ULB UNIVERSITÉ
LIBRE
DE BRUXELLES

Document Store Databases

Advanced Databases Project

12/24

SHERIF DAOUD SAAD Sara-ID: 000606127
SULAIMAN Marwah- ID: 000603869

NERI Pérez Sebastian - ID: 000605855,
WANTLAND Conde Otto - ID: 000607140

UNIVERSITÉ LIBRE DE BRUXELLES

Document Stores Introduction

NoSQL Databases

- Non-relational databases.
- Stem from the need for managing ever expanding datasets

Main Benefits of NoSQL

01 **Scalability**

- Non Relational structure allows for easy implementation of horizontal scaling.
- Database systems managed in Clusters designed for multiple node integration.

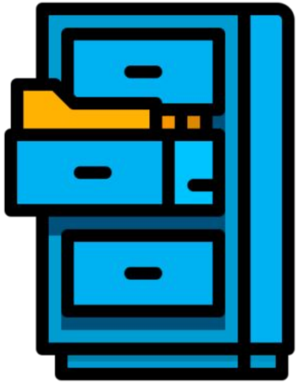
02 **Replicability**

- Databases are designed with distribution in mind.
- Documents are easily copied and maintained across multiple systems.

03 **Ease of Integration**

- Little to no changes in the data structures from application to database means ease of integration with modern systems.

Document Store Databases



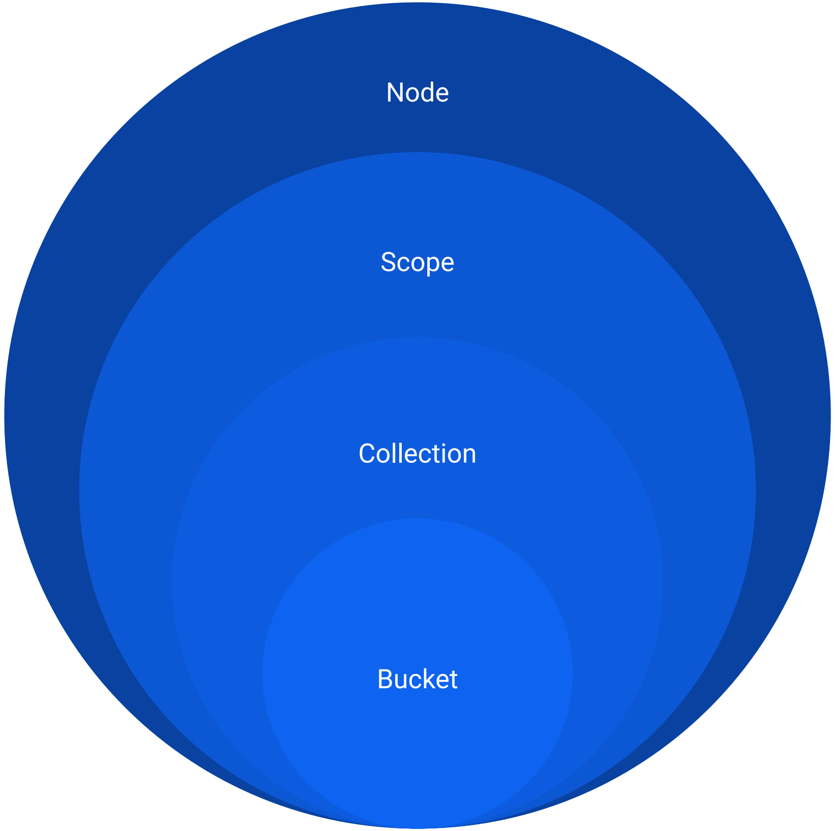
- File Cabinet approach to data management. No direct link between documents.
- Data stored in a specific document format: JSON
- No central schema.
- Ideal for:
    - E-commerce platforms
    - Multimedia Storage

ULB UNIVERSITÉ LIBRE DE BRUXELLES

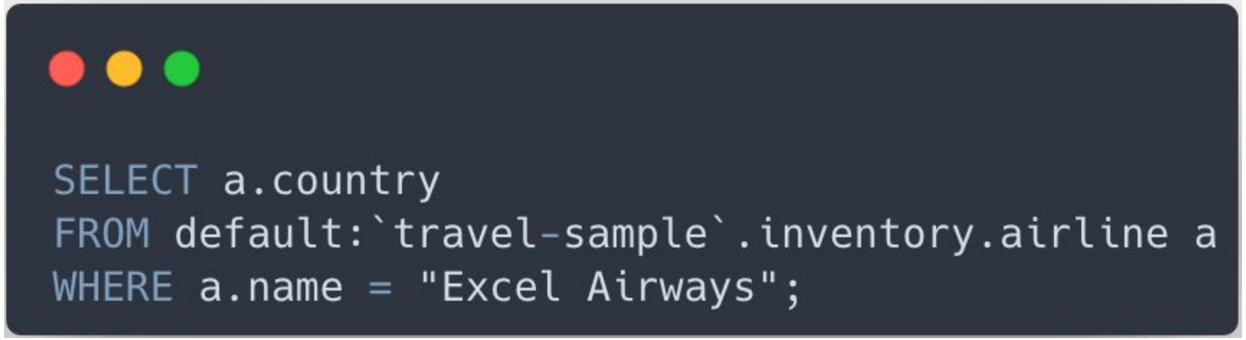
CouchBase & CouchDB Introduction

Tool #1: CouchBase

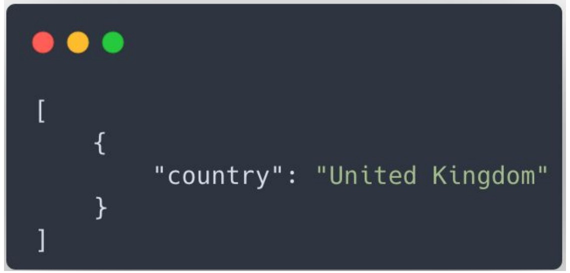
- Documents are stored in buckets.
- Collections/Scopes used for providing structure to database.
- Querying with SQL++
  - Allows for JSON operations and basic query functions.
- Distribution and replication handled by clusters.



Querying With SQL++



```
SELECT a.country
FROM default:`travel-sample`.inventory.airline a
WHERE a.name = "Excel Airways";
```

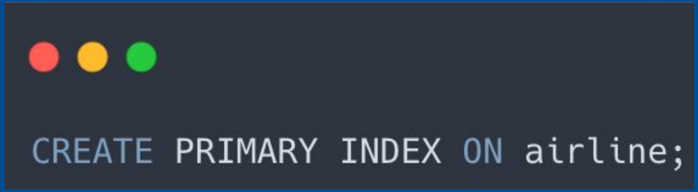


```
[
    {
        "country": "United Kingdom"
    }
]
```

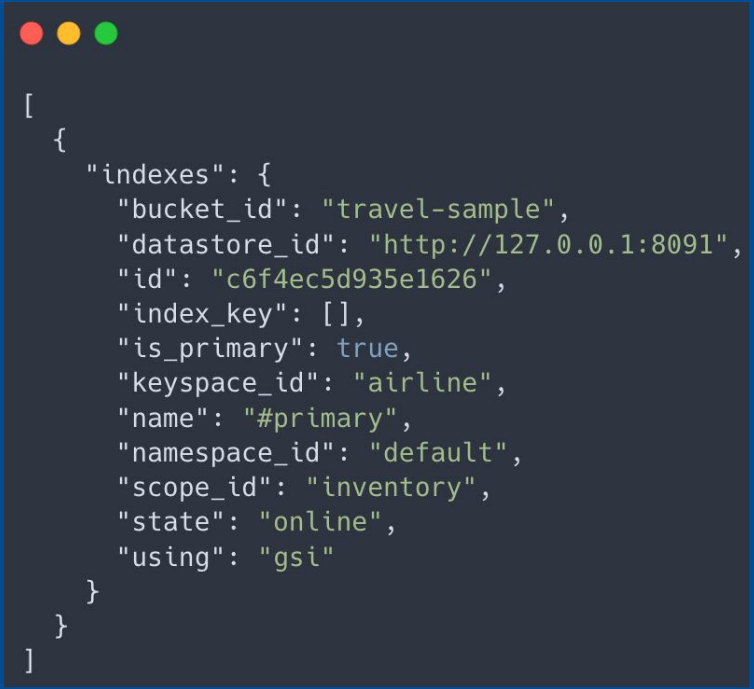
- SQL++ Implements basic SQL elements such as:
  - Arithmetic operations
  - Null value handling
  - Joins
  - Index Creation

