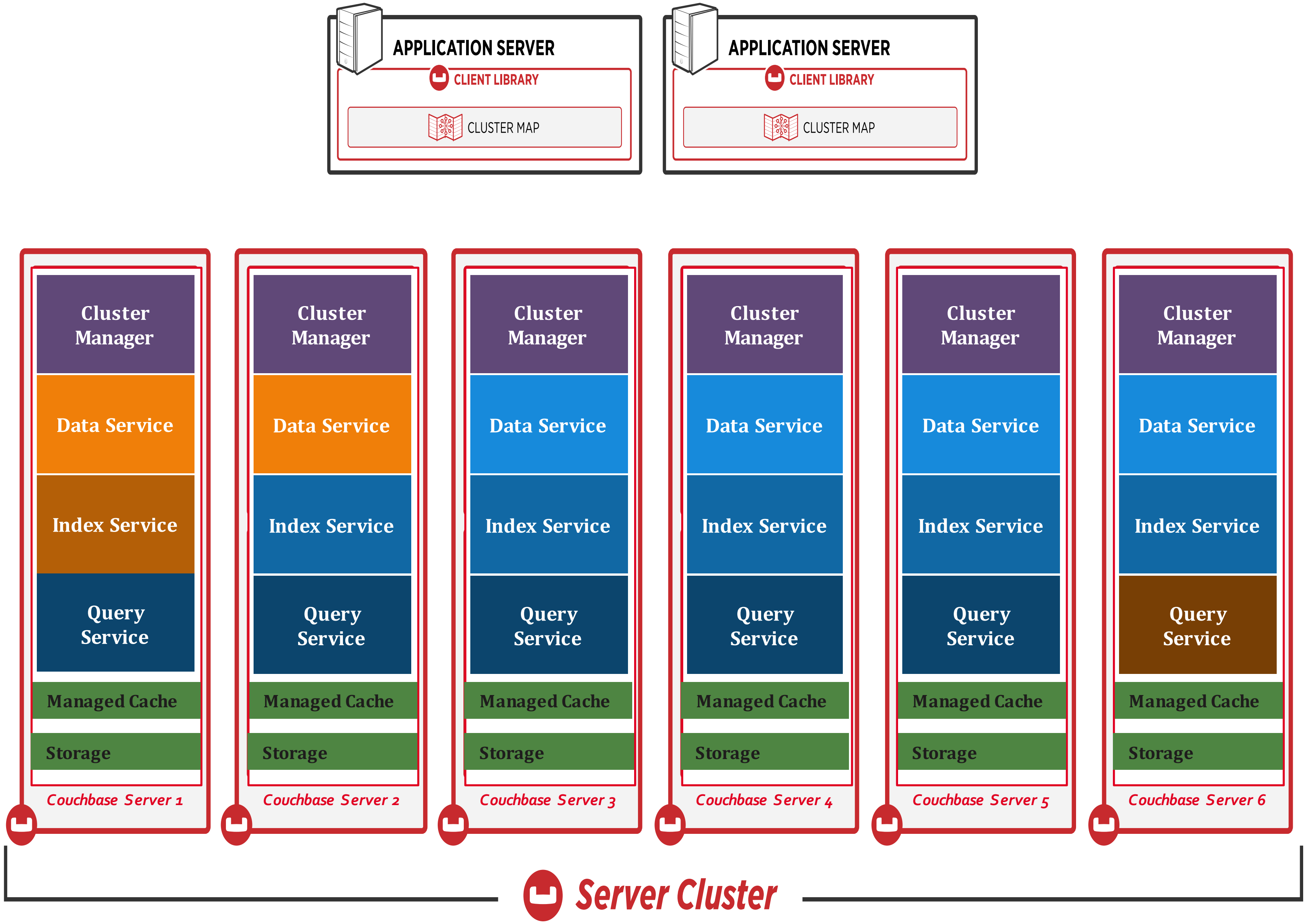
Global Secondary Indexes
new high performance indexer

Couchbase Server Cluster Architecture



HP

Indexing in Couchbase Server 4.0

- **Multiple Indexers**
  - **GSI – Index Service**

New

New indexing for N1QL for low latency queries without compromising on mutation performance (insert/update/delete)

Independently partitioned and independently scalable indexes in Indexing Service

  - **Map/Reduce Views – Data Service**

Powerful programmable indexer for complex reporting and indexing logic.

Full partition alignment and paired scalability with Data Service.

