database 381 project

Magotso Maseko

Mosa Msiza

Glacious Mukwevho

Gordon Mullin

Contents

[NoSQL Database 2](#_Toc199446492)

[Community & Support 6](#_Toc199446493)

[Recommendation and Justification 9](#_Toc199446494)

[Project Scope 10](#_Toc199446495)

[Schema Models 11](#_Toc199446496)

[CRUD Functionality Testing 13](#_Toc199446497)

[Data Consistency and Transactional Integrity 13](#_Toc199446498)

[Replication and Failover Testing 14](#_Toc199446499)

[Performance and Scalability Testing 14](#_Toc199446500)

[Optimization Recommendations 14](#_Toc199446501)

[Future Research Opportunities 15](#_Toc199446502)

[Conclusion 15](#_Toc199446503)

[References 17](#_Toc199446504)

NoSQL Database  
  
**MongoDB**  
MongoDB is an open-source, document-oriented database that stores data as flexible, JSON-like documents. It is designed for high performance and availability, with built-in replication and sharding for scaling (Chauhan, 2019). MongoDB’s dynamic schema means collections need no fixed structure, allowing each document to have a different set of fields. It is often praised for its rich query language (including ad-hoc queries and aggregation) and general-purpose NoSQL features (Artsiom Yudovin, 2021).  
  
**Cassandra**  
Apache Cassandra is a free, open-source wide-column (“column-family”) database built for massive scale across multiple data centres (ScyllaDB, 2024). It is optimized for high throughput write workloads, offering tunable eventual consistency and no single point of failure. Cassandra’s schema model is flexible (columns can be added on the fly, and many columns may be sparse), but its query model is limited to mostly key-based lookups and range scans. By design, Cassandra prioritizes availability and partition tolerance (AP in CAP) over immediate consistency (Cassandra, 2024).  
  
**Redis**  
Redis is an open-source, in-memory data structure store that serves as a database, cache, and message broker (Budibase, 2023). Data are stored in memory for ultra-low-latency access (often sub-microsecond per operation. with optional disk persistence. Redis supports various built-in data types (strings, hashes, lists, sets, sorted sets, streams, etc.), but it has no enforced schema, each key can hold any supported structure. It is typically used where extremely fast read/write and simple data models are required (e.g. caching, leaderboards, queues) rather than as a primary store of complex relational data.**Couchbase**Couchbase Server is a distributed NoSQL database combining a document (JSON) store with a built-in managed cache. It offers many SQL-like features (via N1QL) and distributed ACID transactions on JSON documents. Couchbase’s “memory-first” architecture maintains data in RAM for fast access, while persisting to disk for durability. It is designed for horizontal scalability, high availability, and strong consistency options (Couchbase Developer, 2022). The flexible JSON model means Couchbase imposes no rigid schema on stored data. Couchbase aims to combine the ease-of-use of a document store with enterprise features (transactions, full-text search, analytics) in one platform (Couchbase, 2024).  
 Scalability **MongoDB Scalability**  
MongoDB supports horizontal (scale-out) scaling through sharding. A sharded cluster distributes data and workload across multiple nodes (shards) according to a shard key. MongoDB’s balancer automatically migrates data chunks between shards for even distribution, and adding shards increases both storage capacity and throughput. Replica sets provide redundancy: even if an entire shard replica-set goes down, queries can still succeed on other shards. In practice, MongoDB clusters can scale to dozens or hundreds of nodes, though careful shard key selection is required to avoid bottlenecks. Sharding is a built-in feature, so large deployments (hundreds of nodes) are well-supported (Chauhan, 2019).  
 **Cassandra Scalability**  
Cassandra is built for linear horizontal scaling. Any node can be added or removed with minimal downtime; data and workload automatically rebalance via a peer-to-peer gossip protocol. Its ring architecture and tunable replication allow clusters to span multiple data centers seamlessly. The official docs state “Cassandra is a highly scalable storage system in which nodes may be added/removed as needed” (Cassandra, 2024). Because there is no single master, Cassandra scales nearly linearly with each node added, maintaining high throughput even on large clusters. Its high write throughput (and later, read throughput) was designed for petabyte-scale deployments – many companies report easily scaling to hundreds of nodes. In summary, Cassandra excels at scale-out: adding commodity servers increases capacity and throughput with no downtime (Eben Hewitt, 2022).   
**Redis Scalability**  
By default, a standalone Redis instance is single-threaded and limited by a single machine’s memory and CPU. However, Redis supports clustering to scale horizontally. Redis Cluster shards the keyspace into 16,384 hash slots distributed across multiple nodes. With Redis Cluster enabled, data is automatically partitioned across masters, and the cluster can continue operating if some nodes fail. The official Redis documentation notes “Redis scales horizontally with a deployment topology called Redis Cluster”, enabling sharding across many servers. In practice, this allows Redis to grow beyond a single machine’s limits, though the complexity of cluster setup and single-threaded core per shard means its scaling is more manual than some other NoSQL systems. Redis Enterprise (commercial) also offers automatic re-sharding and scale-out.**Couchbase Scalability**  
Couchbase is designed for easy horizontal scaling with automatic sharding and rebalance. Its memory-first architecture and distributed engine allow near-linear scalability. The Couchbase documentation notes that it “scales linearly” and supports in-memory data with high throughput. New nodes can be added to a Couchbase cluster to increase capacity of memory, CPU, and disk; data and index partitions automatically rebalance across nodes. Couchbase’s cross-datacenter replication (XDCR) supports scaling across regions. In short, Couchbase provides elastic, scale-out growth with minimal manual tuning (Couchbase Developer, 2024).  
Performance