Indexing on Couchbase

- Primary and secondary indexes
- Functions like index on relational databases

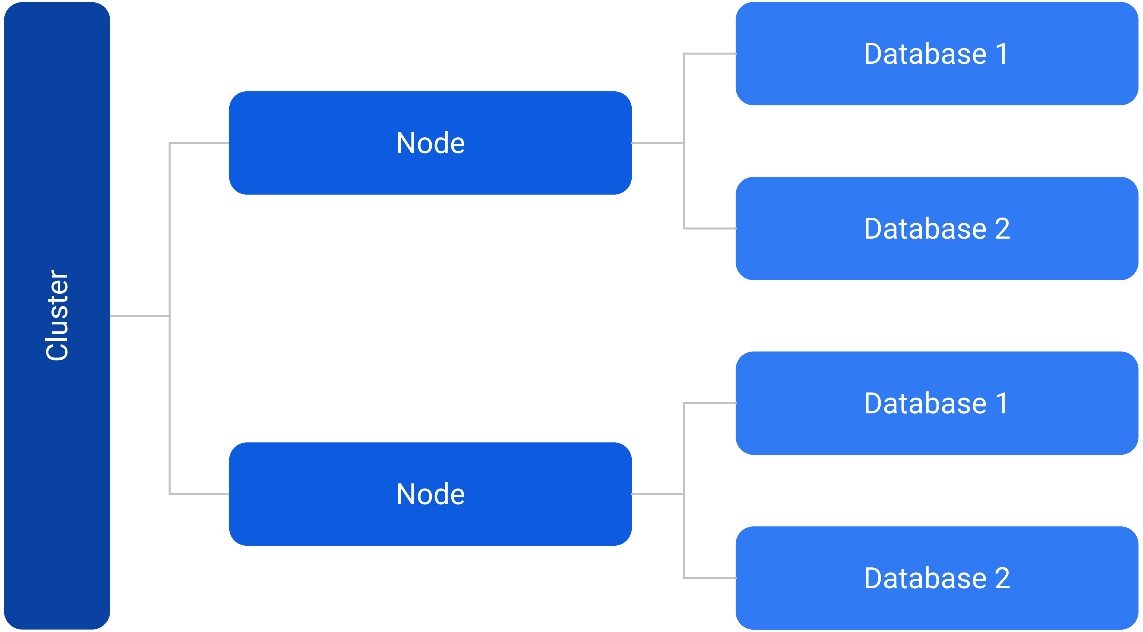


```
CREATE PRIMARY INDEX ON airline;
```



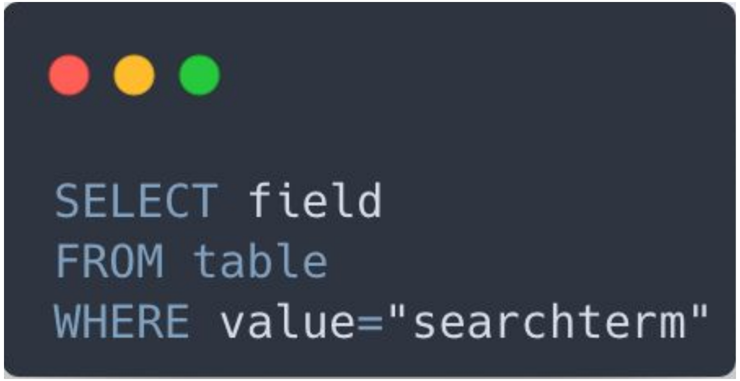
```
[
  {
    "indexes": {
      "bucket_id": "travel-sample",
      "datastore_id": "http://127.0.0.1:8091",
      "id": "c6f4ec5d935e1626",
      "index_key": [],
      "is_primary": true,
      "keyspace_id": "airline",
      "name": "#primary",
      "namespace_id": "default",
      "scope_id": "inventory",
      "state": "online",
      "using": "gsi"
    }
  }
]
```

Tool #2: CouchDB

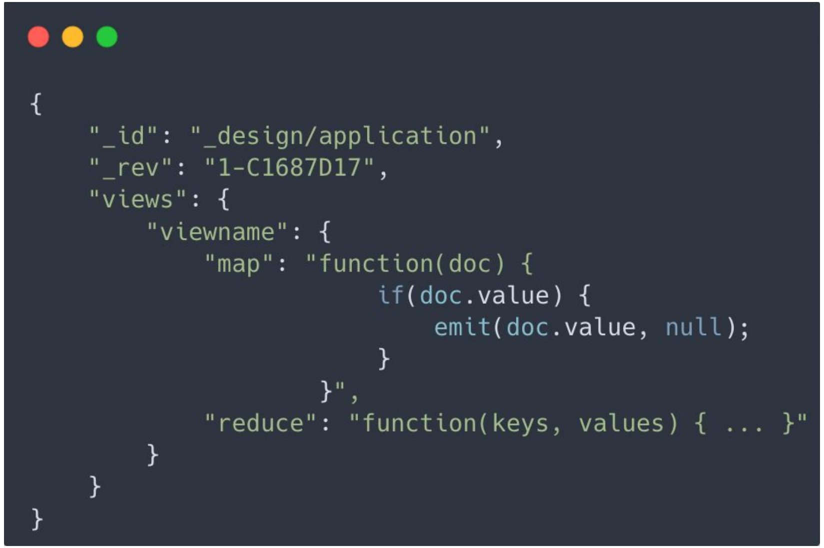


- Cluster system with documents in databases.
- Nodes create shards and replicas
- Relies on View structures for querying data through Javascript

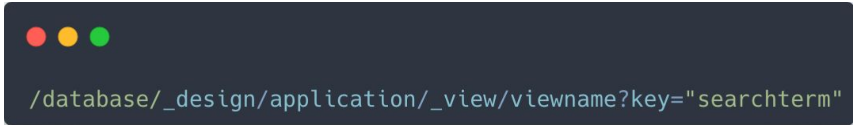
Querying CouchDB Databases



```
SELECT field
FROM table
WHERE value="searchterm"
```



```
{
    "_id": "_design/application",
    "_rev": "1-C1687D17",
    "views": {
        "viewname": {
            "map": "function(doc) {
                        if(doc.value) {
                            emit(doc.value, null);
                        }
                    }",
            "reduce": "function(keys, values) { ... }"
        }
    }
}
```



```
/database/_design/application/_view/viewname?key="searchterm"
```

- Relies on View structures to query information

Real World Application

Booking.com

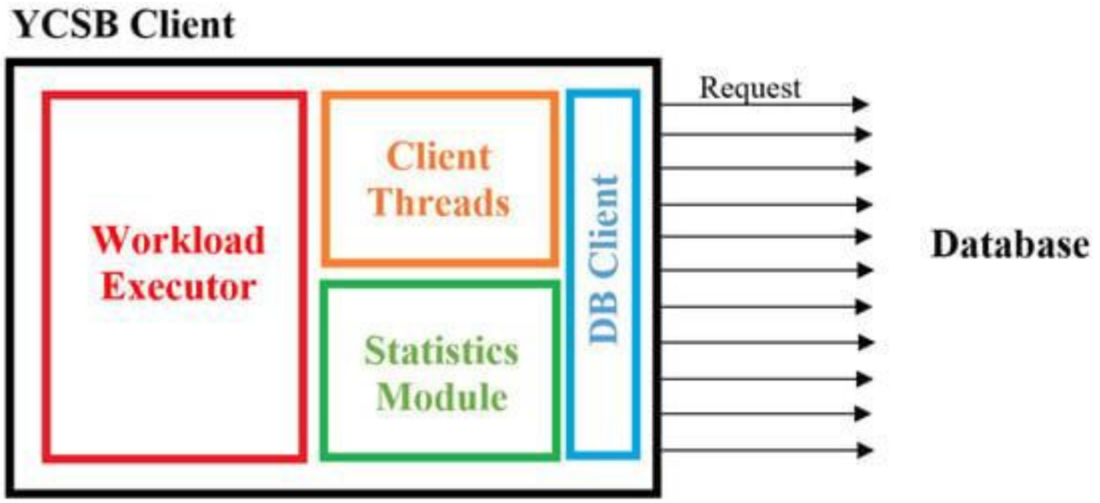
- Web application for finding accommodations.
- Millions of real-time queries.
- Demands high availability.
- Manages enormous amounts of data

Why Document Store?

- Provides high availability through replication and distribution.
- Perfect for handling catalogs and user generated content.

YCSB Benchmarking Framework

Yahoo! Cloud Serving Benchmark (YCSB)

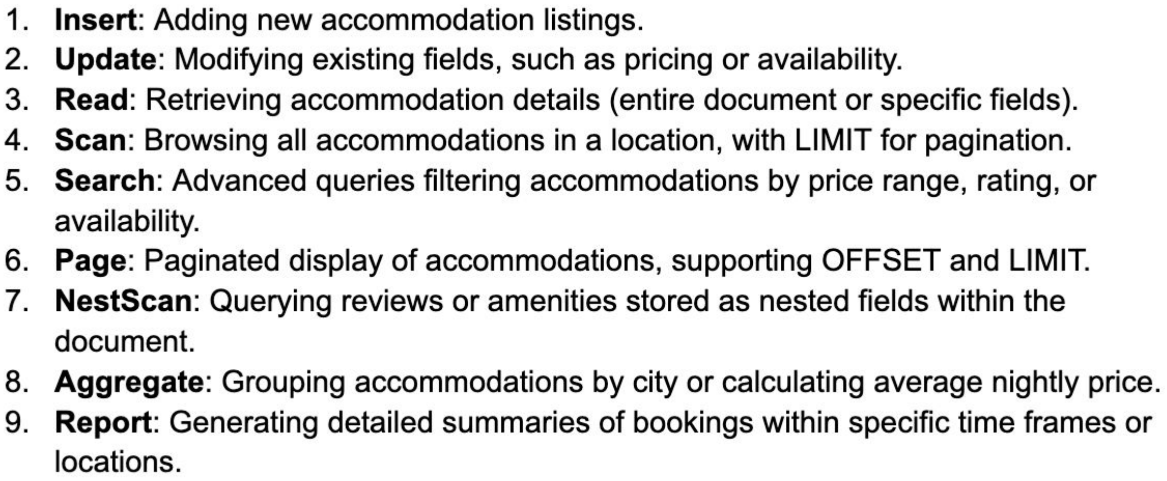


- Standard benchmark for NoSQL systems.
- Highly extendable
- Comprised of core workloads

Benchmarking Our Application

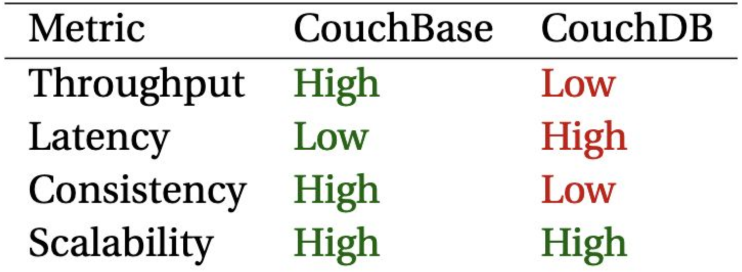
- YCSB may not tell the whole story.
- Identifying key metrics for evaluation.
- Developing new queries for testing.