  - **Spatial View – Data Service**

Incremental R-tree indexing for powerful bounding-box queries

Full partition alignment and paired scalability with Data Service

HP

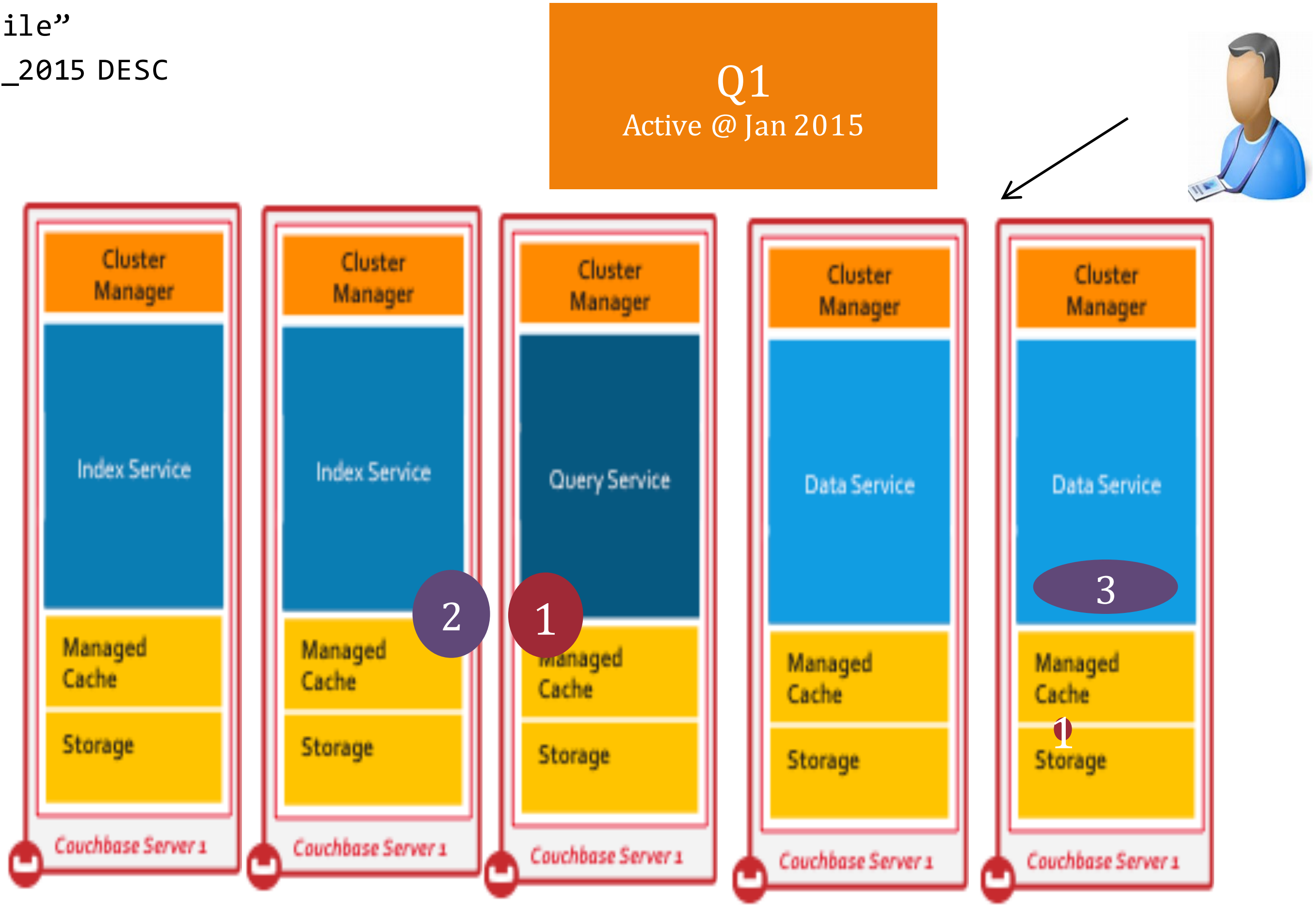
Query and Index with GSI

```
Create INDEX ON Customer_bucket(customer_name, total_logins.jan_2015)
WHERE type="customer_profile";

SELECT customer_name, total_logins.jan_2015
FROM customer_bucket
WHERE type="customer_profile"
ORDER BY total_logins.jan_2015 DESC
LIMIT 10;
```

Q1: Execution Plan on N nodes

- Execute Q1 on N1QL Service node
- Scan index on Index Service node



Introducing Global Secondary Indexes

What are Global Secondary Indexes?

High performance indexes for low latency queries with powerful caching, storage and independent placement.

- Power of GSI
  - **Fully integrated into N1QL** Query Optimization and Execution
  - **Independent Index Distribution** for Limiting scatter-gather
  - **Independent Scalability** with Index Service
  - **Powerful caching and storage** with ForestDB

HP

Which to choose – GSI vs Views

Workloads	New GSI in v4.0	Map/Reduce Views
Complex Reporting	Just In Time Aggregation	Pre-aggregated
Workload Optimization	Optimized for Scan Latency & Throughput	Optimized for Insertion
Flexible Index Logic	N1QL Functions	Javascript
Secondary Lookups	Single Node Lookup	Scatter-Gather
Tunable Consistency	Staleness false or ok or everything in between	Staleness false or ok

