrations	Compute ML features from months of telemetry + QC outcomes	[5], [22], [25], [47], [48], [49]

Table 3: Open Research Challenges

Research Challenge	Why It Matters	Affected DB Classes	Current Gap / Open Problem	Refs
Millisecond inference vs. infrastructure cost	RAM caches, GPU ANN search, replication deliver speed but are energy- and cost-intensive at scale	In-Memory; Vector; (plus serving layers)	Need adaptive tiering (RAM/NVMe/cold), quantized embeddings, hybrid CPU/GPU planners, cost-aware schedulers	[11], [12], [14], [15], [28], [29], [55], [56], [58], [59], [60], [69], [70]
Freshness and consistency of semantic indexes	RAG quality and policy safety depend on syncing text â†' embeddings â†' ANN indexes and permissions	Vector DBs (+ Document + SQL source of truth)	Lack atomic updates for embedding + index + ACL; need near-real-time incremental rebuilds and provable â€œno-stale-contentâ€ guarantees	[11], [12], [26], [27], [69], [71], [72]
Provenance, auditability, and regulator-facing explanations	High-stakes domains require explainable answers backed by tamper-evident lineage	Graph; Ledger; Relational (legal record)	Hard to unify causal paths (graph) with cryptographic proofs (ledger) and structured compliance (SQL) into one auditable pipeline	[16], [18], [41], [66], [67], [68], [75]
Governance for converged / hybrid databases	Postgres and Mongo now act as relational + doc + time-series + vector platforms	Relational (Postgres+JSON+pgvector); Document (Mongo+ACID+TS+vector)	Need consistent, auditable access/retention/versioning across rows, JSON, embeddings, streams inside one logical engine	[23], [24], [25], [45], [46], [61], [62]
Sustainability and energy efficiency (edge + cloud)	Caches, GPU vector search, blockchain replication, and big scans consume significant energy	In-Memory; Vector; Ledger; Columnar; Time-Series (edge)	Research into compressed/approximate retrieval, edge buffering + delayed sync, carbon-aware analytics scheduling	[28], [29], [52], [54], [58], [59], [60], [69], [70], [75]