Couchbase

SEPTEMBER 2025

Comparing MongoDB and Couchbase for Real-Time Analytics

A benchmarking report from Couchbase and an independent benchmarking subcontractor

Executive Summary

Document databases were developed in response to a rise in the volume of data and the frequency with which information is stored, accessed, and updated. The YCSB benchmark is the de facto standard for measuring operational performance across operational NoSQL database workloads. For analytic workloads, there is not yet a de facto standard for testing NoSQL databases, so these tests utilized the CH2++ benchmark, a modern methodology for evaluating hybrid operational and analytical processing. More on that below. This report presents a comparative performance analysis of two leading document database platforms, Couchbase Capella, including its Capella Analytics service, and MongoDB Atlas with Atlas SQL. While much more detail is contained within the report, as a summary, **Couchbase Capella outperforms MongoDB Atlas in a range of ~100-150 times better** for the analytic workload latencies that both platforms completed.

Separately, **Atlas analytic queries often timed out (~30-70% of tests)** and failed to return results. Where the Capella query might return results <10 seconds, Atlas would not return results in the allocated 300-second (5-minute) timeout limit. Capella had no timeout issues, and performed and returned results for 100% of queries.

Overall, both the analytic latency metrics and timeout factors were surprisingly poor given MongoDB's place in the NoSQL market as an operational database.

Challenges with Analyzing JSON Data

JavaScript Object Notation (JSON) is the data consumption format for operational and AI applications, especially those that change frequently and cannot afford the delays of systemically modifying a relational schema in order to introduce a simple new data structure. For instance, applications that manage account profiles, which are updated frequently, are commonly implemented with JSON.

JSON's flexibility is its appeal. It does not require a predefined schema. JSON uses a simple syntax and does not require data types, only quotations around text. JSON documents can contain key/value pairs, simple arrays, arrays of objects, and multidimensional tables. JSON can nest these features within a document, creating multi-level nested documents. Developers can add new fields to a single document, a collection of documents, or only new documents. All of this is easy and controlled by the programmer or program. JSON is the data format for operational use cases.

Challenges arise when organizations seek to analyze operational JSON data due to mismatched requirements. Business intelligence tools prefer their data to be managed with structured schemas, yet JSON does not conform to that expectation. Normalizing and flattening JSON can be a significant challenge for data engineers responsible for setting up and managing data pipelines. Although many utilities help address this problem, it is one that requires constant attention due to the changing nature of the JSON being transformed.

In fact, while this document was originally intended to compare the analytic and operational performance of two popular JSON document databases, it is also useful as a reminder that analyzing JSON-formatted data is challenging, not only for consumers, but also vendors. A NoETL approach, preserving the JSON structure while enabling analytics queries, is what this benchmark aims to measure.

THE BENCHMARK

The primary goal of this study is to assess how well these two cloud-native database platforms support primarily analytic workloads and secondarily operational scenarios, where fast operational access must coexist with ad-hoc analytical insights, while providing SQL-based query access. The benchmark covers a representative workload that reflects the demands of modern applications that rely on document-centric data models and expect low-latency operational queries as well as scalable analytical processing. To ensure a fair and practical comparison, price-comparable cluster configurations were benchmarked across both platforms.

The results aim to inform architects, engineers, and decision-makers about the strengths and limitations of each system under realistic, mixed workload conditions, with a focus on transparency, reproducibility, and practical relevance.

Comparison Methodology

To compare both vendors, the cloud Database-as-a-Service offering of each was utilized, Couchbase Capella and MongoDB Atlas. Both offer support for SQL-based JSON analytics without requiring operational data to be flattened prior to analysis.

- **Capella** offers Capella Operational for transaction-oriented workloads and Capella Analytics for analytical workloads. Capella supports real-time movement of data in the operational store with zeroETL to the analytics store. Changes made in the source are automatically applied to analytics.
- **Atlas** uses standard clusters for operational workloads. In addition, they offer customers analytical nodes that are intended to be used for analytical processing. These analytic nodes are just secondary nodes deployed alongside the operational cluster to provide performance-isolated data access.