Key Operations to Benchmark



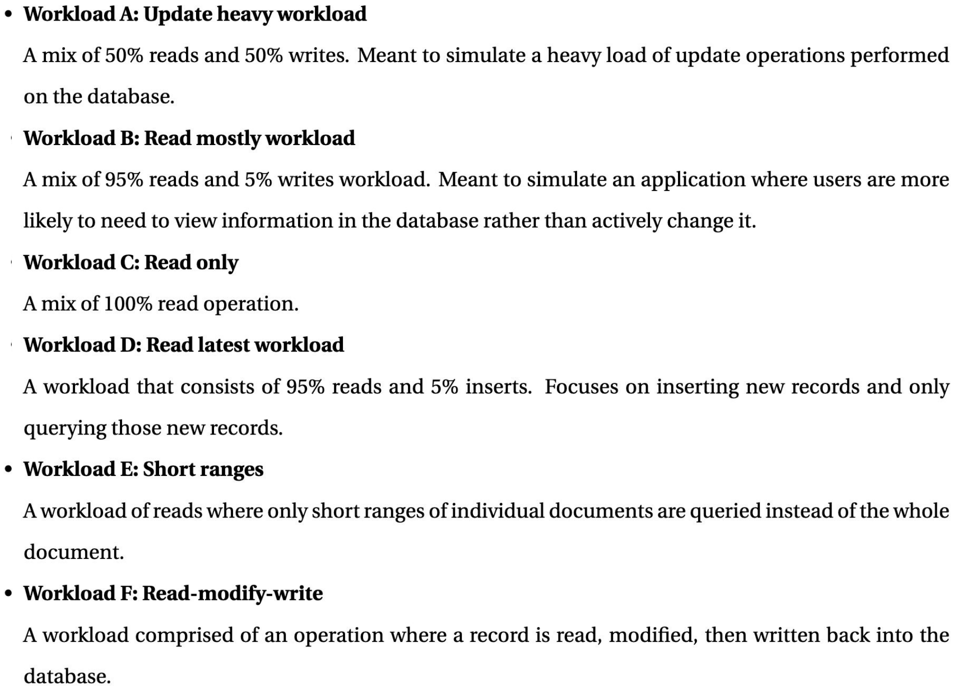
1. **Insert**: Adding new accommodation listings.
2. **Update**: Modifying existing fields, such as pricing or availability.
3. **Read**: Retrieving accommodation details (entire document or specific fields).
4. **Scan**: Browsing all accommodations in a location, with LIMIT for pagination.
5. **Search**: Advanced queries filtering accommodations by price range, rating, or availability.
6. **Page**: Paginated display of accommodations, supporting OFFSET and LIMIT.
7. **NestScan**: Querying reviews or amenities stored as nested fields within the document.
8. **Aggregate**: Grouping accommodations by city or calculating average nightly price.
9. **Report**: Generating detailed summaries of bookings within specific time frames or locations.

YCSB Workloads



Metric	CouchBase	CouchDB
Throughput	High	Low
Latency	Low	High
Consistency	High	Low
Scalability	High	High

Expected results for core workload



- **Workload A: Update heavy workload**

  A mix of 50% reads and 50% writes. Meant to simulate a heavy load of update operations performed on the database.

- **Workload B: Read mostly workload**

  A mix of 95% reads and 5% writes workload. Meant to simulate an application where users are more likely to need to view information in the database rather than actively change it.

- **Workload C: Read only**

  A mix of 100% read operation.

- **Workload D: Read latest workload**

  A workload that consists of 95% reads and 5% inserts. Focuses on inserting new records and only querying those new records.

- **Workload E: Short ranges**

  A workload of reads where only short ranges of individual documents are queried instead of the whole document.

- **Workload F: Read-modify-write**

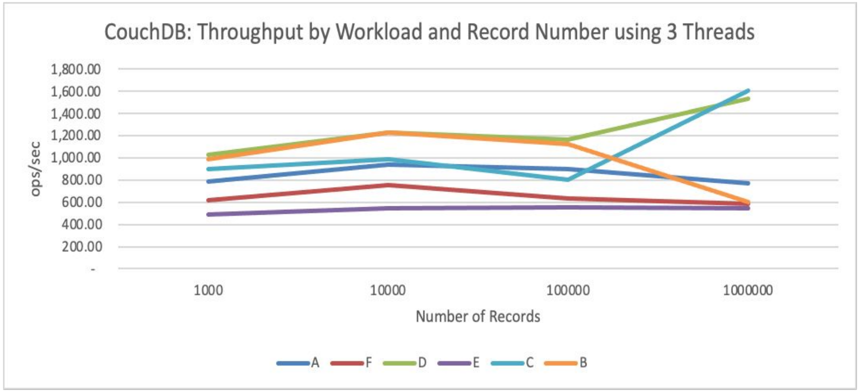
  A workload comprised of an operation where a record is read, modified, then written back into the database.

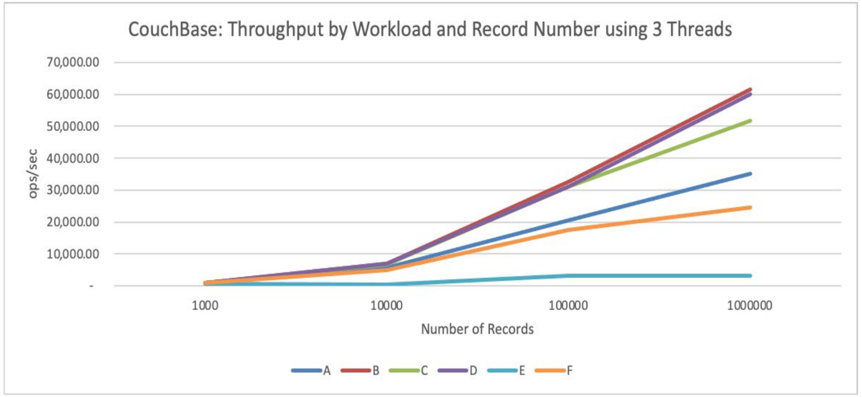
Tools Assessment

- **CouchDB** demonstrated a **constant and limited throughput across all workloads** (~887 ops/sec), with occasional inefficiencies, such as a 50% drop in workload B performance, indicating challenges in handling high-volume reads.

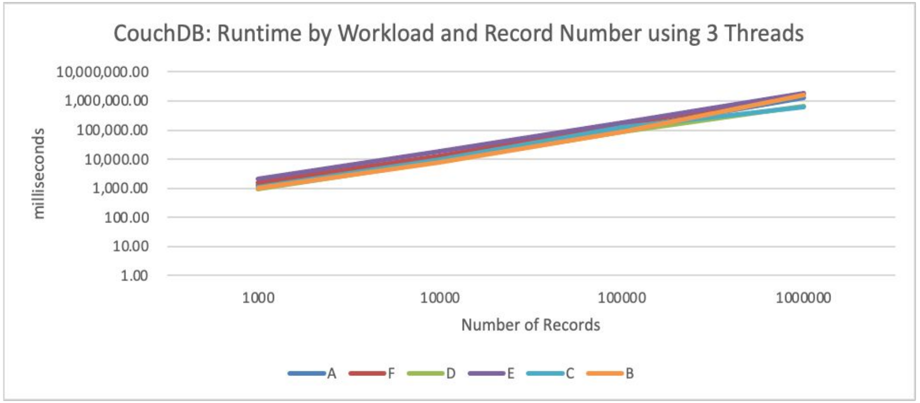
- **Couchbase** achieves **exceptional scalability**, handling up to 60,000 ops/sec for **read-heavy workloads** (B, C & D) and consistently outperforming CouchDB across all workload types, including mixed and scan-intensive scenarios.

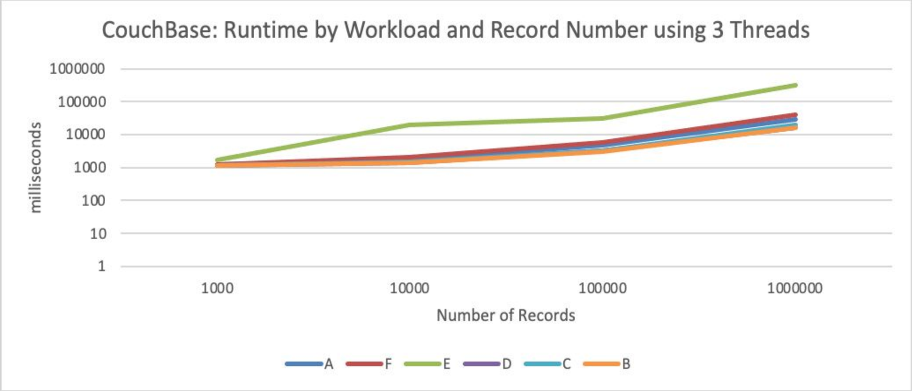
- Couchbase excels in performance and scalability, especially for **read-intensive and large-scale operations**, while CouchDB is **reliable on small datasets** but struggles to scale efficiently on big datasets.