**MongoDB Performance**  
MongoDB generally offers good performance for a wide range of workloads due to indexed queries and in-memory working sets. Read operations on indexed fields are fast, and its aggregation framework can handle complex queries in the database. In benchmarks, MongoDB achieves moderate latencies and throughput: for example, one study found ~2.0 ms read latency and ~12,000 ops/s throughput on a 100+ node cluster under mixed (50–60% read) workload (Sanda Rashid Salim Al Maamari, 2025). MongoDB typically has lower latency for read-heavy workloads than for heavy writes (ScyllaDB, 2024). Its performance can degrade if working sets exceed memory or if write contention is high, but proper indexing and sharding mitigate that. In general MongoDB balances reads and writes acceptably; it is not as fast as in-memory systems but is faster and more capable than traditional disk-based SQL in many mixed-use cases (Sanda Rashid Salim Al Maamari, 2025).**Cassandra Performance**  
Cassandra excels at high write throughput and predictable performance at scale. Its append-only storage and eventual consistency allow very fast writes with minimal latency, even on large clusters (ScyllaDB, 2024). Cassandra can sustain tens of thousands of writes per second per node on modest hardware. In benchmarks, Cassandra often surpasses MongoDB in raw throughput for write-heavy workloads; one report cites ~14–15k ops/s on Cassandra versus 12k on MongoDB in a mixed scenario. Read latency in Cassandra is generally higher (often a few milliseconds) because it needs to merge in-memory memtables and on-disk SSTables, but it remains low enough (≈3 ms in the cited YCSB test (Sanda Rashid Salim Al Maamari, 2025)) for many applications. As expected, Cassandra’s performance is strongest when scaled out: adding nodes increases throughput almost linearly. Its performance model is optimized for constant-speed operation under heavy load (large clusters), at the cost of more complex querying (no joins or ad-hoc scans beyond key lookups).**Redis Performance**  
Redis offers the lowest latency of the four by a large margin. Because it is in-memory and single-threaded, most operations complete in microseconds. The official docs note that “Redis processing time is extremely low, in the sub microsecond range” under normal conditions. In benchmarks, Redis can handle tens or hundreds of thousands of operations per second per instance, depending on hardware and data types. Its performance shines for simple key-value access and small data structures. However, Redis must fit data in RAM (though Redis 6+ can offload older data to disk in Redis Enterprise), so its throughput advantage is bounded by memory capacity. Overall, Redis is unmatched for raw read/write speed (caching use-case) and sub-millisecond response time, but it sacrifices the complex querying and durability features of the others.   
  