Benchmark

This study applies the CH2++ benchmark, a modern benchmarking workload designed to evaluate the performance of hybrid operational and analytical processing (HTAP/HOAP) on document-oriented databases that natively store and query JSON data. It builds upon earlier benchmarks such as TPC-C and TPC-H, as well as the original CH-benCHmark, which was developed for relational HTAP systems.

Originally introduced as CH2 to extend the CH-benCHmark into the NoSQL domain, the CH2++ benchmark represents a significant evolution. CH2++ features a fully document-oriented schema, designed specifically to reflect a more real-world-like usage of JSON databases. It introduces additional fields and schema refinements that better utilize the columnar and analytical capabilities of modern document databases. The benchmark includes both operational queries and updates (e.g., point lookups, inserts, updates) and analytical queries (e.g., aggregations, joins, filtering over nested fields), enabling the comprehensive evaluation of how well a system supports mixed workloads. All queries have been designed to operate on nested JSON structures.

The intensity of both the transactional (TPC-C like) and analytical (TPC-H like) sides of the benchmark can be scaled independently, while the data set size is the same for both. As with many TPC benchmarks, the scaling of the CH2++ data set is dependent on the number of "warehouses" the benchmark is using.

To measure analytic performance in different ways, CH2++ uses 22 analytics queries.

Like many other database benchmarks, CH2++ distinguishes a LOAD phase to prepare the database and a RUN phase where the operational side continuously modifies the content of the database. During the RUN phase, the analytical side of the benchmark iterates over a fixed set of analytical queries. Depending on the database, it may use a different endpoint than the operational part.

For more information on the benchmark techniques, see Appendix 2.

Cluster Set-up

In order to achieve comparable results for both contenders, the tests were set up aiming for a price-equal scenario. The Couchbase Capella configuration was set as a baseline that was also used in the paper highlighted in Appendix 2. More precisely, the setup in Table 1 below was chosen.

Table 1. Cluster Configurations

	Operational Cluster	Analytics Cluster	Add-on services	Hourly Costs
Couchbase Capella	**32 Cores:** 4 nodes with 8 cores and 32 GB RAM	**32 Cores:** 4 nodes with 8 cores and 32 GB RAM	None	$11.80 ($7.22 + $4.58)
MongoDB Atlas	**32 Cores:** 3 nodes M80 (32 cores and 128 GB RAM)	**32 Cores:** 1 nodes M80 32 cores and 128 GB RAM	Atlas SQL	$11.63

As shown in the right-most column, the hourly costs for both setups were very similar. Further, even the technical equipment is similar, even if it appears different on first sight.

- **Operational** – Couchbase's operational cluster uses an active-active approach with four nodes of 4 cores and 32 GB of RAM each, leading to a total of 32 cores and 128 GB of RAM. While MongoDB was given this amount of hardware for **each** of its three nodes, its primary-secondary architecture disallows that all three nodes participate equally in transaction processing and mostly leaves the primary with all the work. From that perspective, MongoDB's setup has an advantage as they do not have to coordinate updates amongst multiple nodes.

- **Analytics** – The cluster for Couchbase consists of four nodes with 8 cores and 32 GB RAM each, while MongoDB's set-up consists of a single M80 node (32 cores and 128GB). As with the operational cluster configuration, this set-up leaves both contenders with a similar amount of hardware and MongoDB with the advantage that it does not have to distribute and coordinate analytical queries across multiple nodes.

Query Language

Natively, MongoDB comes with its MongoDB query language (formerly known as MQL) to run operations against the database. For supporting SQL-type of queries, its Atlas SQL language was used as an add-on service. Atlas SQL is designed to enable analytical querying of JSON data across Atlas clusters and external data sources. It is part of the broader Atlas Data Federation service, which allows users to query, transform, and analyze data in place without needing to move or duplicate it. For this report it was used to query data from one MongoDB cluster deployed as a three-node replica set.