- **CouchDB** shows **linear runtime** growth, reaching ~1,000,000 ms as records scale exponentially, constrained by its **static throughput** and inefficiency at high scales.
- Most workloads for Couchbase maintain **low runtimes** (~25,000 ms), due to its exceptional throughput scalability, except for scan-intensive workload E, which peaks at 306,133 ms showcasing an exponential runtime growth.
- Couchbase's superior scalability **minimizes runtime** for most workloads, while CouchDB's **limited throughput** results in significantly higher runtimes as dataset sizes grow.

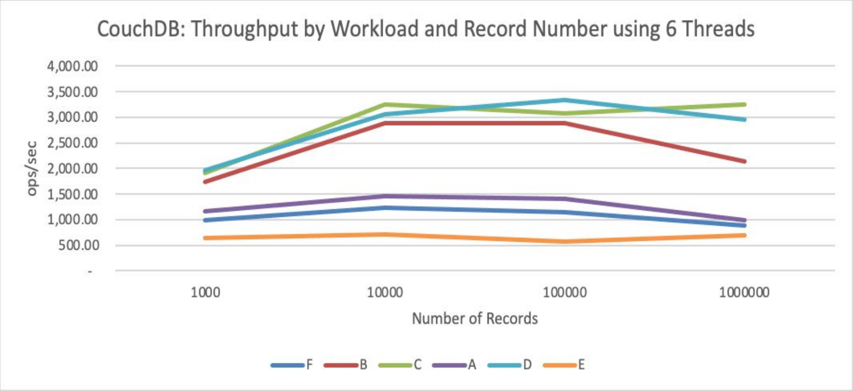


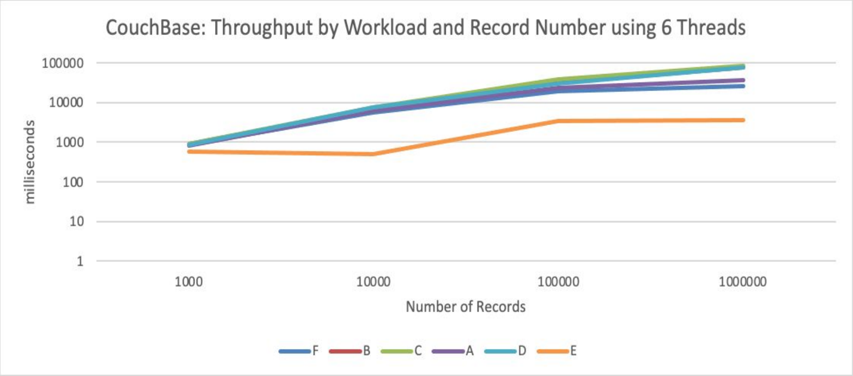


- On **CouchDB** doubling resources from 3 to 6 threads improved throughput by 2.1x (from 887 to 1,848 ops/sec), yet its peak of ~3,500 ops/sec remains significantly below CouchBase's performance.

- On **Couchbase** with 6 threads, throughput scales to 86,000 ops/sec for most workloads, though scan-intensive workload E remains unaffected (~3,600 ops/sec). Overall, throughput increased modestly by 20% with the double of threads.

- Couchbase maintains **superior scalability** and performance across workloads, even with diminishing returns from additional threads, while CouchDB's resource scaling **yields limited improvements**, keeping its performance well behind.

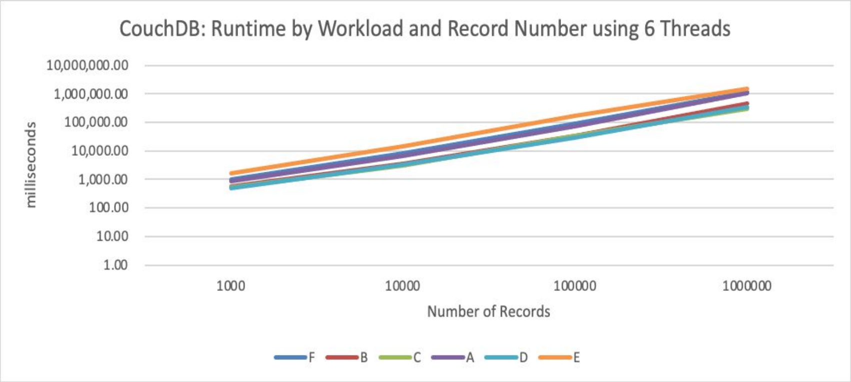


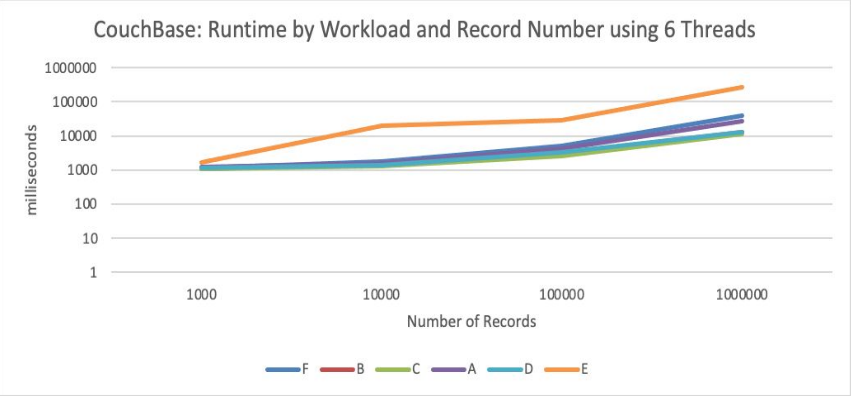


- **CouchDB** achieved with 6 threads a 40% average runtime reduction, but linear growth persisted, peaking at 1.1 million ms for 1 million records, remaining inefficient for large datasets.

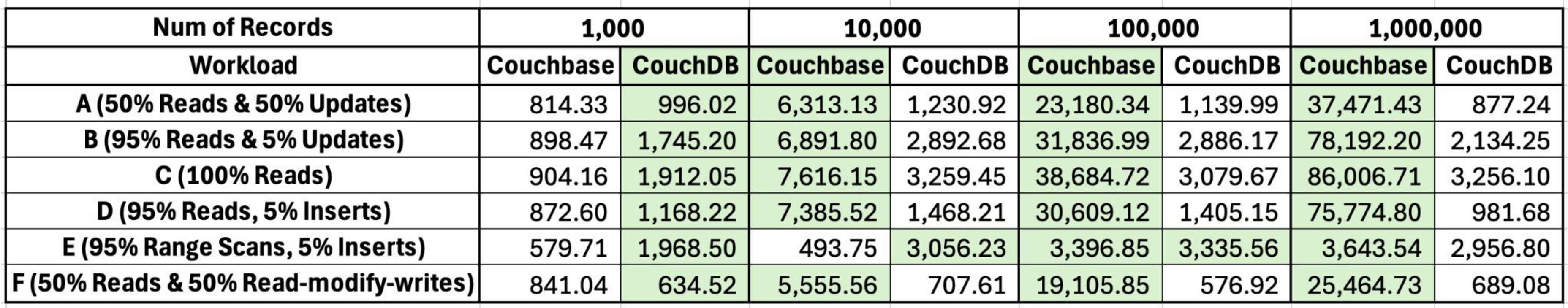
- **Couchbase** showed a modest 10% average runtime decrease with 6 threads, averaging 19,219 ms. Scalability remained strong, though workload E still lagged, with runtimes reaching 274,458 ms.

- Couchbase maintains **efficient runtime scalability** for large datasets, while CouchDB's linear growth and high runtimes, even with resource improvements, reduced its performance for large-scale operations.





- For a **small** number of records (<10k) **CouchDB** achieved a throughput **71% higher** than Couchbase.
- For a **high** number of records (>=10k) **Couchbase** throughput outperformed CouchDB in **almost every workload**



Num of Records	1,000		10,000		100,000		1,000,000	
Workload	**Couchbase**	**CouchDB**	**Couchbase**	**CouchDB**	**Couchbase**	**CouchDB**	**Couchbase**	**CouchDB**
A (50% Reads & 50% Updates)	814.33	996.02	6,313.13	1,230.92	23,180.34	1,139.99	37,471.43	877.24
B (95% Reads & 5% Updates)	898.47	1,745.20	6,891.80	2,892.68	31,836.99	2,886.17	78,192.20	2,134.25
C (100% Reads)	904.16	1,912.05	7,616.15	3,259.45	38,684.72	3,079.67	86,006.71	3,256.10
D (95% Reads, 5% Inserts)	872.60	1,168.22	7,385.52	1,468.21	30,609.12	1,405.15	75,774.80	981.68
E (95% Range Scans, 5% Inserts)	579.71	1,968.50	493.75	3,056.23	3,396.85	3,335.56	3,643.54	2,956.80
F (50% Reads & 50% Read-modify-writes)	841.04	634.52	5,555.56	707.61	19,105.85	576.92	25,464.73	689.08