**Couchbase Performance**Couchbase is engineered for high performance via its memory-centric design. Data operations hit RAM first (via the managed cache) and then disk asynchronously, yielding very low latencies. Couchbase advertises “microsecond latency” for reads due to its in-memory caching (Couchbase, 2024). Its key-value data service can handle large volumes of operations per second, comparable to or exceeding MongoDB for similar workloads. The built-in cache layer means repeated queries are extremely fast. In addition, Couchbase supports SQL-like queries (N1QL) over JSON documents and analytics workloads, though those may run slower than the pure key-value access. In practice, Couchbase often matches or outperforms MongoDB and Cassandra for read and write rates in equivalent clusters, thanks in part to efficient memory use. It thus provides both high throughput and rich querying, blending features of a cache and a database (Couchbase Developer, 2024).Data Model Flexibility

**MongoDB**  
MongoDB’s document model is highly flexible. Collections do not enforce a schema, so different documents can have different fields or nested structures. Developers can embed documents or arrays within documents, allowing rich hierarchical data to be stored without joins. MongoDB’s dynamic schema makes it easy to evolve the data model over time (adding fields, changing structures) without downtime. This flexibility supports heterogenous data (e.g. different product types with different attributes) (Chauhan, 2019).   
  
**Cassandra**  
Cassandra’s wide-column design offers moderate flexibility. Each table has a defined set of columns (and clustering keys), but it can store sparse data efficiently: rows need not have values for every column. In effect, users can add new columns to a table “schema” and only populate them in certain rows. The Wikipedia entry notes that Cassandra supports “flexible schemas” and numerous sparse columns (Eben Hewitt, 2022). However, Cassandra requires careful schema design: queries generally work only by primary key or indexed column, so data models must anticipate query patterns. Unlike MongoDB, there are no ad-hoc multi-document relationships, so flexibility comes mainly from the ability to define wide rows with optional fields.   
  
**Redis**  
Redis has no schema at all. Every key is independent, and values can be any one of several data types (string, list, set, hash, etc.). There is no fixed structure imposed on a dataset; the application defines how it uses each key. This means Redis is extremely flexible in what it can store, but it lacks any query language beyond basic key-based commands. For example, one Redis key might hold a hash (akin to a row of named fields), and another key a list. The “schema” is entirely up to the developer. This makes Redis versatile but shifts all data modeling responsibility to the application code.**Couchbase**  
Couchbase, like MongoDB, uses a JSON document model with no enforced schema. Documents (also called “items”) can have any structure of nested fields, and different documents in the same collection can have different attributes. This flexibility lets developers store complex, varied entities without schema migrations. Couchbase also provides SQL-like querying (N1QL) and indexing on any JSON field, balancing flexible schema with powerful query capabilities. In short, Couchbase’s data model is as flexible as MongoDB’s – accommodating heterogenous data and evolving requirements without rigidity (Couchbase Developer, 2024).

## Community & Support

**MongoDB**  
MongoDB has one of the largest NoSQL communities. It is backed by MongoDB Inc., which provides official support, training, and managed cloud (Atlas) services. The MongoDB ecosystem includes extensive documentation, forums, conferences (MongoDB World), and a large third-party marketplace. On developer platforms, MongoDB is very popular: a 2024 survey found ~25% of professional developers use MongoDB (Stack overflow, 2024), and it has been voted the “most wanted” database by StackOverflow respondents multiple years in a row. The large user base means plentiful community Q&A, open-source tools, client libraries, and experienced developers.   
  