Benchmark Configurations

Table 2 summarizes the configurations that were run against both database systems. **B.1 - B.3** serves the purpose of setting a performance baseline, applied to a single warehouse (around **500MB** of raw data) and then varying the number of transactional and analytical clients. **B.1** aims at simply gaining an initial understanding of the respective single-client performance of the analytic client portions of the benchmark. **B.2** targets gaining an understanding of interference between the two workload classes, while **B.3** tests the scalability of the transactional workload.

The **remaining three configurations** make use of 1,000 warehouses (around **500GB** of raw data). **B.4, B.5, and B.6 correspond to B.1, B.2, and B.3, respectively, while B.6** significantly increases the number of transactional clients to evaluate the scalability of the transactional workload.

Table 2. Benchmark Configurations

	#warehouses	#T_Clients	#A_Clients
B.1	1	0	1
B.2	1	1	1
B.3	1	2	1
B.4	1000	0	1
B.5	1000	1	1
B.6	1000	128	1

Benchmark Setup

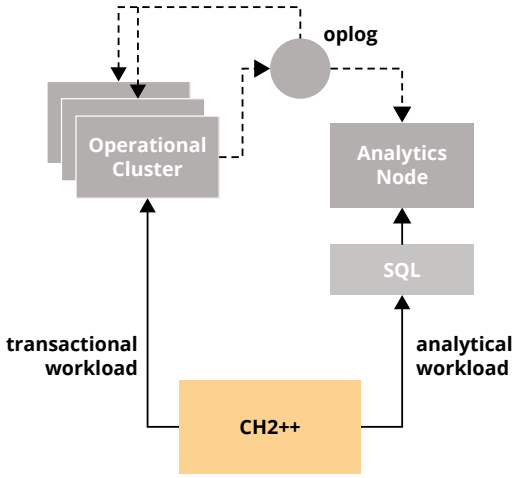


Figure 1: MongoDB Runtime Benchmark Set-up

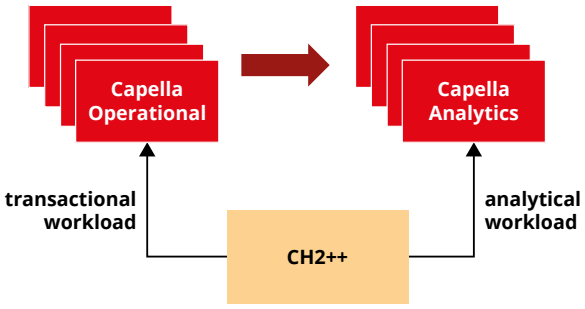


Figure 2: Couchbase Runtime Benchmark Set-up

Figure 1 and Figure 2 illustrate the setup of the benchmark and the respective MongoDB and Couchbase clusters. For MongoDB, the transactional part of the benchmark connects to the 3-node replica set, issuing requests with the MongoDB query language. The analytical part of the benchmark makes use of the Atlas SQL feature targeting the analytics node. Changes made on the operational cluster reach the Analytics Node via the MongoDB oplog via the same mechanism MongoDB uses to keep its Secondary nodes updated (technically, the Analytics Node is another Secondary node).

In the case of Couchbase, the transactional workload targets the Capella Operational cluster, while the analytical workload targets the Capella Analytics cluster. Capella takes care of synchronizing the analytics cluster with the operational cluster.

Benchmark Execution

When running the benchmarks, all infrastructure was deployed on AWS. This infrastructure included the (i) database clusters and (ii) a virtual machine running the CH2++ workload driver. Note that while both clients and servers use the same cloud provider and same provider region, VPC peering is not used. The client targets the database clusters using their respective DNS names, which resolve to publicly routable IP addresses.

For running the benchmarks, the clusters were first initialized.