Key Takeaways

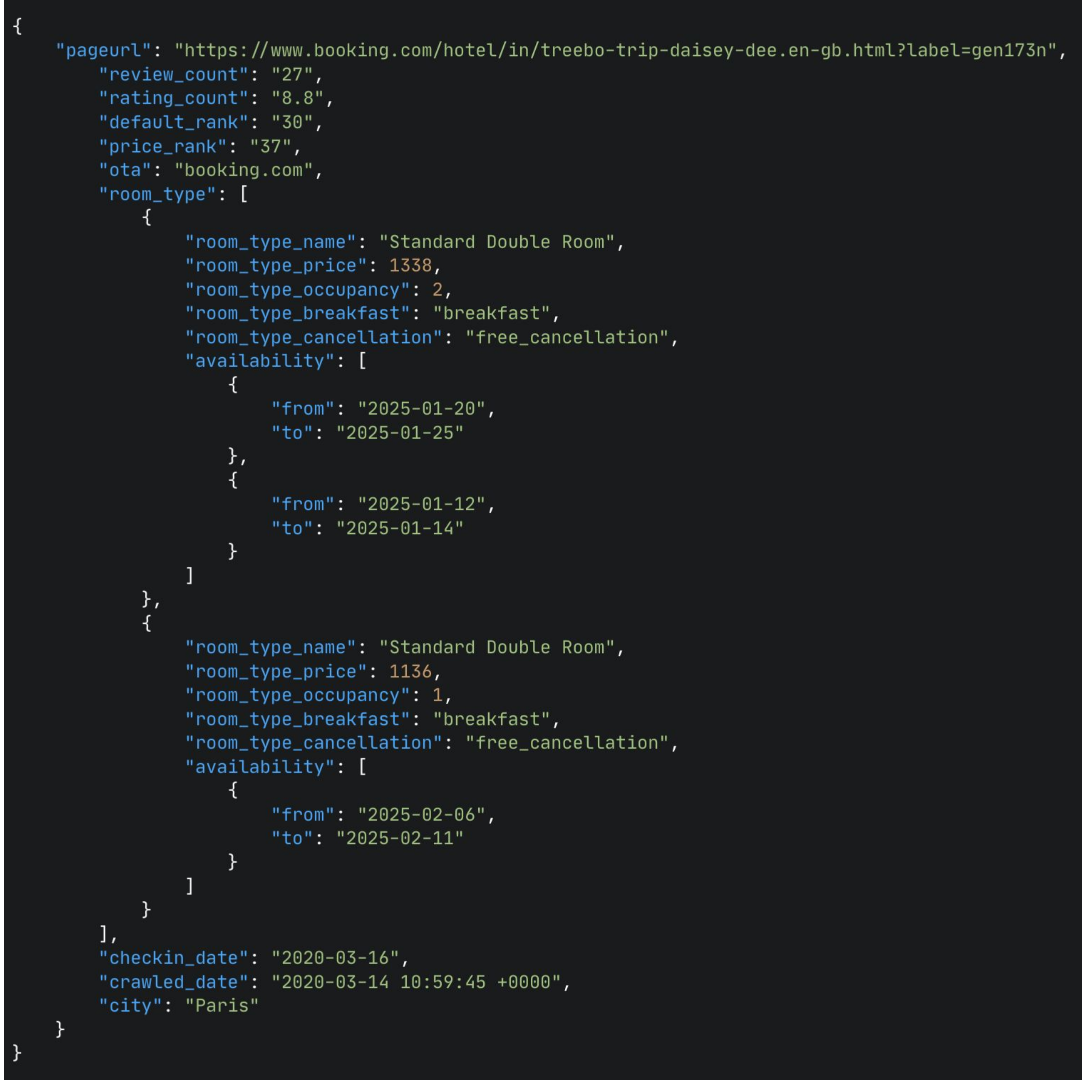
- **CouchDB** excels with **small datasets** (<10k), achieving 71% higher throughput, but **Couchbase** outperforms it for **larger datasets** (≥10k) across most workloads.
- **CouchDB** has a **limited throughput** up to ~3.250 ops/sec, whereas **Couchbase scales excellently**, reaching up to 60,000 ops/sec, outperforming CouchDB across almost all workloads.
- **CouchDB's** runtime **improved** by **40%** by doubling resources but **still remained inefficient** for large datasets, peaking at 1.1 million ms. **Couchbase** had a modest **10% runtime reduction**, showing strong scalability for most cases.

Additional Queries

Dataset

Travel & Hotel Listing from Booking.com 2020

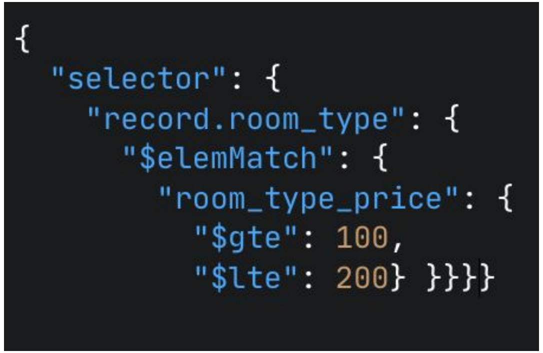
30k records



```
{
    "pageurl": "https://www.booking.com/hotel/in/treebo-trip-daisey-dee.en-gb.html?label=gen173n",
        "review_count": "27",
        "rating_count": "8.8",
        "default_rank": "30",
        "price_rank": "37",
        "ota": "booking.com",
        "room_type": [
            {
                "room_type_name": "Standard Double Room",
                "room_type_price": 1338,
                "room_type_occupancy": 2,
                "room_type_breakfast": "breakfast",
                "room_type_cancellation": "free_cancellation",
                "availability": [
                    {
                        "from": "2025-01-20",
                        "to": "2025-01-25"
                    },
                    {
                        "from": "2025-01-12",
                        "to": "2025-01-14"
                    }
                ]
            },
            {
                "room_type_name": "Standard Double Room",
                "room_type_price": 1136,
                "room_type_occupancy": 1,
                "room_type_breakfast": "breakfast",
                "room_type_cancellation": "free_cancellation",
                "availability": [
                    {
                        "from": "2025-02-06",
                        "to": "2025-02-11"
                    }
                ]
            }
        ],
        "checkin_date": "2020-03-16",
        "crawled_date": "2020-03-14 10:59:45 +0000",
        "city": "Paris"
    }
}
```

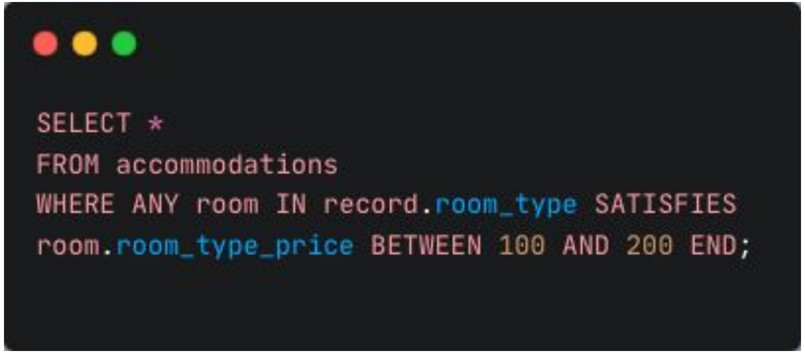
Business Questions - Search

Business Question 1: Retrieve all accommodations with a nightly price between $100 and $200.



```
{
  "selector": {
    "record.room_type": {
      "$elemMatch": {
        "room_type_price": {
          "$gte": 100,
          "$lte": 200} }}}}
```

Couchdb: 7.691s

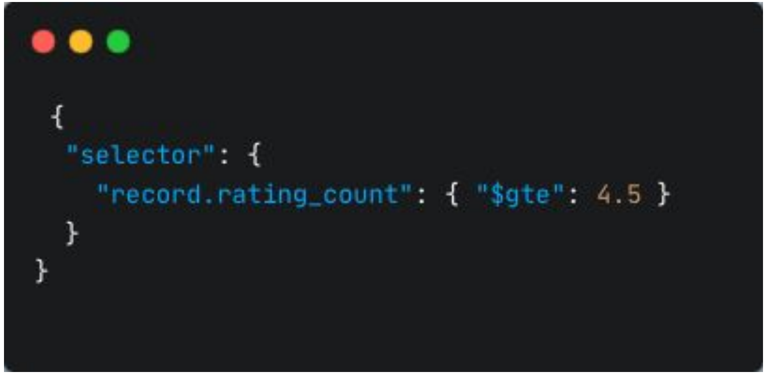


```
SELECT *
FROM accommodations
WHERE ANY room IN record.room_type SATISFIES
room.room_type_price BETWEEN 100 AND 200 END;
```

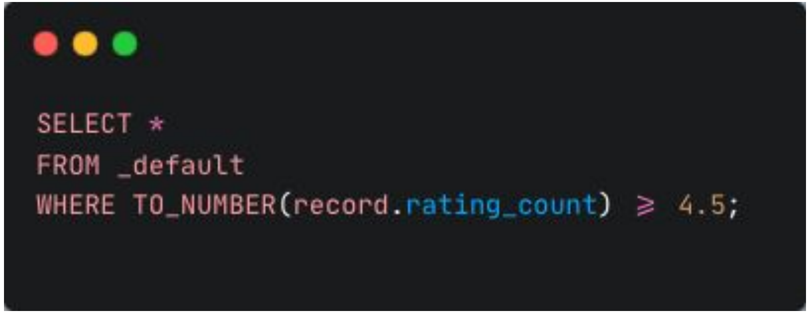
Couchbase: 1.951s

Business Questions - Search

Business Question 2: Find accommodations with an average rating of 4.5 or higher.



```
{
  "selector": {
    "record.rating_count": { "$gte": 4.5 }
  }
}
```