HP

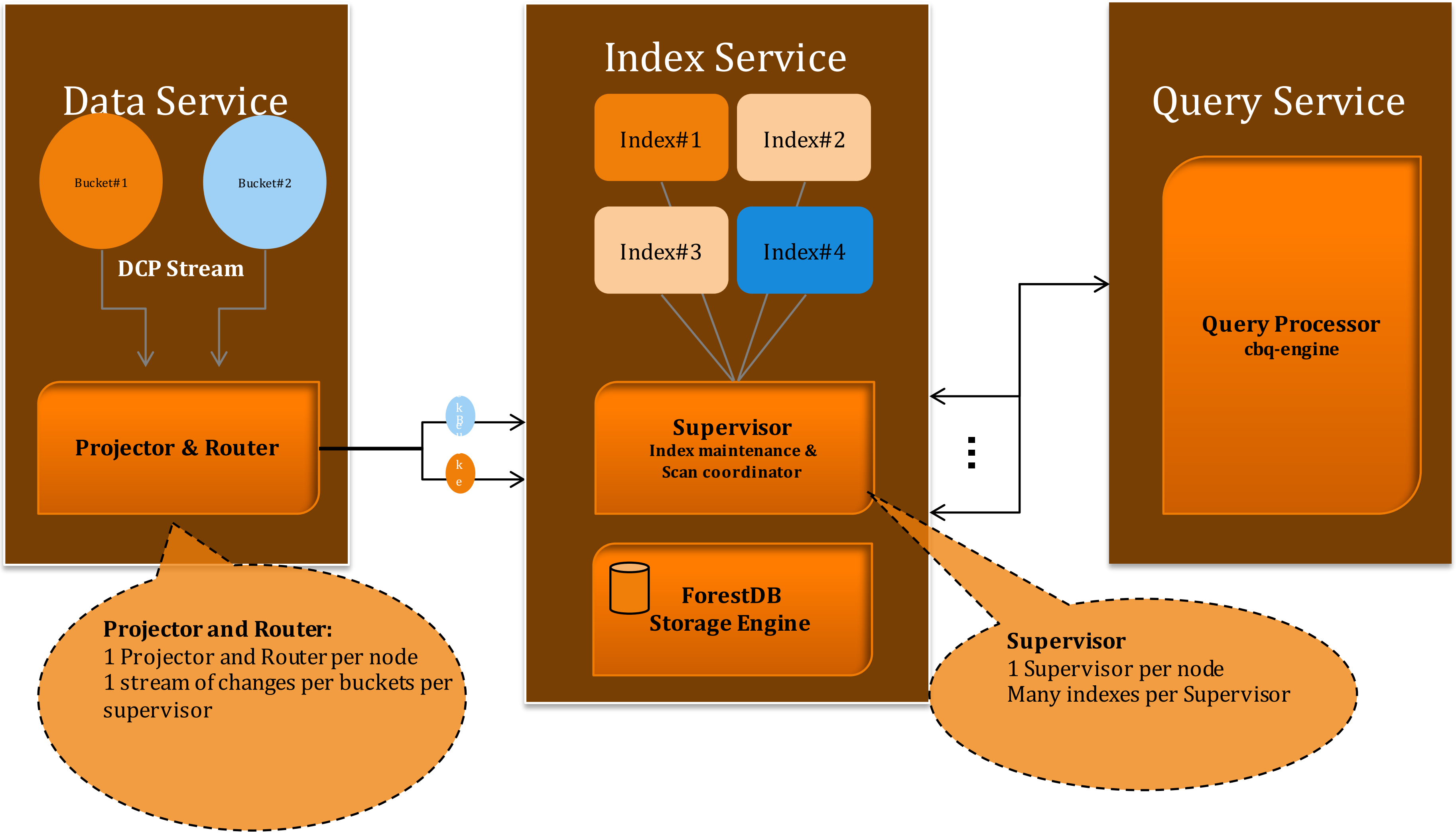
Which to choose – GSI vs Views

Capabilities	New GSI in v4.0	Map/Reduce Views
Partitioning Model	Independent – Indexing Service	Aligned to Data – Data Service
Scale Model	Independently Scale Index Service	Scale with Data Service
Fetch with Index Key	Single Node	Scatter-Gather
Range Scan	Single Node	Scatter-Gather
Grouping, Aggregates	With N1QL	Built-in with Views API
Caching	Managed	Not Managed
Storage	ForestDB	Couchstore
Availability	Multiple Identical Indexes load balanced	Replica Based

GSI Architecture

HP

Indexing Service



Projector and Router:
1 Projector and Router per node
1 stream of changes per buckets per supervisor

Supervisor
1 Supervisor per node
Many indexes per Supervisor

GSI Lifecycle

HP

Indexing Lifecycle

- Primary vs Secondary
  - Primary Index is a full list of document keys within a given bucket

```
CREATE PRIMARY INDEX index_name
ON bucket_name
USING GSI|VIEW
WITH `{"nodes”: [“node_name”], “defer_build”:true}`; //GSI-ONLY
```

  - Secondary Index is an index on a field/expression on a subset of documents for lookups

```
CREATE INDEX index_name
ON bucket_name (field/expression, …)
USING GSI|VIEW
WHERE filter_expressions
WITH `{"nodes”: [“node_name”], “defer_build”:true}`; //GSI-ONLY
```

Deferred Index Building

- Index building can be deferred to build multiple indexes all at once with greater scan efficiency.

```
CREATE INDEX … WITH {…“defer_build”:true};
CREATE INDEX … WITH {…“defer_build”:true};
…
BUILD INDEX ON bucket_name(index_name1, …) USING GSI;
```

GSI Partitioning and Placement

GSI Placement and Partitioning

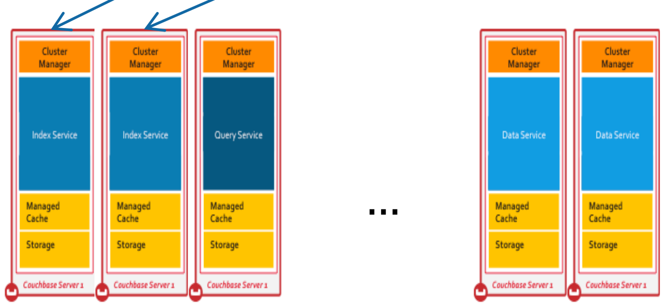
Place GSI Indexes using NODES clause

- *Each GSI reside on 1 node*
  - You can specify the node using nodes clause

```
CREATE INDEX i1 … WITH {“nodes”:”node1”};
```

  - You can scale out the index by creating identical indexes (load balanced)

```
CREATE INDEX i1 … WITH {“nodes”:”node1”};
CREATE INDEX i2 … WITH {“nodes”:”node2”};
```



GSI Placement and Partitioning

Partition Indexes Manually with WHERE clause

- You can partition with the WHERE clause and place on various nodes for scaling out

```
CREATE INDEX i1 … WHERE zipcode between “94000” and “94999” …
CREATE INDEX i2 … WHERE zipcode between “95000” and “96000” …
```

GSI Availability and Rebalance

GSI Availability and Rebalance

Use multiple identical indexes for availability

- GSI Availability
  - Create multiple identical indexes on separate nodes for availability
  - GSI will auto divert traffic if any copy goes down

- GSI & Rebalance
  - Removing/Failing a node with index service, remove the GSI indexes on that node
  - Adding a node with index service, won’t automatically move some indexes to the node.