For **Couchbase**

- Scopes and collections were created for Capella's operational data store
- Some operational indexes were also added (Appendix 1)
- Then indexes were created for Capella Analytics (Appendix 1)
- Then the load step was initiated, where CH2++ inserts the initial data set into the operational cluster
- An ANALYZE statement was run for each collection (cf. Appendix)

For **MongoDB**

- The load step was initiated, where CH2++ inserts the initial data set into the operational cluster
- Several indexes were set up that help both the transactional and analytics workloads (cf. Appendix)

Only after that was the actual benchmark started.

Results

This section presents the results of the evaluations performed following the above methodology. The initial results are for the single warehouse cases (500MB) (B.1 - B.3) before continuing to 1,000 warehouses (500GB) (B.4 - B.6).

In this result section, the performance of each individual analytical query is not discussed, but rather considered as the set of all queries combined. The overall execution time (sum) of all queries is presented, their geometric mean, and the maximum execution time of any query.

Benchmark with 1 Warehouse

Table 3 shows the results of the runs with 1 warehouse (500MB). For the case with one analytical client, Couchbase outperforms MongoDB by several orders of magnitude. The execution of all 22 analytical queries takes **10.75 seconds** for Couchbase, but **more than 46 minutes** for MongoDB (including the queries that timed out for MongoDB).

The longest running query required 1.15 seconds for Couchbase; for MongoDB, the benchmark reported a query time of 300 seconds. This, however, is only the case as the query timeout was set to 300 seconds (5 minutes), which appeared to be a generous timespan for the small data set used in this setup. Overall, for MongoDB, 8 out of 22 queries timed out and did not complete in the allocated five minutes. For Couchbase, all 22 queries executed without timing out.

Table 3. Results for 1 warehouse

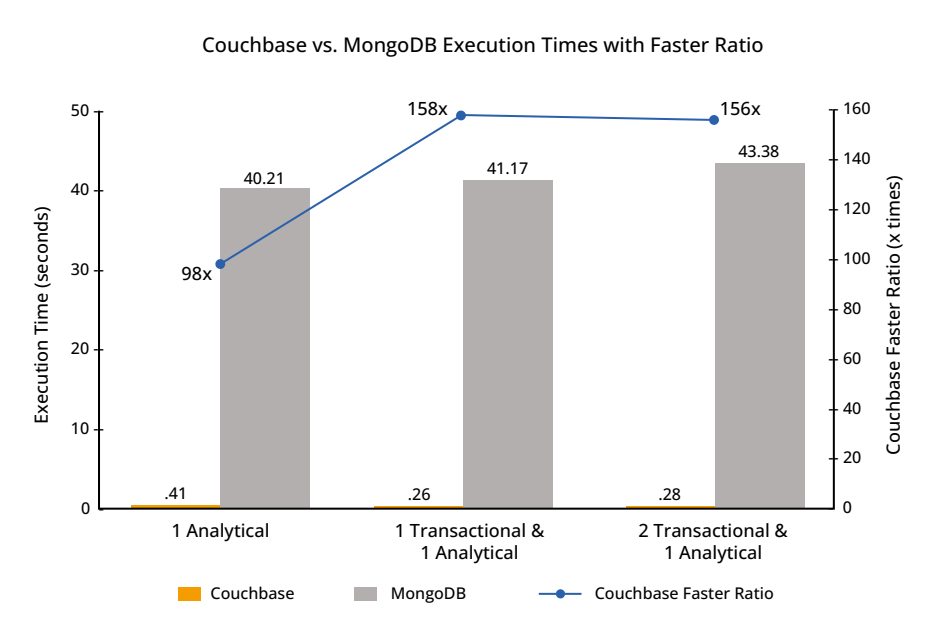
		1 Analytical Client		1 Transactional Client & 1 Analytical Client		2 Transactional Clients & 1 Analytical Client	
		Couchbase	**MongoDB**	**Couchbase**	**MongoDB**	**Couchbase**	**MongoDB**
Analytical Metrics	Sum (sec.)	11.28	2,813	16.64	2,836	18.28	5,093
	Max (sec.)	1.21	300	9.34	300	10.74	300
	Geometric mean (sec.)	**0.41**	**40.21**	**0.26**	**41.17**	**0.28**	**43.38**
	Timed out	0	8	0	8	0	16

As the benchmark is comprised of 22 different analytic queries, the report focuses on the geometric mean as the key metric. Compared to the arithmetic mean, it is less affected by extreme values in a skewed distribution. The geometric mean provides a robust, outlier-resistant summary that's ideal for relative performance insights, making it easier to identify better performance across all queries without misleading averages. The geometric mean is a way to find the "average" of a set of numbers, but instead of adding them up like the regular (arithmetic) mean, you multiply them together and then take the nth root, where n is the number of values.

