Database architecture: A dot framework research paper

A Software Engineering Study Project

Yardi van Nimwegen
Complex software semester 6
Fontys University of applied sciences
2025-10-24

Document Version History

Version	Date	Author	Description
1	21-10-2025	Yardi van Nimwegen	Initial draft created
2	23-10-2025	Yardi van Nimwegen	First version finished
3	24-10-2025	Yardi van Nimwegen	Improving the how part of the dot framework chapter

Table of Contents

Contents

Introduction

Modern streaming application such a music streaming platform have multiple architectural challenges which are very important to create a good streaming platform. These are: the need for a scalable catalog of stream-able content and the requirement of low-latency discovery of said catalog.

Problem statement. The current system is attempting to fulfill the search requirement by using apache Cassandra with a song_by_title table, as is normal with Apache Cassandra this table has a partitioned primary key. The primary key has a partition on the title prefix which are the first three letters of the song name. This is to make the search for songs faster since looking for the prefix and then in another column after the prefix the whole title of a song.

But when assessing this further i though what if there were extremely many songs with the same prefix and the partition gets too large. Since the maximum size that should be aimed for with a Cassandra partition is 100MB but ideally should be 10MB[2]. To counter this issue there was looked into best practices for data modelling within Cassandra and bucketing[3] is a way to keep partitions smaller by using another column in the database to partition there as well as with the main column a partition has been created upon to. Within this project there was the idea to create an x amount of buckets per prefix so specific prefixes such as ‘lov’ will not become too large to follow the ideal partition sizes to keep the reading speed high.

However, by applying the buckets on every prefix i would cause fan-out reading[4]. Now when a user would search a song with a very common prefix. Every single bucket of the prefix would have to searched through to find the song and all buckets have to return a result which would cause massive overfetching.