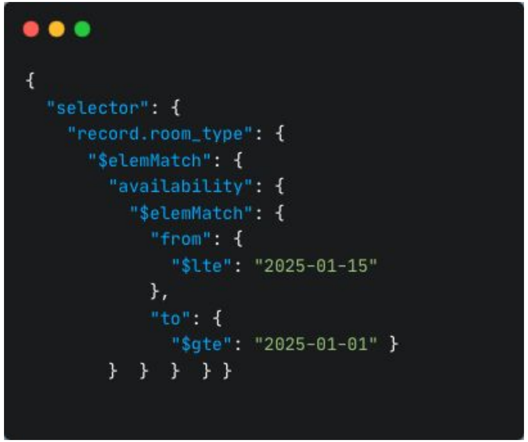
```
SELECT *
FROM _default
WHERE TO_NUMBER(record.rating_count) ≥ 4.5;
```

Couchdb: 12.55s

Couchbase: 6.497s

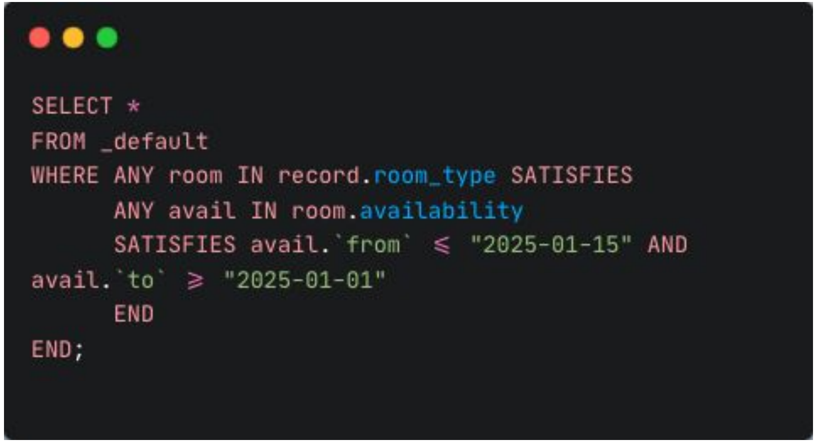
Business Questions - Search

Business Question 3: Search for accommodations that are available between "2024-01-01" and "2024-01-15".



```
{
  "selector": {
    "record.room_type": {
      "$elemMatch": {
        "availability": {
          "$elemMatch": {
            "from": {
              "$lte": "2025-01-15"
            },
            "to": {
              "$gte": "2025-01-01" }
        }  }  }  } }
```

Couchdb: 11.55s



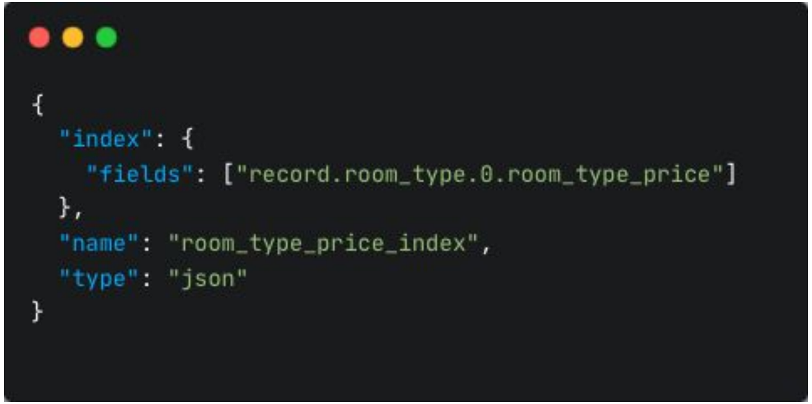
```
SELECT *
FROM _default
WHERE ANY room IN record.room_type SATISFIES
      ANY avail IN room.availability
      SATISFIES avail.`from` ≤ "2025-01-15" AND
avail.`to` ≥ "2025-01-01"
      END
END;
```

Couchbase: 9.288 s

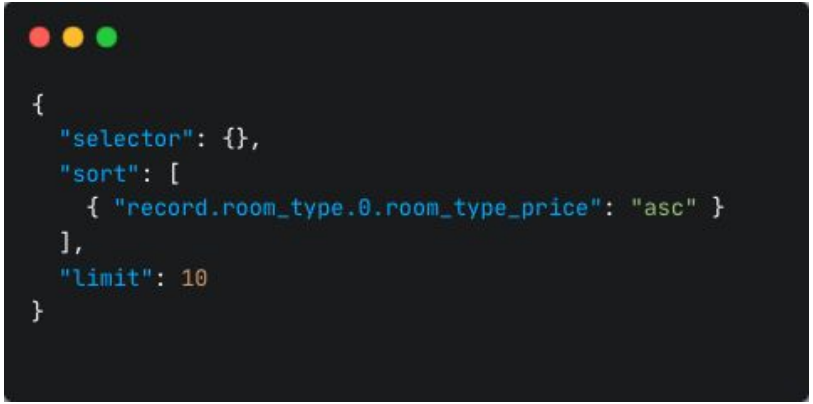
Business Questions - Page

Business Question 1: Show the first 10 accommodations sorted by price.

On couchdb, Query doesn’t work unless we create an index :

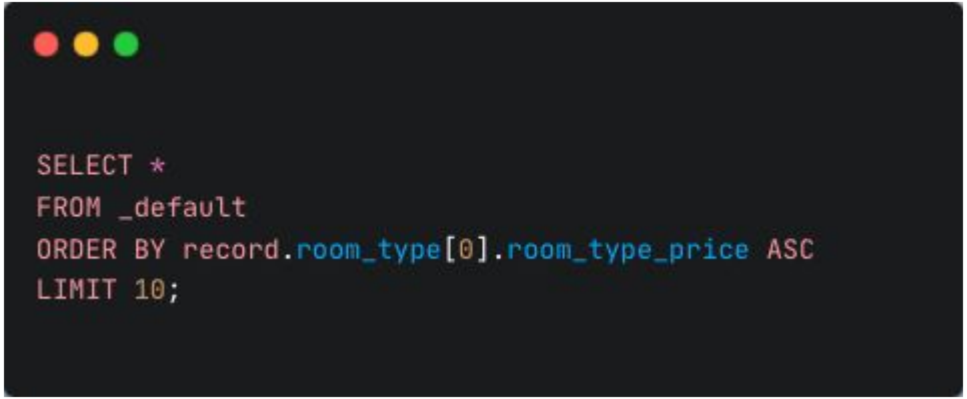


```
{
  "index": {
    "fields": ["record.room_type.0.room_type_price"]
  },
  "name": "room_type_price_index",
  "type": "json"
}
```



```
{
  "selector": {},
  "sort": [
    { "record.room_type.0.room_type_price": "asc" }
  ],
  "limit": 10
}
```

Couchdb: 7.89s



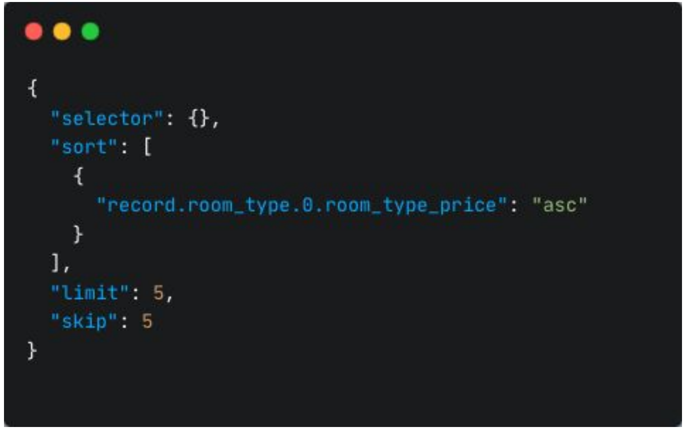
```
SELECT *
FROM _default
ORDER BY record.room_type[0].room_type_price ASC
LIMIT 10;
```

Couchbase: 3.2634s

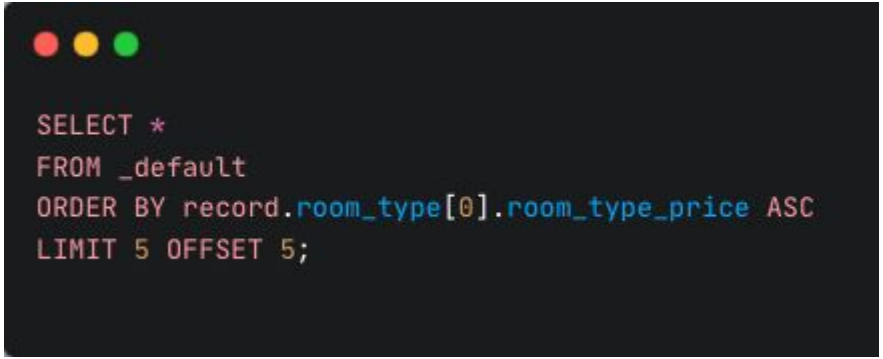
Business Questions - Page

Business Question 2: Display the second page of accommodations with 5 results per page.