The geometric mean shines in this scenario because latencies are positive, multiplicative, and often skewed. Why use the geometric mean?

- Handles the multiplicative nature of performance
- Reduces the impact of outliers and skew
- Better for ratios and relative comparisons
- Standard in performance benchmarking

The following is a comparison of Couchbase Analytics to MongoDB Atlas for this metric.



With the above results and to explore MongoDB's timeout behavior more, the timeout was extended to 60 minutes, with defined indexes (Shown in the Appendix 1). The results:

- *6 of 22 (27%) queries still timed out, not finishing their queries in 60 minutes*
- 8 of 22 queries had latencies below 10 seconds
- 5 of 22 were between 24 and 62 seconds
- 1 of 22 needed around 2 minutes
- 1 of 22 needed 44 minutes

Benchmark with 1,000 Warehouses (500GB)

Considering the poor results obtained for MongoDB with 1 warehouse (500 MB), it did not make sense to attempt testing workloads against MongoDB at a scale of 1,000 warehouses (500 GB). While there is no comparison to be made, it was still interesting to test how Capella Analytics would perform and scale.

Table 4. Couchbase Capella Results for 1,000 warehouses

		1 Analytical Client	1 Transactional Client & 1 Analytical Client	128 Transactional Client & 1 Analytical Client
Analytical Metrics	**Sum (sec.)**	659s	708	730
	Max (sec.)	89.4	91.0	95.74
	Geometric mean (sec.)	22.75	24.82	25.63

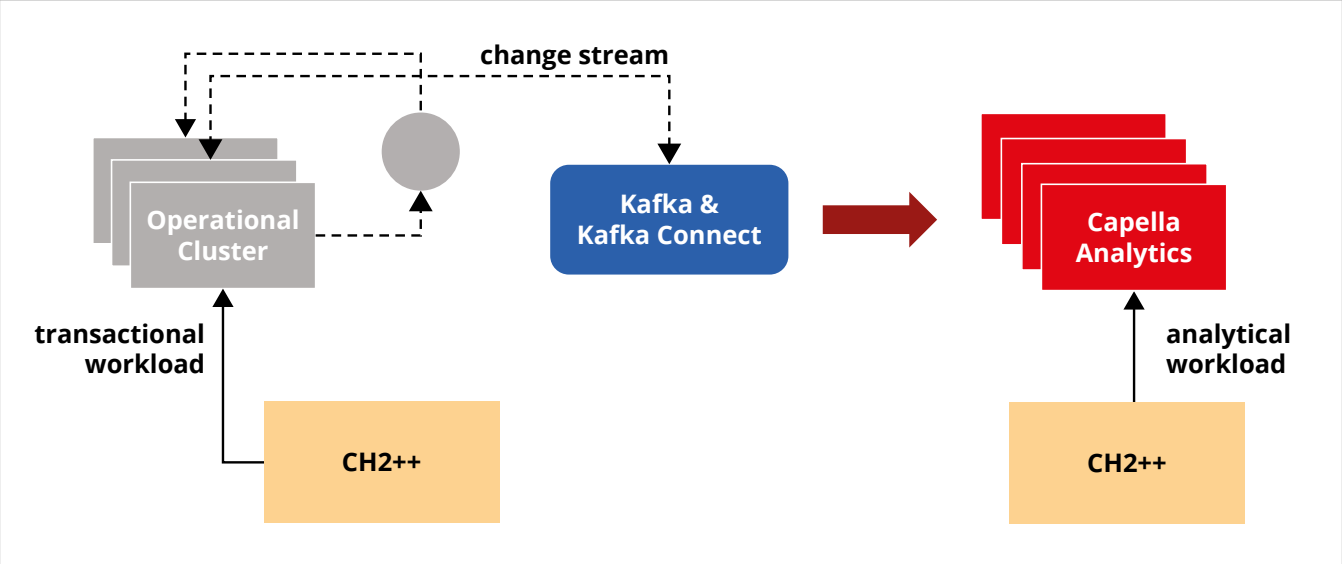
For these numbers, given a 1000X increase in data volume with no infrastructure changes, one might expect a 1000X increase in analytic metric times resulting. Below is a table showing that in fact the geometric mean times increase by less than 100X, or 1/10th relative to that of the data volume increase.