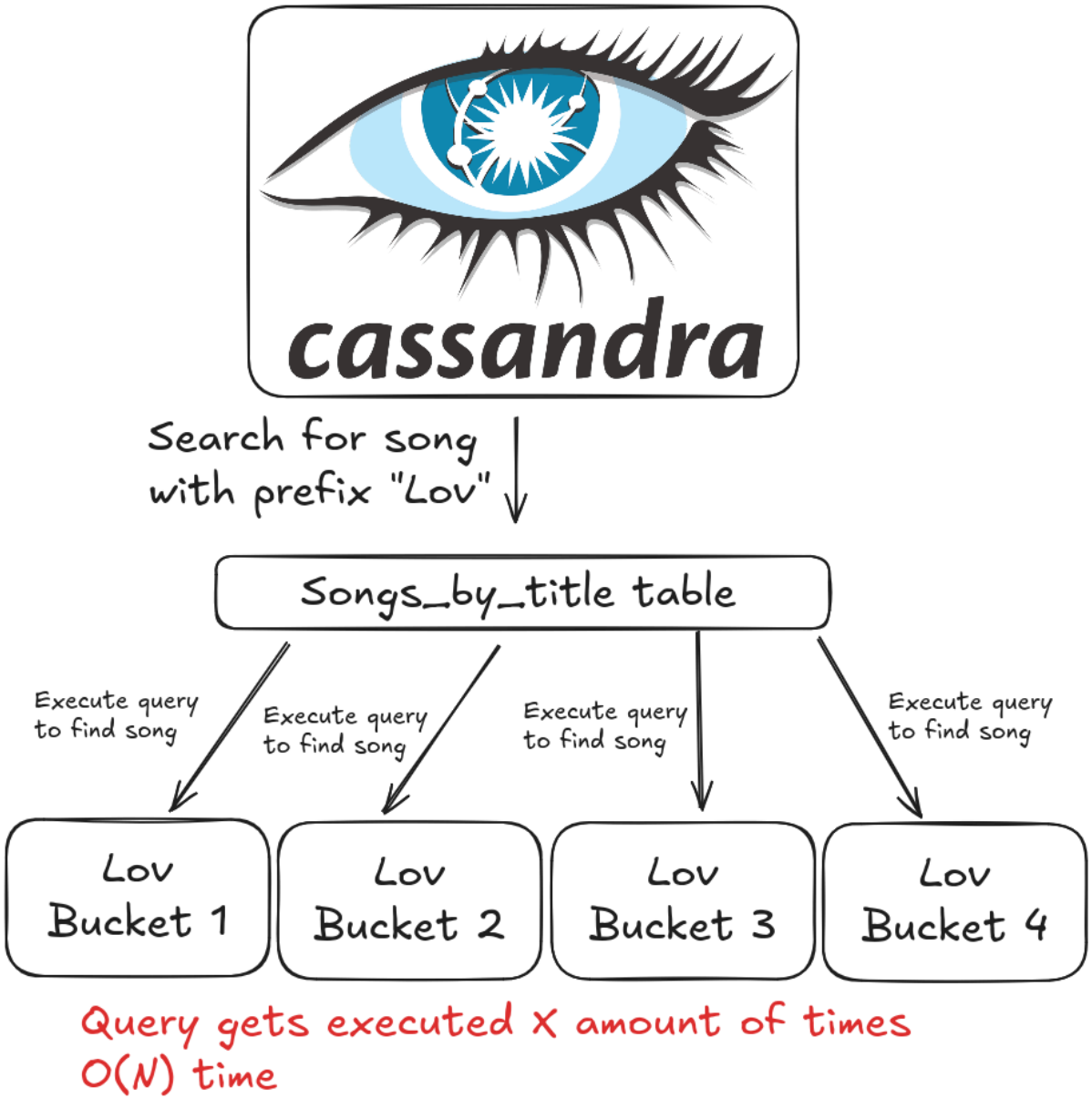

Figure 1: What happens in a fan out read

Cassandra is fundamentally designed as a distributed wide-column database, which is optimized for high availability, and very high write throughput. It's amazing read performance relies on having very specific partition keys to be able to achieve great read speeds for the queries. The fan-out read design which has previously been thought of fundamentally violates this core principle, because of this it will be impossible to guarantee low latency queries and makes the system unsuitable for the low latency searching which is required for the music streaming platform.

Hybrid system justification

The technically superior and industry-validated solution will be adopting a hybrid architecture. This approach will apply the principle of separation of concerns[5], only using Apache Cassandra specifically for it's great write performance and storing all the records[6], and deploying a specialized application which is optimized for reading operations for high-performance indexing and querying of data[7].

Applying the DOT framework

To structure this justification for this architectural shift, the research is written using the three core levels of the DOT framework. This ensures that the advice that is given in this research is using the application context, available expertise and a data driven research.

Defining the research scope (the what). Within this research the goal is to bridge the gap between the application domain, which is the need to resolve the high-latency search bottleneck when truly working at scale. And the available work domain, which are already established ways to solve this problem. The research focuses on validating the principle of separation of concerns by splitting up the massive data storage in Cassandra and applying another technology which specializes in text search. At the end of this research there will be a recommendation which is the innovation domain.

Justifying the goals (the why). The research is driven by validating new architectural choices against trade-offs, ensuring that the final product is technically robust and implementable within the constraints the project has.

The objective is to achieve product fit by proving existing expertize, this means proving that the existing Cassandra implementation is inefficient and therefore cannot meet the requirements of the project and deliver a modern search experience.

Research strategies and methods applied (the how).

The main research strategies that were used within this research are the

1. Library research
   - methods:
     - Literature study: Research the theoritical limitations of Cassandra secondary indexes and fan-out reads
     - Available product analysis: analyzing the architecture of large streaming platforms wtih full text search such as Spotify
     - Design pattern research: Researching best practices for synchronizing data with change data capture
2. Workshop research
   - methods:
     - IT architecture sketching: Proposed a new hybrid data model where ElasticSearch handles the searching and Cassandra is used as the system of record.
     - Multi-criteria decision making: A key part of the new structure was a trade-off analysis which led to using the dual-write pattern instead of using the technically superior CDC because of the time constraints
3. Showroom research
   - methods:
     - guideline conformity analysis: Justifying the choice of using ElasticSearch based on it being close to best practices for implementing full-text search with inverted indexes and fuzzy finding. The new hybrid architecture showcases a big improvement in meeting the non functional requirements for search speeds compared to the initial design

Technical analysis

The original data model had a large issue. To find music it required scanning of multiple distinct partitions for every single search, which is a direct contradiction of Cassandra’s optimization techniques[8] and is known as fan-out reads[4].