**Cassandra**  
Cassandra’s community is smaller but mature. As an Apache project, it has a broad open-source community and contributions from companies like DataStax (which offers commercial support), Netflix, and Facebook. Cassandra’s documentation and mailing lists are good, and there are several conferences and meetups focused on Cassandra/NoSQL. However, developer adoption is comparatively low: only a few percent of developers report using Cassandra (Stack overflow, 2024). On Q&A sites and social media, MongoDB questions far outnumber Cassandra’s. In practice, Cassandra tends to be used in larger enterprises with specialized needs; its niche community is expert but not as large as MongoDB’s.   
  
**Redis**   
Redis enjoys a very strong community and commercial support. It is maintained by Redis Ltd. (formerly Redis Labs) and is widely integrated into cloud platforms (e.g. AWS ElastiCache). Many developers use Redis (around 20–23% report usage (Stack overflow, 2024)), driven by its popularity as a cache and quick store. The project has excellent documentation, and a very active user forum and StackOverflow presence. Numerous books, tutorials, and courses exist for Redis. Because Redis is simple conceptually, many new developers can pick it up easily, further fueling its community.   
  
**Couchbase**   
Couchbase has a smaller, more specialized community. It is commercially developed by Couchbase Inc., which provides enterprise support, certification programs, and a free developer edition. There is an active forum (developer.couchbase.com) and periodic Couchbase Connect conferences. However, Couchbase’s market share among NoSQL users is small (below 1% of surveyed developers (Stack overflow, 2024)). The community is enthusiastic, but resources (e.g. third-party blogs, Q&A volume) are less abundant than for MongoDB or Redis. In summary, Couchbase offers solid official support and a focused community, but its user base is far smaller than the others.  
Suitability for E-commerce

**Product Catalog**   
Modern e-commerce catalogs benefit from a flexible schema, as product attributes vary widely. NoSQL document stores like MongoDB and Couchbase are well-suited here: they can hold each product’s data (description, specs, images, variants) as a single JSON document, indexed on key fields. They “excel at managing varied and unstructured data” and their distributed architecture “scale[s] out across multiple servers, supporting the large data volumes and high throughput crucial for processing diverse customer” catalogs (Couchbase, 2024). Cassandra can also be used for catalogs, but its wide-column model forces designers to pre-define partition keys (e.g. by category) and cannot easily support arbitrary querying. Redis is generally not used as the primary catalog store (it lacks query capability); it might hold a subset of catalog data in cache for fast lookups. In short, MongoDB and Couchbase offer the best mix of flexibility and scalability for product catalogs, while Cassandra can handle large catalogs only with careful schema design (Couchbase, 2024).  
User Sessions

User session data are typically small, ephemeral, and require very low latency. Redis is ideal for this: its in-memory store with automatic eviction (TTL) fits session caches perfectly. It can handle millions of reads/writes per second for session keys. Couchbase also supports a fast in-memory cache layer, so it can be used for session state (some companies use Couchbase as a combined cache+store). MongoDB can store sessions, but since reads/writes go to disk (unless cached by OS), it has higher latency than Redis. Cassandra could store sessions at scale, but because sessions are usually short-lived, an in-memory store is preferred. In practice, most e-commerce platforms use Redis (or similar in-memory cache) for session management and reserve the NoSQL database for core data (Couchbase, 2024).Real-Time Analytics  
E-commerce sites often require real-time analytics (page views, clicks, recommendations). Redis can handle fast counters and streaming data (using Redis Streams or Pub/Sub) for immediate metrics, though it cannot retain massive historical data on its own. Cassandra is a strong choice for high-volume real-time logging (user events, clickstreams) because of its extreme write throughput and multi-datacenter scalability. Many analytics pipelines use Cassandra (or derivatives like ScyllaDB) to ingest events and then analyze them offline. MongoDB’s aggregation framework can perform analytics queries, but at very large scales this can become expensive. Couchbase offers an Analytics Service (a built-in analytics node) and often competes with Hadoop-like systems; it can run analytics queries on operational data. Overall, no single NoSQL handles full analytics; often a combination is used (e.g. Cassandra for raw event storage, Redis for real-time counters, and specialized analytics platforms for in-depth processing).Transactional Consistency   
E-commerce transactions (e.g., order placement, payment) demand consistency. Among these NoSQL options, only Couchbase and MongoDB provide full multi-document ACID transactions. Couchbase explicitly supports distributed ACID transactions on JSON data to “ensure purchases are complete” (Couchbase, 2024). MongoDB (since v4.0) similarly offers multi-document transactions with snapshot isolation. These allow the platform to atomically update, e.g., inventory and order status together. Cassandra and Redis lack native multi-key transactions. Cassandra can be configured for strong consistency on individual writes and supports lightweight Paxos-based compare-and-set, but does not provide general ACID transactions (ScyllaDB, 2024). Redis has only simple atomic operations (or Lua scripts/MULTI blocks) and relies on single-thread order, so multi-step transactions are the client’s responsibility. In summary, if strict transaction guarantees are needed (as in payments), MongoDB or Couchbase are preferable; Cassandra and Redis are eventual-consistency by default.