		1 Analytical Client	1 Transactional Client & 1 Analytical Client	128 Transactional Client & 1 Analytical Client
Analytical Metrics	**Geometric mean (sec.) 500MB**	.41	.26	.28
	Geometric mean (sec.) 500GB	22.75	24.82	25.63
	Ratio	55.5	95.5	91.6

Conclusion

In terms of document database platforms, despite MongoDB being widely used as an operational database, this report shows that MongoDB Atlas struggles in the area of analytic performance. This is true in terms of the latency of queries that were executed and those that failed to meet the generous timeout limits. Couchbase Capella had no issues beating the timeout limits, with the longest running query in the test taking less than 11 seconds for the 500MB dataset. Capella Analytics has shown itself to be a far better option for companies looking for analytic solutions for JSON document data. So how might companies currently using MongoDB gain the analytic performance and workload advantages of Couchbase?

In fact, Couchbase makes it easy for companies to bring their MongoDB operational data into Couchbase Analytics without making any operational data changes. This can be achieved via Kafka and brings data into Couchbase Analytics with sub-second latency. This is possible for both cloud and on-prem deployments.



In this way, companies can easily adopt Couchbase Analytics for analytics – without having to change their operational setup – and expect to see query latencies improve in the magnitude of 100-150X faster queries for similar hardware.

For more information about Couchbase Analytics, go to:
https://www.couchbase.com/products/analytics/

APPENDIX 1

Link to Benchmark on GitHub: https://github.com/couchbaselabs/ch2/tree/main

MongoDB Indexes

INDEXES FOR OPERATIONAL WORKLOAD

```
db.item.createIndex( {i_w_id: 1})
db.warehouse.createIndex( {w_id: 1, w_tax: 1})
db.district.createIndex( {d_w_id: 1, d_id: 1, d_next_o_id: 1, d_tax} )
db.customer.createIndex( {c_w_id: 1, c_d_id: 1, c_id: 1} )
db.customer.createIndex( {c_w_id: 1, c_d_id: 1, c_last: 1} )
db.stock.createIndex( {s_w_id: 1, s_i_id: 1, s_quantity: 1} )
db.orders.createIndex( {o_w_id: 1, o_d_id: 1, o_id: 1, o_c_id: 1} )
db.orders.createIndex(
       {o_c_id: 1, o_d_id: 1, o_w_id: 1, o_id: 1, o_carrier_id: 1, o_entry_id: 1 })
db.neworder.createIndex( {no_w_id: 1, no_d_id: 1, no_o_id: 1})
db.district.createIndex(
       { "o_orderline.ol_o_id": 1, "o_orderline.ol_d_id": 1,
              o_orderline.ol_w_id": 1, "o_orderline.ol_number": 1 } )
db.orders.createIndex(
       { "o_orderline.ol_o_id": 1, "o_orderline.ol_d_id": 1,
              "o_orderline.ol_w_id": 1, "o_orderline.ol_i_id": 1 } )
```