The performance of Fan-Out reads. Cassandra can achieve high read speeds by routing a request directly to the nodes which are holding very specific partition keys[9]. But when these partition keys are no longer very specific it becomes harder for Cassandra to maintain it's speeds since now, it has to use a fan-out read which now requires the node to search through multiple locations to collect the required results.

This fan-out read increases multiple factors which cause the application to become more difficult to scale:

1. Latency multiplication: The query that is being ran in Cassandra will be limited by the slowest of all the read operations, and with popular prefixes which have large buckets connected to it, will cause degradation in achieving low-latency.
2. I/O: Cassandra stores data compressed so it takes less space but when trying to retrieve data the data needs to be uncompressed, so when scanning through all the buckets, each has to be decompressed which will cause a lot of unnecessary I/O .
3. Scalability limitation: This massive over-fetching and scanning through all the partitions will cause Cassandra nodes to be overloaded quicker and needing more nodes than is required if then when the searching is done more efficiently.

Cassandra secondary indexes. Technically the secondary indexes would be better than the initial implementation, so searching by the prefix and then the full-title of the song. But these are only local to the node that stores the data, if implementing this anyway and using the secondary index, it would cause a fan-out read across every single node running[10].

Architectural decoupling

The solution to resolve the issue of storing the metadata of songs efficiently, so it can be stored and retrieved efficiently and also have a good user experience for searching music with relevant results, would be to split up the responsibility and apply the separation of concerns principle and using two different systems which are specialized[5].

Defining system roles and new data-model. This research proposes a separation of concerns and using specialized tools for the requirements that are needed while trying to achieve the non-functional requirements

System	Primary role	Data model	Reason to choose
Cassandra	The system of record	Write-heavy, Full data access	Ensures high availability and fast retrieving of data based on by_id table to retrieve full data extremely fast
ElasticSearch	Search index	Read heavy, full text indexing	Provides low-latency, extremely fast searching through huge amounts of data with relevance scoring

This new proposed architecture would solve the current flawed model, Cassandra song metadata could be changed from songs_by_title to songs_by_id, the table could now be partitioned

by the unique songId, this would make Cassandra be used exactly as it is intended to be used: to have extremely fast and achieve linear-scaling lookups for the full meta-data of a song once a user has found the song it wants within ElasticSearch and clicks to stream it.

Why ElasticSearch. ElasticSearch is one of the industry standards for requiring full-text search[11]. This is because the enormous performance improvement is based in the data structure which is being used in ElasticSearch, the inverted index maps every single word which in this case are song titles and artist names in to a list of documents which contain this word, which makes searching for text extremely fast[12].