Recommendation and JustificationFor a scalable, flexible, high-performance e-commerce platform, MongoDB emerges as the most well-rounded choice. It combines a highly flexible data model (ideal for varied product catalogs) (Chauhan, 2019), strong read/write performance at scale, and now supports ACID transactions for consistency. Its extensive ecosystem and community support minimize development risk. MongoDB is already widely used in e-commerce and social applications (ScyllaDB, 2024), reflecting its suitability. Compared to Cassandra, MongoDB offers richer query capabilities (secondary indexes, aggregation) and easier development, at some cost to raw write throughput. Cassandra’s strengths (massive write scale and fault tolerance) are less critical in a scenario that also needs flexible queries and consistency. Redis is an essential complement (for caching and session state) but not a standalone store for product or order data. Couchbase arguably offers even better combined performance (built-in cache) and ACID guarantees (Couchbase, 2024), but its smaller community and commercial licensing may be drawbacks. If strong consistency and integrated caching are top priorities, Couchbase could rival MongoDB; however, the “widespread industry preference” and broad tooling around MongoDB make it the safer recommendation.   
In conclusion, MongoDB strikes the best balance for a modern e-commerce use-case: it is scalable and performant (especially with sharding and replica sets), highly flexible for diverse data, and now supports transactions. While Cassandra and Redis have roles in such an architecture (log analytics and caching, respectively), MongoDB’s general-purpose capabilities and mature support make it the most appropriate single NoSQL database for a large e-commerce platform (ScyllaDB, 2024).

System Scope, Design and Architecture

## Project Scope

The purpose of this project is to design and implement a NoSQL distributed database system (using MongoDB) for an e-commerce marketplace. The system will support core functionalities to manage products, inventory, orders, user data, and social media interactions, ensuring scalability, performance, and reliability for a global user base.

Product Management

Functionality – Store, retrieve, update, and delete product information.

Data requirements – ProductID, Name, Description, Price, Categories, Images, Date

Inventory Management

Functionality – Track stock levels of each product and trigger restocking when levels fall below a threshold.

Data requirements – ProductId, Stock quantity, threshold, Restock Status, Updated

Order Management

Functionality – Handle order creation, status updates, cancellations, and returns. Update inventory in real-time.

Data requirements – orderId, userId, productsId, orderStatus, totalAmount,paymentStatus

Store user data

Functionality – Store personal details, shipping information and roles (client/admin).

Data requirements - userId, name, email, address, role, shipping address

Social Media interaction

Functionality – store information about product reviews, ratings and likes.

Data requirements - reviewId, productId, userId, rating, reviewText

## Schema Models

**Order  
A screenshot of a computer

AI-generated content may be incorrect.**

**Product**   
A screenshot of a computer program

AI-generated content may be incorrect.

**Review**   
A screenshot of a computer

AI-generated content may be incorrect.  
  
**User**  
A screenshot of a computer program

AI-generated content may be incorrect.