INDEXES FOR ANALYTICAL WORKLOAD

```
db.orders.createIndex( { o_entry_d: 1 } );
db.orders.createIndex( { "o_orderline.ol_delivery_d" : 1 } );
db.orders.createIndex( { "o_orderline.ol_i_id": 1 });
db.orders.createIndex( { "o_orderline.ol_i_id": 1, "o_orderline.ol_quantity": 1 });
db.orders.createIndex(
       { "o_orderline.ol_i_id": 1, "o_orderline.ol_delivery_d": 1 });
db.orders.createIndex(
       { "o_orderline.ol_i_id": 1, "o_orderline.ol_supply_w_id": 1 });
db.orders.createIndex(
       { "o_orderline.ol_i_id": 1, o_w_id:1, "o_orderline.ol_quantity": 1 });
db.orders.createIndex( { "o_orderline.ol_delivery_d": 1, o_entry_d: 1 } );
db.orders.createIndex(
       { "o_orderline.ol_delivery_d": 1, "o_orderline.ol_amount": 1 });
db.orders.createIndex(
       { "o_orderline.ol_delivery_d": 1, o_entry_d: 1,
              "o_orderline.ol_i_id": 1,  o_w_id: 1 });
db.orders.createIndex( { o_c_id: 1, o_w_id: 1, o_d_id: 1 } );
db.orders.createIndex(
       { o_c_id: 1, o_w_id: 1, o_d_id: 1, o_entry_d: 1, o_id: 1 } );
db.orders.createIndex(
       { o_c_id: 1, o_w_id: 1, o_d_id: 1, o_entry_d: 1, "o_orderline.ol_i_id": 1 });
```

```
db.orders.createIndex( { o_c_id: 1, o_w_id: 1, o_d_id: 1, o_entry_d: 1 } );
db.orders.createIndex( { o_c_id: 1, o_w_id: 1, o_d_id: 1, o_carrier_id: 1 });
db.orders.createIndex(
       { “o_orderline.ol_supply_w_id”: 1, “o_orderline.ol_i_id”: 1,
              “o_orderline.ol_delivery_d”: 1 });
db.orders.createIndex(
       { “o_orderline.ol_supply_w_id”: 1, “o_orderline.ol_i_id”: 1,
              “o_orderline.ol_delivery_d”: 1, o_c_id: 1, o_w_id: 1, o_d_id: 1 } );
db.orders.createIndex(
       { “o_orderline.ol_supply_w_id”: 1, “o_orderline.ol_i_id”: 1, o_c_id: 1,
              o_w_id: 1, o_d_id: 1, o_entry_d: 1 } );
db.item.createIndex( { i_id: 1 } );
db.item.createIndex( { i_data: 1 } );
db.item.createIndex( { i_id: 1, i_data: 1 } );
db.item.createIndex( { i_id: 1, i_price: 1, i_data: 1 } );
db.stock.createIndex( { s_w_id: 1, s_i_id: 1 } );
db.stock.createIndex( { s_w_id: 1, s_i_id: 1, s_quantity: 1 } );
db.nation.createIndex( { n_nationkey: 1 } );
db.nation.createIndex( { n_nationkey: 1, n_name: 1 } );
db.nation.createIndex( { n_nationkey: 1, n_regionkey: 1 } );
db.supplier.createIndex( { su_nationkey: 1 } );
db.supplier.createIndex( { su_comment: 1 } );
db.supplier.createIndex( { su_suppkey: 1 } );
db.supplier.createIndex( { su_suppkey: 1, su_nationkey: 1 });
db.region.createIndex( { r_regionkey: 1, r_name: 1 });
db.neworder.createIndex( { no_w_id: 1, no_d_id: 1, no_o_id: 1 });
db.customer.createIndex( { c_id: 1, c_d_id: 1, c_w_id: 1 });
db.customer.createIndex(
       { c_id: 1, c_d_id: 1, c_w_id: 1, “c_addresses.c_address_kind”: 1,
              “c_addresses.c_state”: 1 });
db.customer.createIndex(
       { c_balance: 1, “c_phones.c_phone_number”: 1, “c_phones.c_phone_kind”: 1 });
```