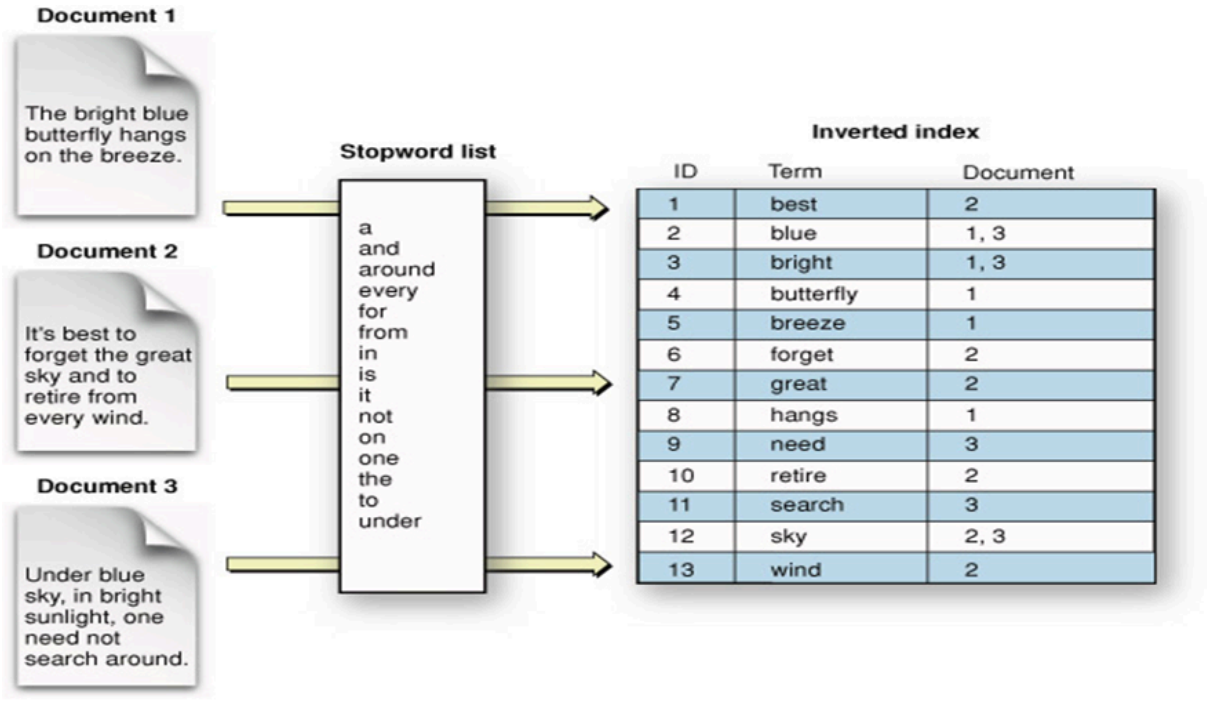

Figure 2: How an inverted index works

Because of this specific data structure it becomes possible to have more features which are very nice for the users:

- Fuzzy[13] and prefix searching: ElasticSearch can due to the data structure that they use handle complex user queries, such as typos and predictive search[14], which is impossible to implement efficiently within Cassandra due to how Cassandra works

Syncing data between databases

To make sure that users get correct data, it is very important that data is being synchronized between Cassandra and Elasticsearch.

Change data capture [1]. Change data capture (CDC) is a synchronization pattern which avoids embedding logic within the application to keep the databases synchronized. Instead of embedding code within the application to synchronize the database you have an external tool such as Debezium[15], these tools look at the source database's transaction log and then stream the changes to the other database.