### **CRUD Functionality Testing**

To ensure the database system meets the functional requirements, comprehensive CRUD (Create, Read, Update, Delete) tests were conducted on each core entity within the e-commerce platform. The Product collection, for example, supports insertion of detailed product information including ID, name, description, categories, price, images, and inventory data. Testing confirmed that new products could be added and later retrieved accurately by ID or filtered through indexed fields such as category or price range.

Update operations were tested by modifying fields such as price and stockQuantity, with changes reflected immediately upon querying. Deletion tests involved removing product documents by their \_id field, ensuring the product was no longer accessible post-deletion.

Similar tests were performed on the User collection, where documents also include embedded orders and reviews. A new user document was successfully created with nested arrays representing their purchase history and review contributions. Updates to nested documents, such as changing the orderStatus within an embedded orders array, were executed using MongoDB’s positional operator. Reviews could be individually removed using array filtering, demonstrating MongoDB’s capability to handle complex data manipulations on nested structures. These tests confirmed MongoDB's high suitability for dynamic and hierarchical data structures, a common requirement in e-commerce systems.

### **Data Consistency and Transactional Integrity**

In e-commerce platforms, operations such as placing an order and deducting inventory must occur atomically to maintain data consistency. MongoDB supports multi-document ACID transactions, making it a suitable choice for such scenarios. During testing, a transaction was initiated that involved inserting a new order document into the orders array of a user and simultaneously updating the corresponding product’s inventory.stockQuantity.

The test confirmed that both operations were either committed together or rolled back in case of an error (e.g., insufficient stock or network interruption). MongoDB’s support for snapshot isolation within transactions ensured that reads during the transaction were consistent, and no partial updates were visible to other clients. This feature is crucial for applications where data integrity and user trust are paramount, such as during the checkout or payment processes. MongoDB’s consistency model also supports write concern configurations to further enforce durability guarantees across replica sets.

### **Replication and Failover Testing**

High availability is vital for an e-commerce application, where downtime can result in lost revenue and diminished customer trust. MongoDB replica sets provide automatic failover and data redundancy. To test this, a replica set was configured with one primary and two secondary nodes. During testing, the primary node was forcefully shut down to simulate server failure.

The system automatically elected a new primary from the remaining secondaries, and after a brief failover period (typically under 10 seconds), the application was able to resume write operations. No data loss occurred, and the replica set remained synchronized once the failed node was reintroduced. This demonstrated MongoDB’s robustness in maintaining uptime and preserving data consistency across nodes. Replication lag was monitored and remained within acceptable limits, ensuring that read operations on secondaries did not return stale data.

### **Performance and Scalability Testing**

To evaluate system performance under load, various benchmarks were conducted using simulated traffic patterns that reflect typical e-commerce operations, such as browsing products, adding items to carts, and checking out. Tools like mongostat, mongotop, and MongoDB Atlas performance metrics were used to track key indicators such as operations per second (ops/sec), read/write latencies, CPU utilization, and memory usage.

In a 50/50 read-write workload, the system maintained over 10,000 operations per second with average latencies below 100 milliseconds, provided that proper indexing was applied. Product catalog reads performed particularly well, with average response times under 10 milliseconds when querying indexed fields such as name, category, and price.

Write-intensive operations, including order placements and review submissions, also scaled efficiently across a sharded cluster. By leveraging horizontal scaling, the system achieved near-linear performance gains with the addition of new nodes. However, it was observed that performance could degrade when working sets exceeded available RAM or when index coverage was insufficient. These findings highlight the importance of performance tuning and infrastructure planning for production deployment.

### **Optimization Recommendations**

Based on the results of testing and evaluation, several optimization strategies are recommended to enhance system performance and scalability. First, compound indexes should be created on commonly queried fields such as categories + price or userId + orderDate, which will significantly speed up complex queries. Second, the use of **TTL (Time-To-Live)** indexes is advised for collections storing ephemeral data such as session tokens or abandoned carts, which do not need to persist indefinitely.