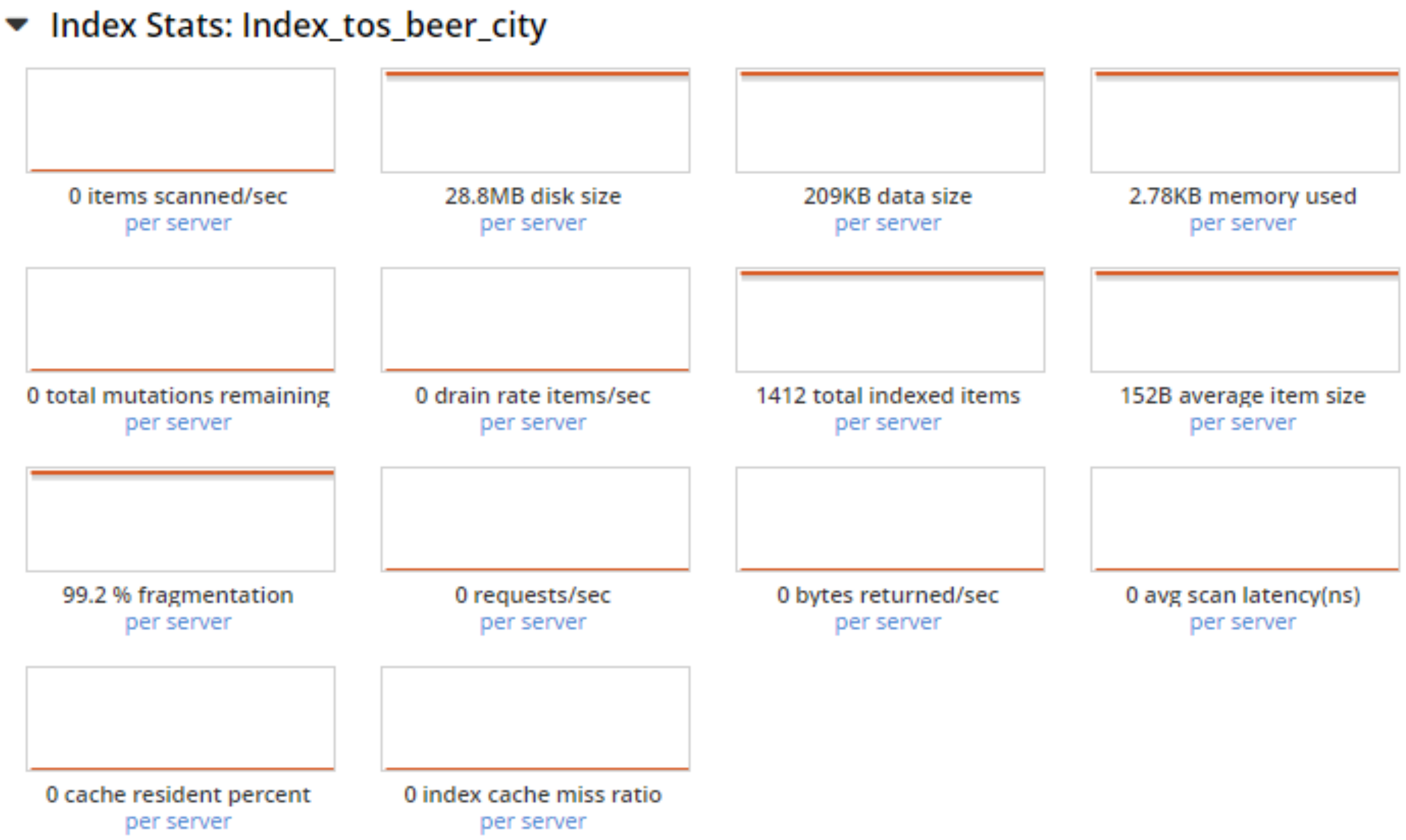
Monitoring GSI

HP

Monitoring GSI Indexes

- Index Size and Maintenance Stats
- Index Scan Stats

tos.couchbase > Servers > Statistics



Index Stats: Index_tos_beer_city

0 items scanned/sec
per server

28.8MB disk size
per server

209KB data size
per server

2.78KB memory used
per server

0 total mutations remaining
per server

0 drain rate items/sec
per server

1412 total indexed items
per server

152B average item size
per server

99.2 % fragmentation
per server

0 requests/sec
per server

0 bytes returned/sec
per server

0 avg scan latency(ns)
per server

0 cache resident percent
per server

0 index cache miss ratio
per server

Complex Types and GSI

- Indexing Complex Types
  - Sub-documents attributes

```
CREATE INDEX ifriend_id ON default(friends.id)
USING GSI;

SELECT * FROM default
WHERE friends.id= "002819”;
```

```
{
 "id": "000000000000001",
 "desc": ”---",
 "type": "friends",
 "tags": [0,1,2,3,4,5,6,7,8,9],
 "friends": {
   "id": "002819",
   "class": "005"
  }
}
```

Complex Types and GSI

- Indexing Complex Types
  - Compound Keys

```
CREATE INDEX ifriends_id_class ON
default(friends.id, friends.class) USING GSI;

SELECT * FROM default
WHERE friends.id="002819" and friends.class="005”;
```

```
{
 "id": "00000000000001",
 "desc": ”---",
 "type": "friends",
 "tags": [0,1,2,3,4,5,6,7,8,9],
 "friends": {
   "id": "002819",
   "class": "005"
  }
}
```