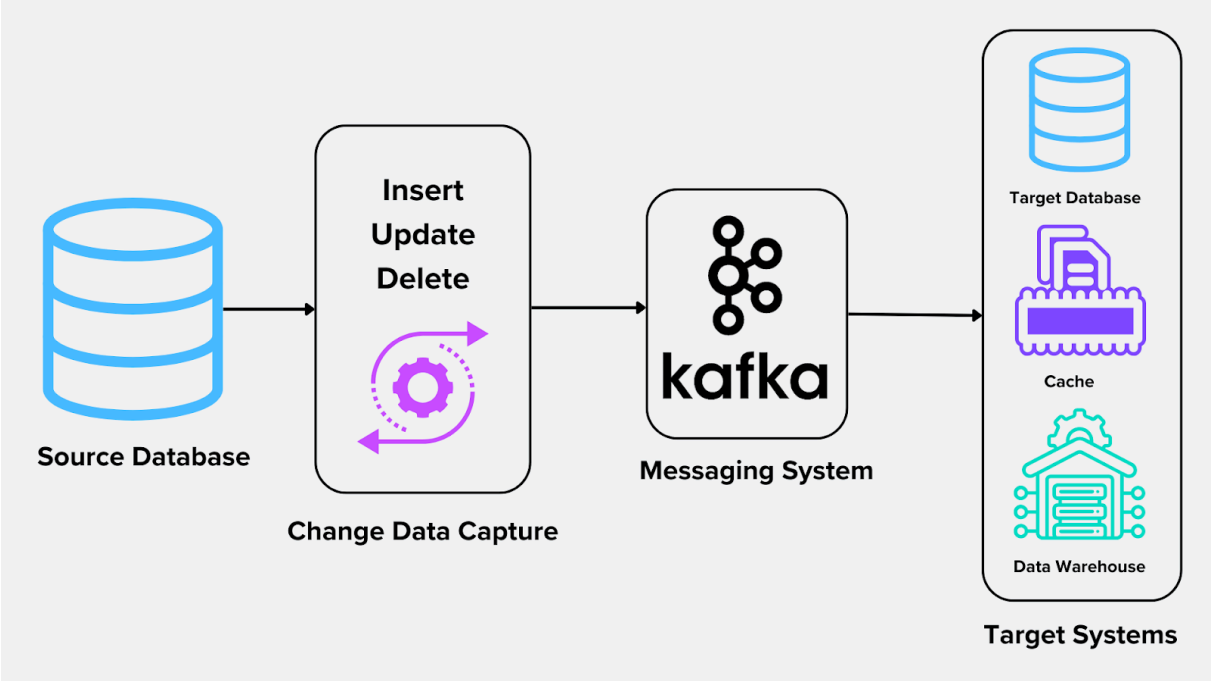

Figure 3: How a CDC works

How CDC works in Cassandra

1. Log monitoring: To enable CDC log monitoring in Cassandra there is requirement to set cdc=true for that specific table.[15]
2. Agent: A dedicated agent which is often a Kafka connect worker[15]
3. Event generation: The deployed agent reads changes (inserts, updates and deletes) and transforms these changes into a structured change event and publishes them into a message broken such as Kafka[16]
4. Reliability: This agent is tolerant for failures and keeps in mind how many changes it has already processed. It only deletes the log file after is has confirmed that all events in the log have been published[16]

Why CDC is the best for consistency. CDC is considered best practice when compared to other manual methods for synchronizing data between the multiple database that are implemented.

1. Decoupling: Synchronization logic will be completely separated from core business logic within the application. The application would simply just update data within Cassandra and the CDC tool would handle the rest of inserting, updating or deleting the data within ElasticSearch.
2. Guaranteed the same records: Unlike when building the synchronization logic within the application, the CDC will make sure all changes are captured directly from the source which is the transaction log.

Using dual-write pattern for synchronization

Having synchonization between the Cassandra system of record and ElasticSearch index is required for achieving the non-functional requirements. While the research above has proven that from a technical perspective the best option to implement synchonization between Cassandra and ElasticSearch would be best with CDC. In reality this is quite a complex set-up, and with the time constraints on this project it would take up more time which is also required for other implementations and can still be implemented if there is excess time.

Because of this reason, the dual-write pattern has been chosen to implement the synchronization between the Cassandra and ElasticSearch.

Risk and trade-offs of using dual-write. The reason why CDC is best for implementing this synchronization are also the reasons why dual-write is inferior[17].

1. Inconsistency: Since the writes happen separtely, if the write succeeds to Cassandra but the write to ElasticSearch fails for whatever reason possible, the databases will be permanently inconsistent since data is missing in one of the databases[17]
2. Increased latency: Since there now has to be written to two databases, this will of course increase the latency since two writes have to be performed instead of a single write.