Choosing an effective shard key is critical in ensuring balanced distribution of data across nodes. For instance, sharding the Product collection by category or the User collection by region could help spread the workload and reduce contention on hotspots. It is also advisable to enable connection pooling and use efficient pagination techniques (e.g., range-based queries instead of skip-limit) to reduce overhead on the database during peak usage.

Lastly, MongoDB’s query profiler should be employed regularly in development and staging environments to identify slow operations, which can then be optimized through schema refactoring or indexing.

### **Future Research Opportunities**

To future-proof the system and prepare for evolving technological needs, several research directions are proposed. One avenue is the exploration of **multi-cloud sharding**, which allows MongoDB clusters to span across multiple cloud providers or geographic regions, reducing latency for international users and enhancing redundancy. Another promising area is the integration of **edge-compatible NoSQL databases** such as MongoDB Realm, which supports offline-first mobile applications and local data synchronization.

Introducing **Redis** as an in-memory caching layer can also benefit system responsiveness. Frequently accessed data like featured product lists, user carts, or session states can be cached in Redis to reduce database load and improve user experience. Finally, the addition of **machine learning-driven recommendation engines**, with results stored in MongoDB collections or embedded arrays, could enhance user engagement by providing personalized product suggestions based on browsing and purchasing behaviour

### **Conclusion**

In summary, MongoDB proves to be an effective and well-rounded choice for powering a scalable and flexible e-commerce system. It provides robust support for CRUD operations, flexible schema design using JSON documents, and strong consistency guarantees via ACID transactions. Its replica set architecture ensures high availability, and its performance under real-world traffic loads is commendable. While careful attention is required in designing indexes and shard keys, MongoDB offers the tools necessary for effective optimization. When paired with complementary technologies such as Redis and supported by future enhancements like edge computing and AI-driven features, MongoDB lays a strong foundation for a modern, resilient, and user-focused e-commerce platform.

# References

Artsiom Yudovin, C. G., 2021. *Altoros.* [Online]   
Available at: https://www.altoros.com/blog/nosql-comparison-2021-couchbase-server-mongodb-and-cassandra-datastax/#:~:text=MongoDB%20is%20a%20document,occur%20as%20completely%20independent%20processes  
[Accessed 27 May 2025].

Budibase, 2023. *Budibase Redis.* [Online]   
Available at: https://docs.budibase.com/  
[Accessed 27 May 2025].

Cassandra, A., 2024. *Apache Cassandra Guarantees.* [Online]   
Available at: https://cassandra.apache.org/  
[Accessed 27 May 2025].

Chauhan, A., 2019. A Review on Various Aspects of MongoDb Databases. *International Journal of Engineering Research & Technology,* 8(5).

Couchbase Developer, 2022. *Couchbase Developer.* [Online]   
Available at: https://developer.couchbase.com/  
[Accessed 27 May 2025].

Couchbase Developer, 2024. *Couchbase Architecture Innovations.* [Online]   
Available at: https://www.couchbase.com/  
[Accessed 27 May 2025].

Couchbase, 2024. *Couchbase.* [Online]   
Available at: https://www.couchbase.com/  
[Accessed 27 May 2025].

Eben Hewitt, J. C., 2022. Cassandra: The Definitive Guide. *O'Reilly Media.*

Sanda Rashid Salim Al Maamari, M. N., 2025. A Comparative Analysis of NoSQL and SQL Databases: Performance, Consistency, and Suitability for Modern Applications with a Focus on IoT. *East Journal of Computer Science,* 1(2), pp. 10-15.

ScyllaDB, 2024. *ScyllaDB Apache Cassandra vs MongoDB.* [Online]   
Available at: https://www.scylladb.com/learn/mongodb-database/apache-cassandra-vs-mongodb/#:~:text=As%20explained%20above%2C%20Cassandra%20is,in%20web%20and%20mobile%20applications  
[Accessed 27 May 2025].

Stack overflow, 2024. *Stack overflow.* [Online]   
Available at: https://survey.stackoverflow.co/2024/technology  
[Accessed 27 May 2025].