Lab: Designing Index for Query in Couchbase N1QL

Every query has a query plan. The performance and efficiency of a query depend on its plan. The N1QL query engine prepares the query plan for each query. N1QL uses a rule-based optimizer. The creation and selection of the right index have a major influence on both performance and efficiency.

This lab will walk through the step-by-step process of how to create an index and modify the query for optimal performance.

We'll use travel-sample dataset shipped with Couchbase. Install travel-sample shipped with Couchbase 4.5.1, then drop all of the indexes to start with a clean slate.

Step 0

```
DROP INDEX `travel-sample`.def_city;
DROP INDEX `travel-sample`.def_faa;
DROP INDEX `travel-sample`.def_icao;
DROP INDEX `travel-sample`.def_name_type;
DROP INDEX `travel-sample`.def_airportname;
DROP INDEX `travel-sample`.def_schedule_utc;
DROP INDEX `travel-sample`.def_type;
DROP INDEX `travel-sample`.def_sourceairport;
DROP INDEX `travel-sample`.def_route_src_dst_day;
DROP INDEX `travel-sample`.def_primary;
```

We will walk through the step-by-step process and create the index for the following query. In this query, we're looking to get airlines in the United States with an id between zero and 1000, ordered by id. We're interested in the 10 airlines, ordered by id and skipping the first five.

```
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
AND country = "United States"
AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;
```

Step 1: Using Primary Index

The primary index is the simplest index, indexing all of the documents in `travel-sample`. Any query can exploit the primary index to execute the query.

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

SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
AND country = "United States"
    AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;

{
     "results": [
        {
            "country": "United States",
            "id": 149,
            "name": "Air Cargo Carriers"
        },
        {
            "country": "United States",
            "id": 210,
            "name": "Airlift International"
        },
        {
            "country": "United States",
            "id": 281,
            "name": "America West Airlines"
        },
        {
            "country": "United States",
            "id": 282,
            "name": "Air Wisconsin"
        },
        {
            "country": "United States",
            "id": 287,
            "name": "Allegheny Commuter Airlines"
        },
        {
            "country": "United States",
            "id": 295,
            "name": "Air Sunshine"
        },
        {
            "country": "United States",
            "id": 315,
```

```
            "name": "ATA Airlines"
        },
        {
            "country": "United States",
            "id": 397,
            "name": "Arrow Air"
        },
        {
            "country": "United States",
            "id": 452,
            "name": "Atlantic Southeast Airlines"
        },
        {
            "country": "United States",
            "id": 659,
            "name": "American Eagle Airlines"
        }
    ]
}
```

```
EXPLAIN
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
AND country = "United States"
    AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;
```

```
    "~children": [
            {
               "#operator": "PrimaryScan",
               "index": "def_primary",
               "keyspace": "travel-sample",
               "namespace": "default",
              "using": "gsi"
            },
            {
           "#operator": "Fetch",
               "keyspace": "travel-sample",
               "namespace": "default"
            }
      ]
```

While the query plan tells you indexes selected and predicates pushed down, it won't tell you the amount of work (index items scanned, documents fetched) to execute the query.

```
SELECT * FROM system:completed_requests;

{
    "completed_requests": {
        "ElapsedTime": "1.219329819s",
        "ErrorCount": 0,
        "PhaseCounts": {
            "Fetch": 31591,
            "PrimaryScan": 31591,
            "Sort": 33
        },
        "PhaseOperators": {
            "Fetch": 1,
            "PrimaryScan": 1,
            "Sort": 1
        },
        "RequestId": "cad6fcc0-88cc-4329-ba56-e05e8ef6a53e",
        "ResultCount": 10,
        "ResultSize": 1153,
        "ServiceTime": "1.219299801s",
        "State": "completed",
        "Statement": "SELECT country, id, name FROM `travel-sample` WHE
RE type = \"airline\" AND country = \"United States\" AND id BETWEEN 0
AND 1000 ORDER BY id LIMIT 10 OFFSET 5",
        "Time": "2016-12-08 12:00:02.093969361 -0800 PST"
    }
}
```

The travel-sample bucket has the following type of documents.

Type of document	Count
airline	187
airport	1968
route	24024
landmark	4495
hotel	917
total	3591

Primary Index Scan gets all the item keys from the index and the query engine fetches the items, applies the predicate, sorts the results, and then paginates to produces 10 items.

Query engine processed 31591 items to produce 10 items.

Step 2: Using Secondary Index

Let's create a secondary index on type.