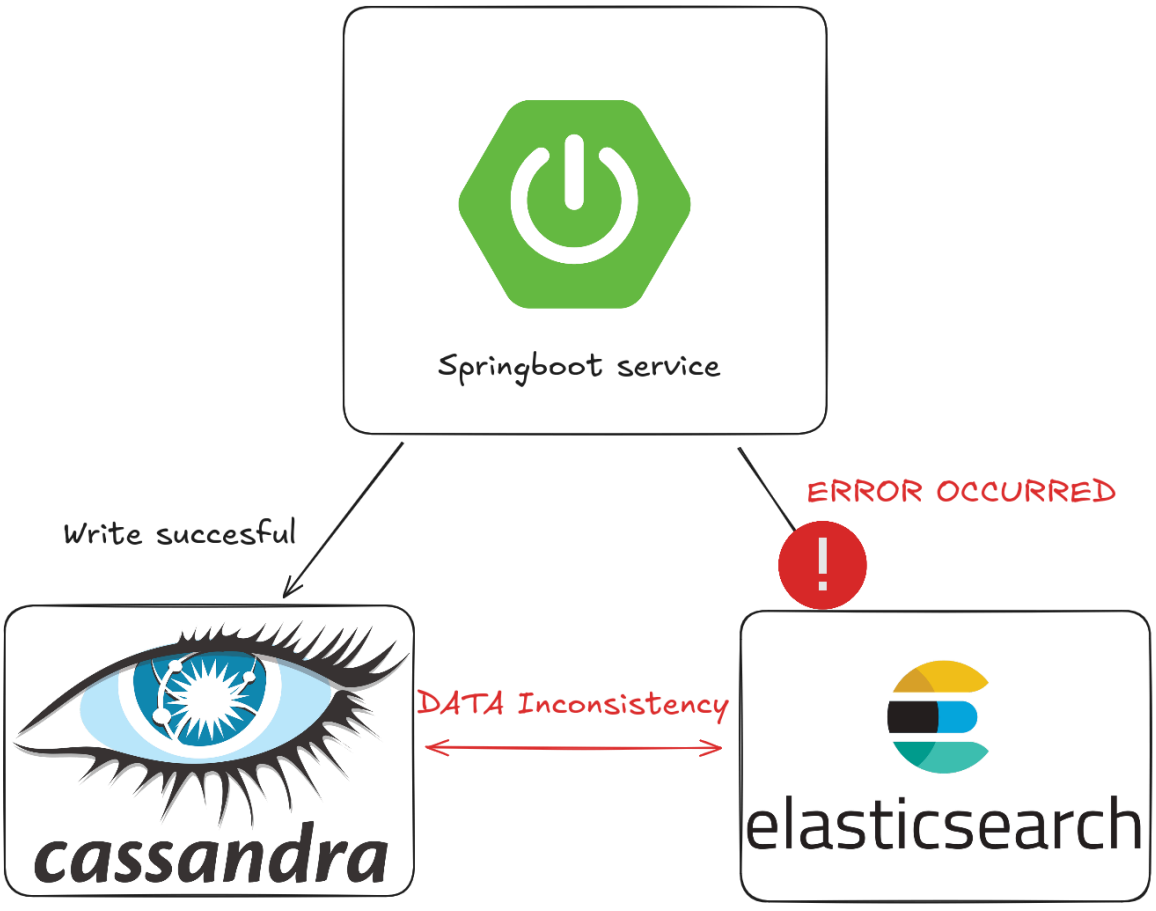

Figure 4: How inconsistencies are caused

Industry validation

This architectural choice is validated by evidency through showroom research since multiple other giant streaming services have adopted this architecture with Cassandra and ElasticSearch.

1. Spotify: The service which was the inspiration for this individual project, confirms using Cassandra for high-scale data storage and using ElasticSearch specifically for meta-data indexing, which can drive it’s amazing search[18], [19]
2. Deezer: A smaller competitor of Spotify, which has also implemented ElasticSearch for search indexing has achieved a 10x improvement in query speed and also uses it for it’s predictive searh field[20].

This provides the necessary certainty which is required by the DOT framework in the why of the research, and proves that the proposed hybrid architecture will be the best for achieving the non-functional requirements.

Conclusion

The technical analysis confirms that migrating from using the high-latency Cassandra songs_by_title structure towards using a new hybrid architecture is required for achieving scalability and a modern user experience. The new architecture also applies separation of concerns[5] which is best practice within software engineering.

The chosen dual-write pattern[17] meets the requirements for being used while the application is still within development. The complexity of implementing a CDC within the time constraints forces the decision to use the dual-write pattern and a temporary risk of data consistency is acceptable.