HP

Complex Types and GSI

- Indexing Complex Types
  - Sub-documents

```
CREATE INDEX ifriend ON default(friends) USING GSI;

SELECT * FROM default
WHERE friends= {"class": "005","id":"002819"};
```

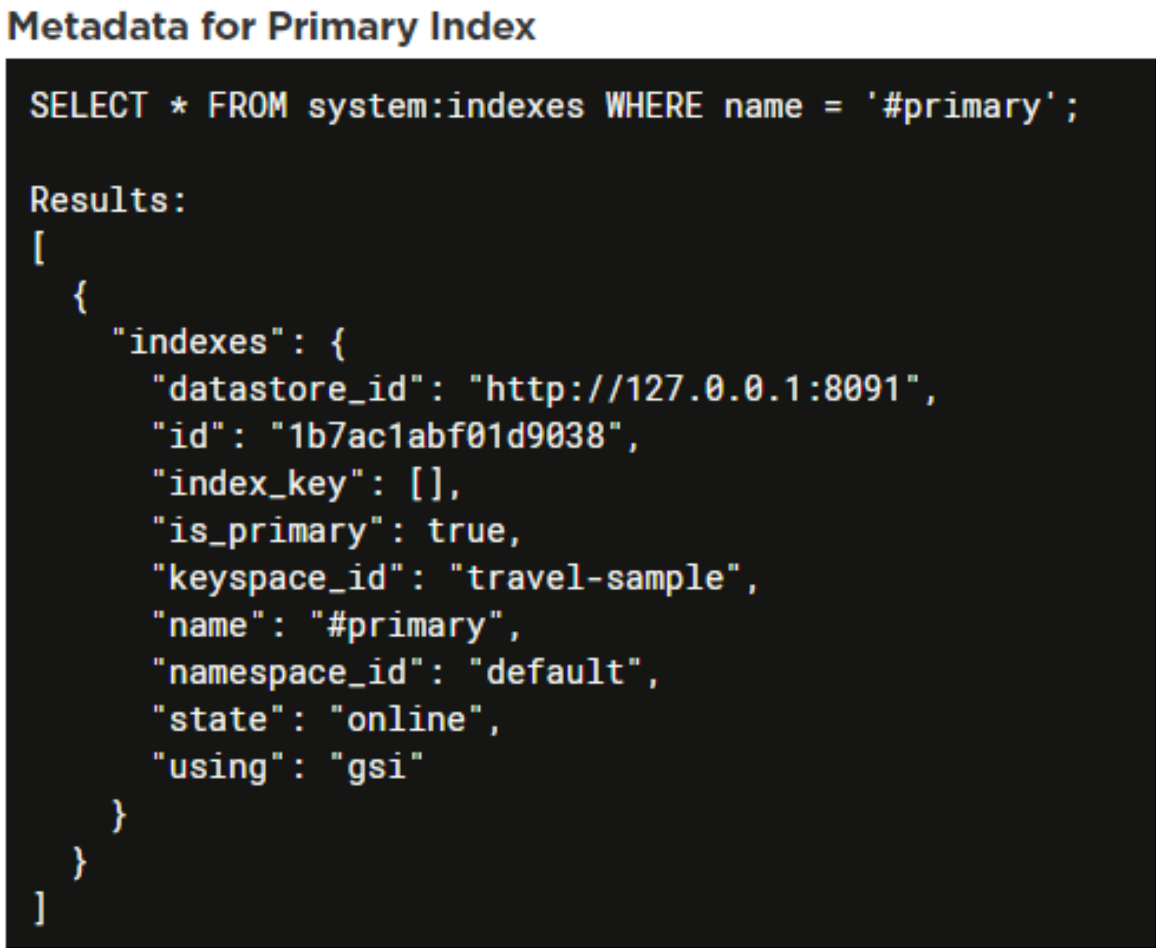
```
{
 "id":"000000000000001",
 "desc":”---",
 "type":"friends",
 "tags": [0,1,2,3,4,5,6,7,8,9],
 "friends":{
   "id":"002819",
   "class":"005"
   }
}
```

Types of Indexes

HP

Primary Index

- CREATE PRIMARY INDEX ON `travel-sample`;