Couchbase Initialization

CREATE SCOPE AND COLLECTIONS

```
CREATE SCOPE bench.ch2pp;
CREATE COLLECTION bench.ch2pp.item;
CREATE COLLECTION bench.ch2pp.warehouse;
CREATE COLLECTION bench.ch2pp.district;
CREATE COLLECTION bench.ch2pp.customer;
CREATE COLLECTION bench.ch2pp.stock;
CREATE COLLECTION bench.ch2pp.orders;
CREATE COLLECTION bench.ch2pp.neworder;
CREATE COLLECTION bench.ch2pp.history;
CREATE COLLECTION bench.ch2pp.supplier;
CREATE COLLECTION bench.ch2pp.nation;
CREATE COLLECTION bench.ch2pp.region;
```

PRIMARY INDEX FOR CAPELLA OPERATIONAL CLUSTER

```
CREATE PRIMARY INDEX ON bench.ch2pp.warehouse
```

SECONDARY INDEXES FOR COUCHBASE OPERATIONAL

```
CREATE INDEX cu_w_id_d_id_last
 ON bench.ch2pp.customer(c_w_id, c_d_id, c_last) USING GSI;

CREATE INDEX di_id_w_id ON bench.ch2pp.district(d_id, d_w_id) USING GSI;

CREATE INDEX no_o_id_d_id_w_id
 ON bench.ch2pp.neworder(no_o_id, no_d_id, no_w_id) USING GSI;

CREATE INDEX or_id_d_id_w_id_c_id
 ON bench.ch2pp.orders(o_id, o_d_id, o_w_id, o_c_id) USING GSI;

CREATE INDEX or_w_id_d_id_c_id
 ON bench.ch2pp.orders(o_w_id, o_d_id, o_c_id) USING GSI;

CREATE INDEX wh_id ON bench.ch2pp.warehouse(w_id) USING GSI;
```

SECONDARY INDEXES FOR CAPELLA ANALYTICS CLUSTER

```
CREATE INDEX customer_c_balance ON customer(c_balance:DOUBLE);

CREATE INDEX orders_entry_d ON orders(o_entry_d:STRING);

CREATE INDEX orderline_i_id
 ON orders(UNNEST o_orderline SELECT ol_i_id:BIGINT)
 EXCLUDE UNKNOWN KEY;

CREATE INDEX orderline_delivery_d
 ON orders(UNNEST o_orderline SELECT ol_delivery_d:STRING)
 EXCLUDE UNKNOWN KEY;
```

ANALYZE STATEMENTS FOR CAPELLA ANALYTICS CLUSTER

```
ANALYZE ANALYTICS COLLECTION `customer` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `district` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `history` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `item` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `nation` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `neworder` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `orders` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `region` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `stock` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `supplier` WITH { “sample”: “high” };
ANALYZE ANALYTICS COLLECTION `warehouse` WITH { “sample”: “high” };
```

APPENDIX 2 – BENCHMARK DETAILS

CH2++ benchmark: A modern benchmarking workload designed to evaluate the performance of hybrid operational and analytical processing (HTAP/HOAP) on document-oriented databases that natively store and query JSON data.

M. Carey, V. Sarawathy, D. Nagy B.-C. Wang, K. Murthy, M. Muralikrishna, P. Gupta, and T. Westmann, "CH2++: New HOAP for Benchmarking JSON Data Analytics", 17th TPC Technology Conf. on Performance Evaluation & Benchmarking (TPC TC), London, England, September 2025.

CH-benCHmark

Modern customer experiences need a flexible database platform that can power applications spanning from cloud to edge and everything in between. Couchbase's mission is to simplify how developers and architects develop, deploy and run modern applications wherever they are. We have reimagined the database with our fast, flexible and affordable cloud database platform Capella, allowing organizations to quickly build applications that deliver premium experiences to their customers – all with best-in-class price performance. More than 30% of the Fortune 100 trust Couchbase to power their modern applications. For more information, visit www.couchbase.com and follow us on X (formerly Twitter) @couchbase.