Using previous index too::



```
{
  "selector": {},
  "sort": [
    {
      "record.room_type.0.room_type_price": "asc"
    }
  ],
  "limit": 5,
  "skip": 5
}
```



```
SELECT *
FROM _default
ORDER BY record.room_type[0].room_type_price ASC
LIMIT 5 OFFSET 5;
```

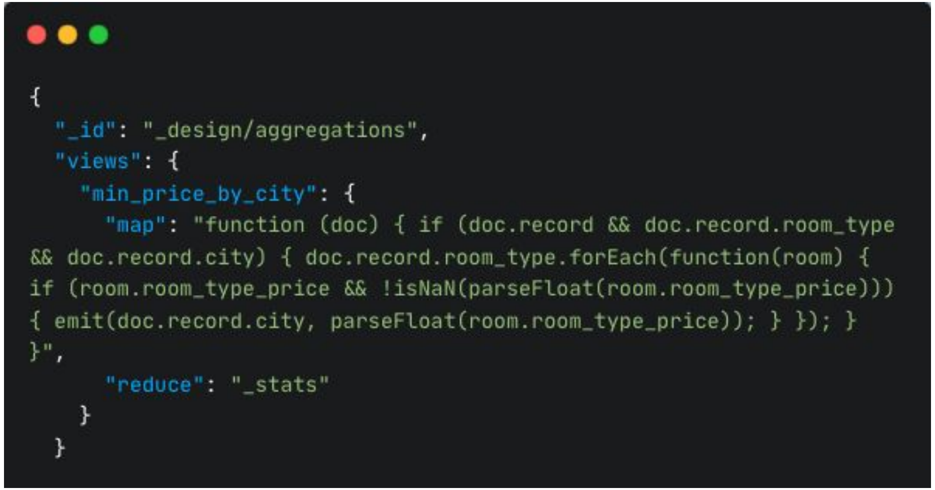
Couchdb: 6.56s

3.25110s

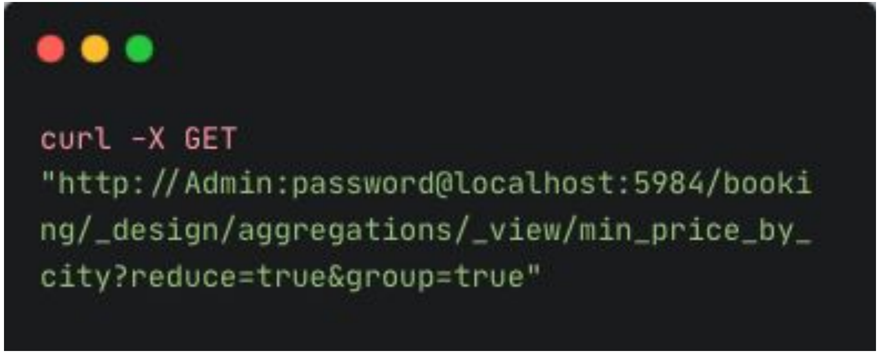
Business Questions - Aggregate

Business Question 1: Calculate the minimum nightly price of accommodations by city.

On couchdb, We need to create a map function and a view :



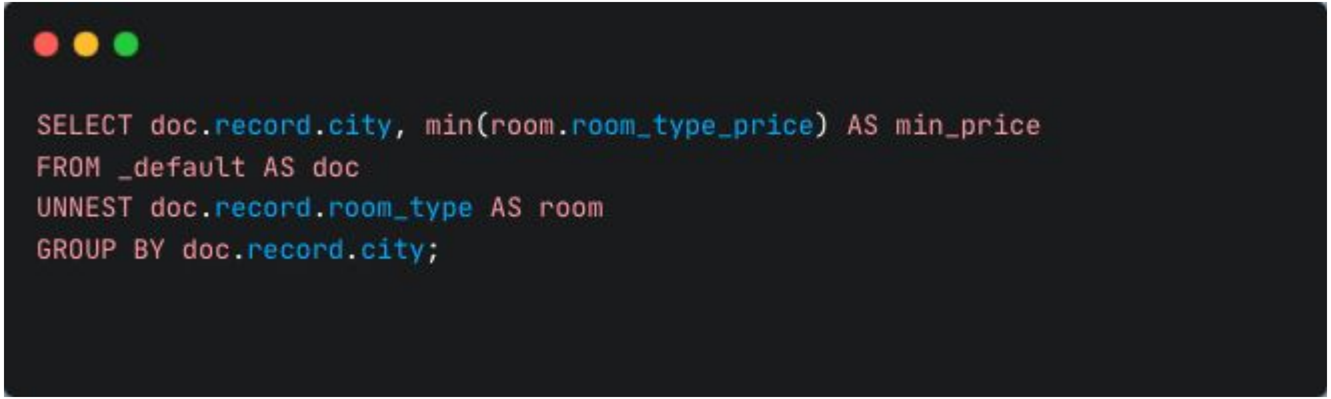
```
{
  "_id": "_design/aggregations",
  "views": {
    "min_price_by_city": {
      "map": "function (doc) { if (doc.record && doc.record.room_type
&& doc.record.city) { doc.record.room_type.forEach(function(room) {
if (room.room_type_price && !isNaN(parseFloat(room.room_type_price)))
{ emit(doc.record.city, parseFloat(room.room_type_price)); } }); }
}",
      "reduce": "_stats"
    }
  }
```



```
curl -X GET
"http://Admin:password@localhost:5984/booki
ng/_design/aggregations/_view/min_price_by_
city?reduce=true&group=true"
```

Couchdb: 7.99s

Business Questions - Aggregate



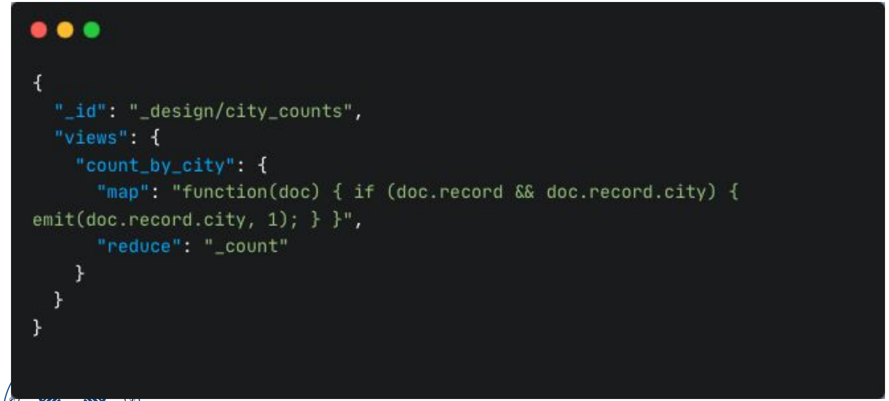
```
SELECT doc.record.city, min(room.room_type_price) AS min_price
FROM _default AS doc
UNNEST doc.record.room_type AS room
GROUP BY doc.record.city;
```

Couchbase: 5.7688s

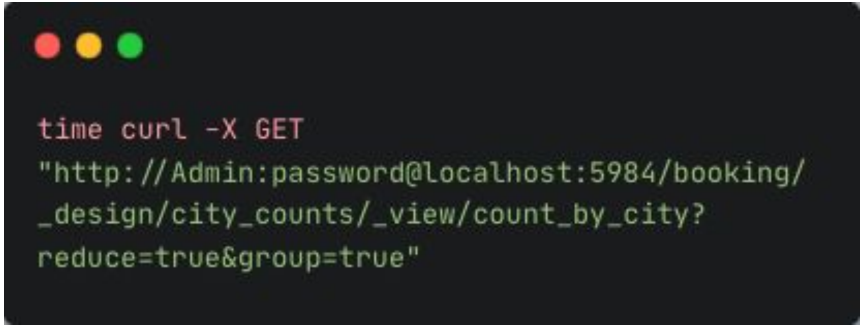
Business Questions - Aggregate

Business Question 2: Count the number of accommodations available in each city.

On couchdb, We need to create a map function and a view :



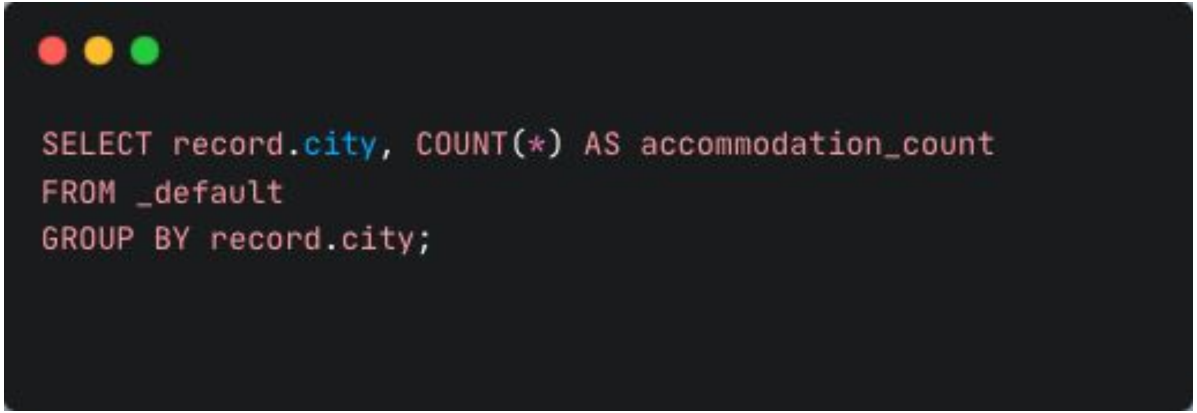
```
{
  "_id": "_design/city_counts",
  "views": {
    "count_by_city": {
      "map": "function(doc) { if (doc.record && doc.record.city) {
emit(doc.record.city, 1); } }",
      "reduce": "_count"
    }
  }
}
```



```
time curl -X GET
"http://Admin:password@localhost:5984/booking/
_design/city_counts/_view/count_by_city?
reduce=true&group=true"
```