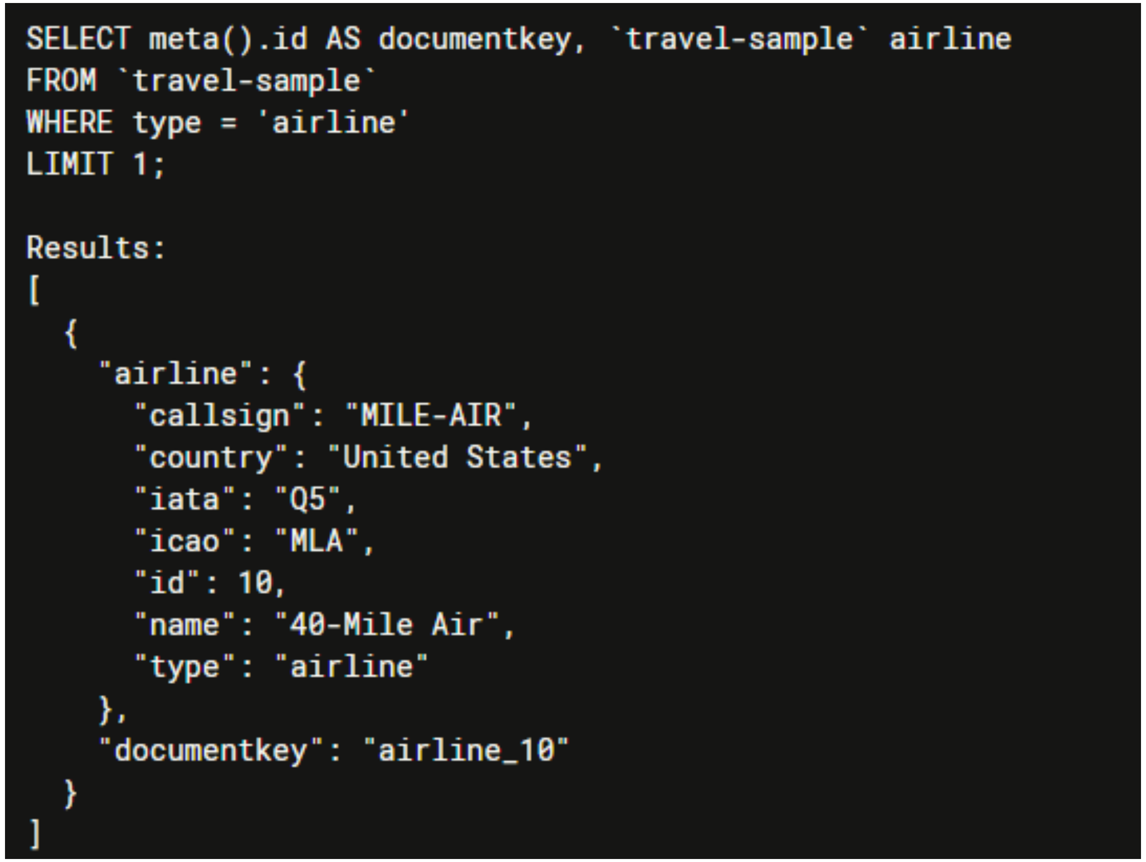
Metadata for Primary Index

```
SELECT * FROM system:indexes WHERE name = '#primary';

Results:
[
  {
    "indexes": {
      "datastore_id": "http://127.0.0.1:8091",
      "id": "1b7ac1abf01d9038",
      "index_key": [],
      "is_primary": true,
      "keyspace_id": "travel-sample",
      "name": "#primary",
      "namespace_id": "default",
      "state": "online",
      "using": "gsi"
    }
  }
]
```

Named Primary Index

- CREATE PRIMARY INDEX def_primary ON `travel-sample`;

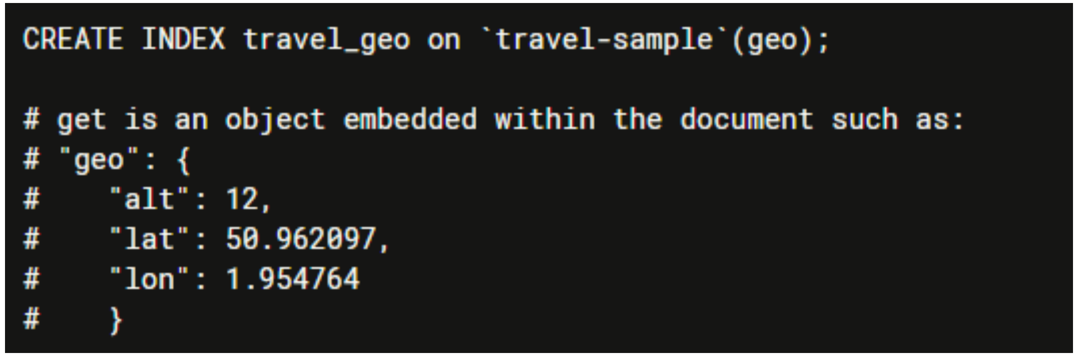


```
SELECT meta().id AS documentkey, `travel-sample` airline
FROM `travel-sample`
WHERE type = 'airline'
LIMIT 1;

Results:
[
  {
    "airline": {
      "callsign": "MILE-AIR",
      "country": "United States",
      "iata": "Q5",
      "icao": "MLA",
      "id": 10,
      "name": "40-Mile Air",
      "type": "airline"
    },
    "documentkey": "airline_10"
  }
]
```

Secondary Index

- CREATE INDEX travel_name ON `travel-sample`(name);



```
CREATE INDEX travel_geo on `travel-sample`(geo);

# get is an object embedded within the document such as:
# "geo": {
#     "alt": 12,
#     "lat": 50.962097,
#     "lon": 1.954764
#     }
```