Future development recommendations

The dual-write pattern must only be considered a temporary solution, if any extra development time becomes available, the project should be upgraded to include a CDC which replaces the dual-write pattern.

Bibliography

[1] [Online]. Available: https://en.wikipedia.org/wiki/Change_data_capture

[2] A. Inamdar, “Define and optimize data partitions in Apache Cassandra | Opensource.com.” Accessed: Oct. 21, 2025. [Online]. Available: https://opensource.com/article/20/5/apache-cassandra

[3] DataStax, “Best practices for data modeling in Cassandra-based databases | CQL for DataStax Hyper-Converged Database | DataStax Docs.” Accessed: Oct. 21, 2025. [Online]. Available: https://docs.datastax.com/en/cql/hcd/data-modeling/best-practices.html

[4] R. Bhatt, “Data Distribution Patterns: Fanout-on-Read vs. Fanout-on-Write - Rutvik Bhatt - Technologist & Engineering Leader.” Accessed: Oct. 21, 2025. [Online]. Available: https://www.rutvikbhatt.com/data-distribution-patterns-fanout-on-read-vs-fanout-on-write/

[5] “Separation of concerns.” Accessed: Oct. 21, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=1317504139

[6] A. Bekker, “Cassandra Performance: The most comprehensive overview you’ll ever see.” [Online]. Available: https://www.scnsoft.com/data/cassandra-performance

[7] Influxes, “How does a search engine database work?: Influxdb.” [Online]. Available: https://www.influxdata.com/search-engine-database/

[8] Datastax, “Data Modeling Concepts.” [Online]. Available: https://docs.datastax.com/en/cql-oss/3.1/cql/ddl/dataModelingApproach.html

[9] Zilliz, “Apache Cassandra vs. Elasticsearch on Vector Search - Zilliz Blog.” [Online]. Available: https://zilliz.com/blog/apache-cassandra-vs-elasticsearch-a-comprehensive-vector-database-comparison

[10] fereshtehfereshteh 53955 silver badges1818 bronze badges, R. 11.1k22 gold badges4848 silver badges3333 bronze badges, R. R. O. a year ago, and devshorts devshorts Over a year ago, “What is the difference between a secondary index and an Inverted index in Cassandra?.” [Online]. Available: https://stackoverflow.com/questions/19248458/what-is-the-difference-between-a-secondary-index-and-an-inverted-index-in-cassan

[11] Elastic, “Full-text search.” [Online]. Available: https://www.elastic.co/docs/solutions/search/full-text

[12] GeeksforGeeks, “Inverted index.” [Online]. Available: https://www.geeksforgeeks.org/dbms/inverted-index/

[13] elastic, “Fuzzy query.” [Online]. Available: https://www.elastic.co/docs/reference/query-languages/query-dsl/query-dsl-fuzzy-query

[14] poster, “Elasticsearch fuzzy query - techniques, use cases & examples.” [Online]. Available: https://opster.com/guides/elasticsearch/search-apis/elasticsearch-fuzzy-query/

[15] debezium, “Debezium connector for Cassandra.” [Online]. Available: https://debezium.io/documentation/reference/stable/connectors/cassandra.html

[16] DataStax, “About CDC for Cassandra: Datastax CDC for Apache Cassandra: DataStax Docs.” [Online]. Available: https://docs.datastax.com/en/cdc-for-cassandra/2.3.2/index.html

[17] W. Waldron, “Understanding the dual-write problem and its solutions.” [Online]. Available: https://www.confluent.io/blog/dual-write-problem/

[18] M. L. Engineer Alexandre Tamborrino, “Introducing Natural Language Search for Podcast Episodes.” [Online]. Available: https://engineering.atspotify.com/2022/03/introducing-natural-language-search-for-podcast-episodes

[19] K. M. Brown and Matt, “Personalization at Spotify using Cassandra.” [Online]. Available: https://engineering.atspotify.com/2015/01/personalization-at-spotify-using-cassandra

[20] [Online]. Available: https://www.elastic.co/customers/deezer