Couchdb: 9.04s

Business Questions - Aggregate



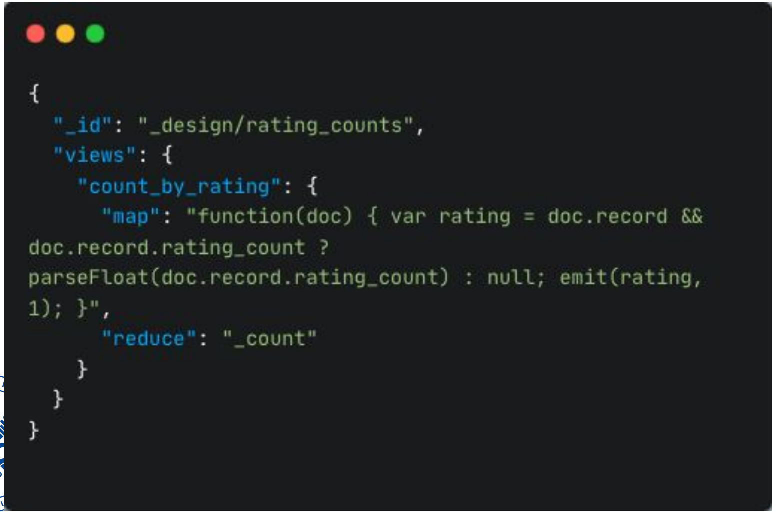
```
SELECT record.city, COUNT(*) AS accommodation_count
FROM _default
GROUP BY record.city;
```

Couchbase: 5.81386s

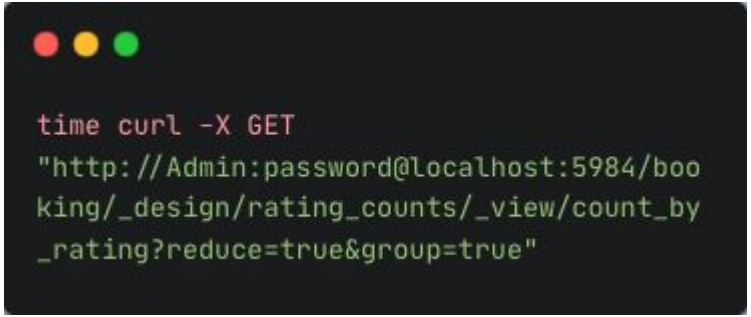
Business Questions - Aggregate

Business Question 3: Group accommodations by rating and count the total in each group.

On couchdb, We need to create a map function and a view :



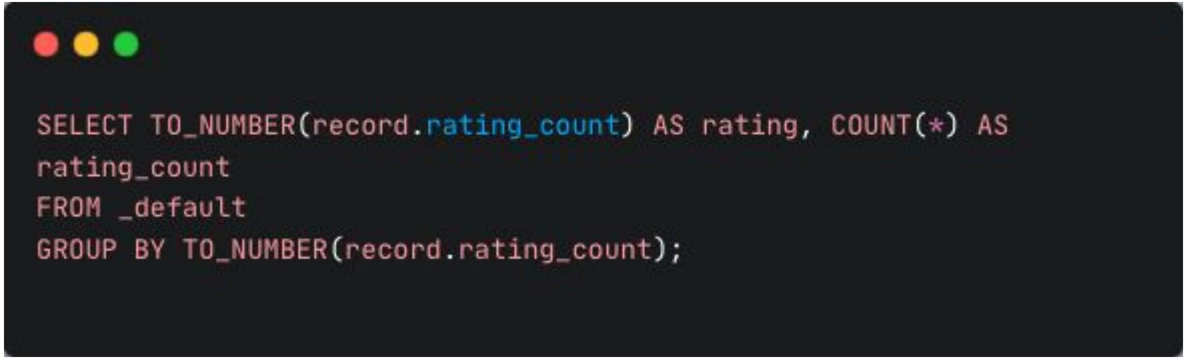
```
{
  "_id": "_design/rating_counts",
  "views": {
    "count_by_rating": {
      "map": "function(doc) { var rating = doc.record &&
doc.record.rating_count ?
parseFloat(doc.record.rating_count) : null; emit(rating,
1); }",
      "reduce": "_count"
    }
  }
}
```



```
time curl -X GET
"http://Admin:password@localhost:5984/boo
king/_design/rating_counts/_view/count_by
_rating?reduce=true&group=true"
```

Couchdb: 8.47s

Business Questions - Aggregate

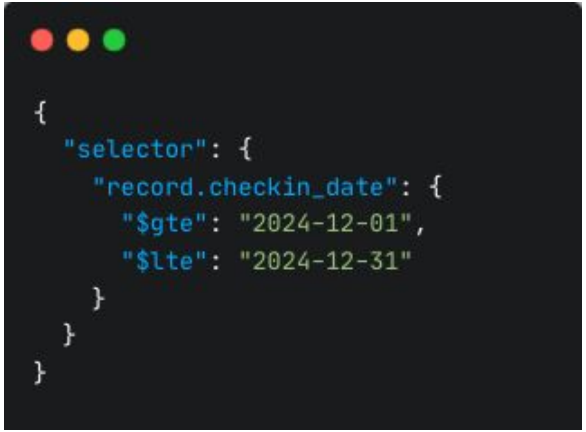


```
SELECT TO_NUMBER(record.rating_count) AS rating, COUNT(*) AS
rating_count
FROM _default
GROUP BY TO_NUMBER(record.rating_count);
```

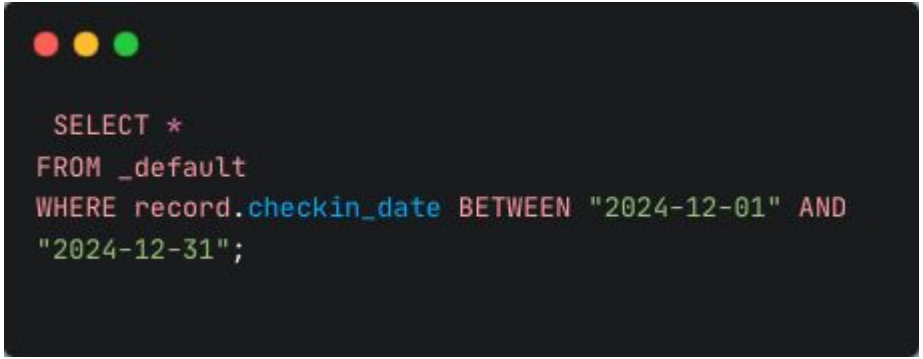
Couchbase: 5.741s

Business Questions - Report

Business Question 1: Generate a report of bookings made in December 2024.



```
{
  "selector": {
    "record.checkin_date": {
      "$gte": "2024-12-01",
      "$lte": "2024-12-31"
    }
  }
}
```



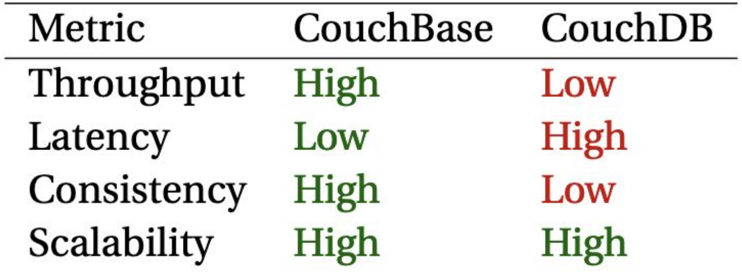
```
 SELECT *
FROM _default
WHERE record.checkin_date BETWEEN "2024-12-01" AND
"2024-12-31";
```

Couchdb: 4.8s

Couchbase: 2.95s

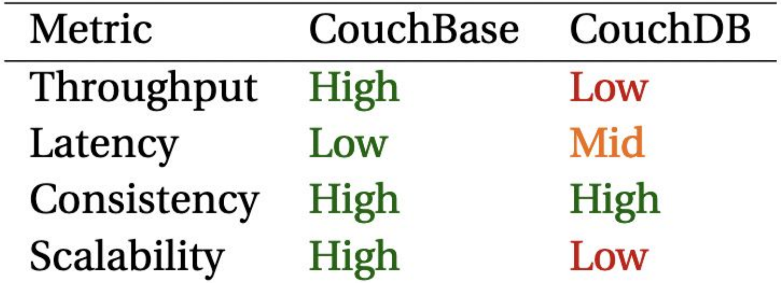
Conclusions

Benchmark Summary



Metric	CouchBase	CouchDB
Throughput	High	Low
Latency	Low	High
Consistency	High	Low
Scalability	High	High

Expected Results for Benchmark



Metric	CouchBase	CouchDB
Throughput	High	Low
Latency	Low	Mid
Consistency	High	High
Scalability	High	Low

Actual Results for Benchmark

Conclusions

- CouchBase is the overall better tool of the two for the application
  - YCSB core workload and application dataset.
- PostgreSQL seems to outperform both tools but needs further testing.
- YCSB should be extended for further tests.

Thank you for your attention!

Learn more at:
https://github.com/marwahmh/YCSB_DocumentDBs