Composite Secondary Index

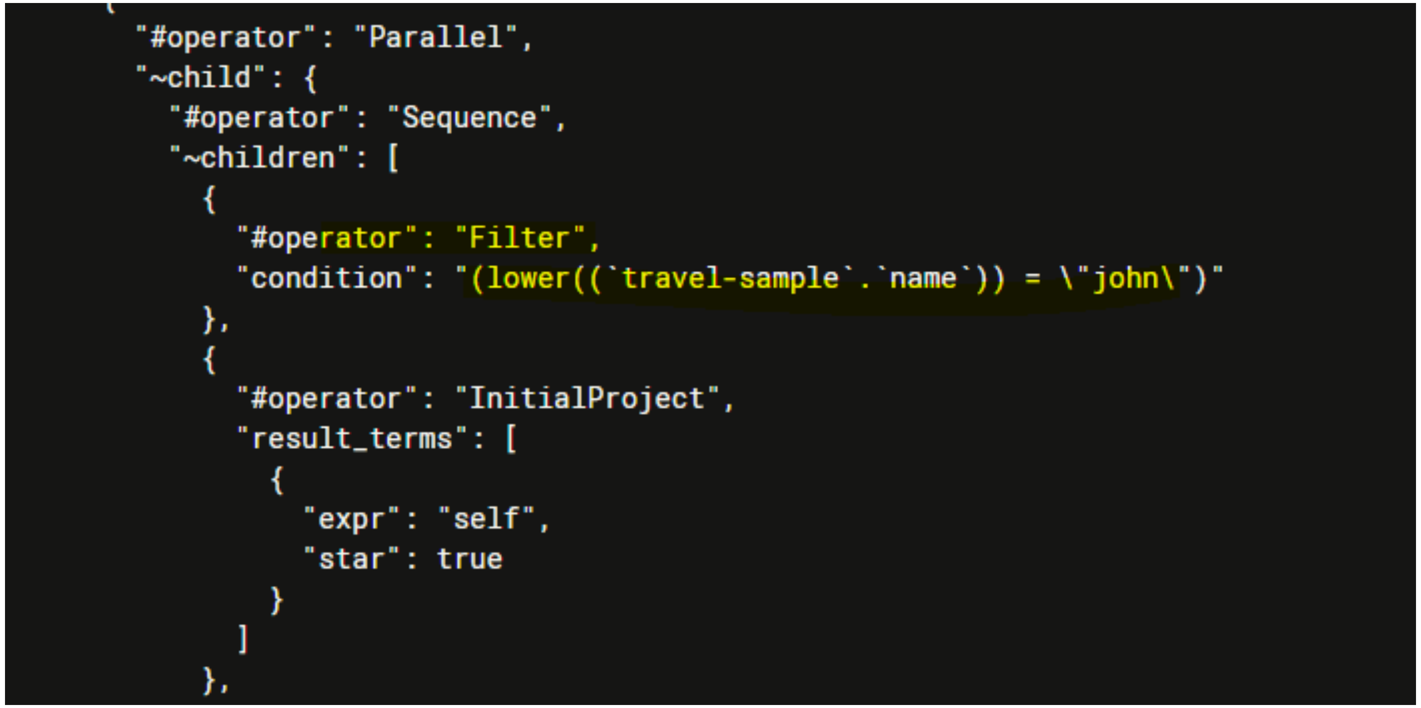
- CREATE INDEX travel_info ON `travel-sample`(name,type,id,icoo,iata);

Each of the keys can be a simple scalar field, object, or an array. For the index filtering to be exploited, the filters have to use respective object type in the query filter.

HP

Functional Index

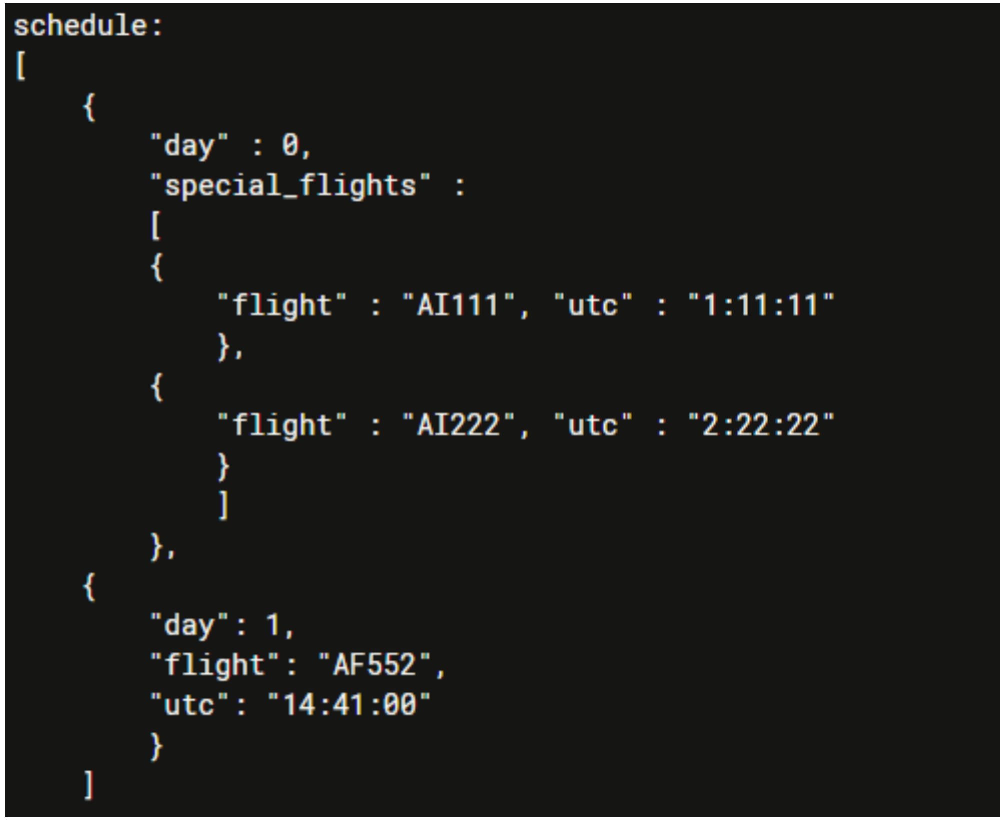
- CREATE INDEX travel_cxname ON `travel-sample`(LOWER(name));



```
  "#operator": "Parallel",
  "~child": {
    "#operator": "Sequence",
    "~children": [
      {
        "#operator": "Filter",
        "condition": "(lower((`travel-sample`.`name`)) = \"john\")"
      },
      {
        "#operator": "InitialProject",
        "result_terms": [
          {
            "expr": "self",
            "star": true
          }
        ]
      },
```

Array Index

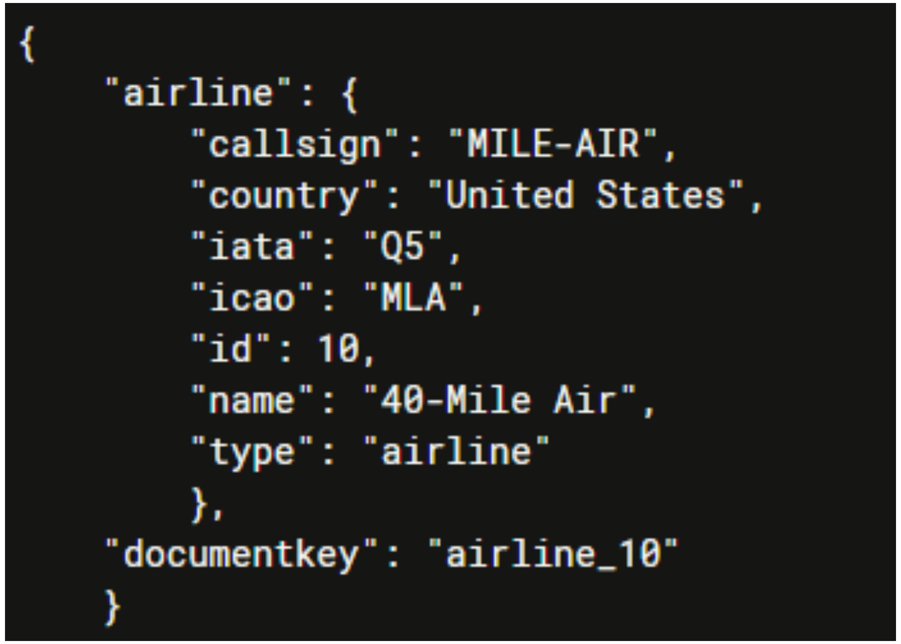
- CREATE INDEX travel_sched ON `travel-sample`
  (ALL DISTINCT ARRAY v.day FOR v IN schedule END);



```
schedule:
[
    {
        "day" : 0,
        "special_flights" :
        [
        {
            "flight" : "AI111", "utc" : "1:11:11"
            },
        {
            "flight" : "AI222", "utc" : "2:22:22"
            }
            ]
        },
    {
        "day": 1,
        "flight": "AF552",
        "utc": "14:41:00"
        }
    ]
```

Partial Index

- CREATE INDEX travel_info ON `travel-sample`(name, id, icoo, iata) WHERE type='airline';



```
{
    "airline": {
        "callsign": "MILE-AIR",
        "country": "United States",
        "iata": "Q5",
        "icao": "MLA",
        "id": 10,
        "name": "40-Mile Air",
        "type": "airline"
        },
    "documentkey": "airline_10"
    }
```

HP

Duplicate Index

- CREATE INDEX i1 ON `travel-sample`(LOWER(name),id, icoo)
WHERE type = 'airline';

- CREATE INDEX i2 ON `travel-sample`(LOWER(name),id, icoo)
WHERE type = 'airline';

- CREATE INDEX i3 ON `travel-sample`(LOWER(name),id, icoo)
WHERE type = 'airline';

All three indexes have identical keys and an identical WHERE clause; the only difference is the name of these indexes. You can choose their physical location using the WITH clause of the CREATE INDEX statement.

providing scale-out, multi-dimensional scaling, performance, and high availability

Covering Index

- Index selection for a query solely depends on the filters in the WHERE clause of your query.
- After the index selection is made, the query engine analyzes the query to see if it can be answered using only the data in the index

- CREATE INDEX travel_info ON `travel-sample`(name, id, icoo, iata)

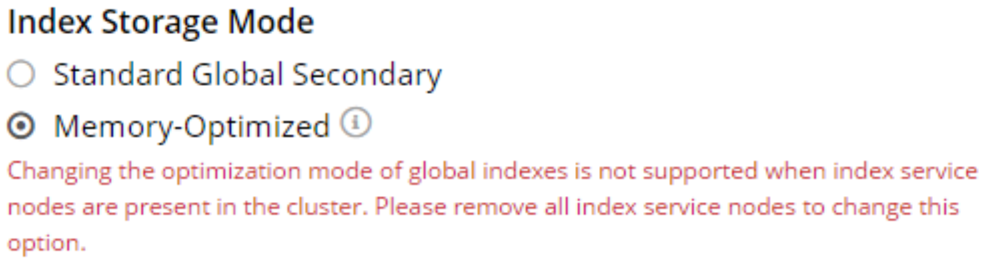
Memory-Optimized Index Storage

- Memory-optimized index-storage allows high-speed maintenance and scanning; since the index is kept fully in memory at all times.
- A snapshot of the index is maintained on disk, to permit rapid recovery if node-failures are experienced.
- less suitable for nodes where memory is constrained;

Standard Index Storage

- *Standard* is the default storage-setting for Secondary Indexes: the indexes are saved on disk; in a disk-optimized format that uses both memory and disk for index-update and scanning.
- The performance of standard index storage depends on overall I/O performance.

Settings are established at cluster-initialization for all indexes on the cluster, across all buckets. Following cluster-initialization, to change from one setting to the other, all nodes running the Index Service must be removed. If the cluster is single-node, uninstall and reinstall Couchbase Server.



Index Storage Mode

○ Standard Global Secondary

● Memory-Optimized ⓘ

Changing the optimization mode of global indexes is not supported when index service nodes are present in the cluster. Please remove all index service nodes to change this option.

Labs:
Query Workbench & Index– 60 Minutes(D)