```
CREATE INDEX ts_ix1 ON `travel-sample`(type);


EXPLAIN
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
        AND country = "United States"
        AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;
{
.......
   "~children": [
        {
            "#operator": "IndexScan",
            "index": "ts_ix1",
            "index_id": "dd95586d9994d427",
            "keyspace": "travel-sample",
            "namespace": "default",
            "spans": [
                {
                    "Range": {
                        "High": [
                            "\"airline\""
                        ],
                        "Inclusion": 3,
                        "Low": [
                            "\"airline\""
                        ]
                    }
                }
            ],
            "using": "gsi"
        }
    ]
........
}
```

The query uses a secondary index and pushes type predicate down to indexer. This results in 187 items.

id and country predicates are not pushed down to the indexer; they are applied after fetching the items.

Step 3: Using Composite Index on All Attributes in Query Predicates

Let's drop the index created in the previous step and create a composite index based on type, id, and country.

```
DROP INDEX `travel-sample`.ts_ix1;

CREATE INDEX ts_ix1 ON `travel-sample`(type, id, country);

EXPLAIN
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
        AND country = "United States"
        AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;

{
........
    "~children": [
        {
            "#operator": "IndexScan",
            "index": "ts_ix1",
            "index_id": "f725e59b0adf5875",
            "keyspace": "travel-sample",
            "namespace": "default",
            "spans": [
                {
                    "Range": {
                        "High": [
                            "\"airline\"",
                            "1000",
                            "\"United States\""
                        ],
                        "Inclusion": 3,
                        "Low": [
                            "\"airline\"",
                            "0",
                            "\"United States\""
                        ]
                    }
                }
```

```
            ],
            "using": "gsi"
        }
........
}
```

The query uses a composite index and pushes all the predicates down to indexer. This results in 18+ items.

This index has airport, route, landmark, and hotel items as well. We can restrict the index only to airline items.

Step 4: Using Partial Composite Index

Let's drop the index created in the previous step and create a partial composite index.

```
DROP INDEX `travel-sample`.ts_ix1;

CREATE INDEX ts_ix1 ON `travel-sample`(id, country) WHERE type = "airli
ne";

EXPLAIN
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
        AND country = "United States"
        AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;

{
........
    "~children": [
        {
            "#operator": "IndexScan",
            "index": "ts_ix1",
            "index_id": "276fabb0b7375a64",
            "keyspace": "travel-sample",
            "namespace": "default",
            "spans": [
                {
                    "Range": {
                        "High": [
                            "1000",
                            "\"United States\""
                        ],
                        "Inclusion": 3,
                        "Low": [
                            "0",
                            "\"United States\""
                        ]
                    }
```

```
                }
            ],
            "using": "gsi"
        }
........
}
```

This index has only items that have type = "airline."

Index condition (type = "airline") is equality predicate, so it is not required to include it in the index keys.

This makes the index lean and makes it perform better by reducing I/O, memory, CPU, and network.

It pushes all the predicates down to the indexer. This results in 18+ items.

Step 5: Using Covering Partial Composite Index

The query uses type, id, country, and name.

The type is part of the index condition as equality predicate and N1QL can derive the type value from the index condition and answer the query.

The id and country are index keys.

The only missing attribute is a name. Let's add it as trailing index key.

Let's drop the index created in the previous step and create a covering partial composite index.

```
DROP INDEX `travel-sample`.ts_ix1;

CREATE INDEX ts_ix1 ON `travel-sample`(id, country,name) WHERE type = "
airline";

EXPLAIN
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
        AND country = "United States"
        AND id BETWEEN 0 AND 1000
ORDER BY id
LIMIT 10
OFFSET 5;
```

```
{
    "requestID": "8763f612-ab85-4d53-ad83-4c0d5248787b",
    "signature": "json",
    "results": [
        {
            "plan": {
                "#operator": "Sequence",
                "~children": [
                    {
                        "#operator": "Sequence",
                        "~children": [
                            {
                                "#operator": "IndexScan",
                                "covers": [
                                    "cover ((`travel-sample`.`id`))",
                                    "cover ((`travel-sample`.`country`)
)",
                                    "cover ((`travel-sample`.`name`))",
                                    "cover ((meta(`travel-sample`).`id`
))"
                                ],
                                "filter_covers": {
                                    "cover ((`travel-sample`.`type`))":
 "airline"
                                },
                                "index": "ts_ix1",
                                "index_id": "64ecc9c1396eb225",
                                "keyspace": "travel-sample",
                                "namespace": "default",
                                "spans": [
                                    {
                                        "Range": {
                                            "High": [
                                                "1000",
                                                "successor(\"United Sta
tes\")"
                                            ],
                                            "Inclusion": 1,
                                            "Low": [
                                                "0",
                                                "\"United States\""
                                            ]
                                        }
                                    }
                                ],
                                "using": "gsi"
                            },
                            {
                                "#operator": "Parallel",
                                "maxParallelism": 1,
                                "~child": {
                                    "#operator": "Sequence",
                                    "~children": [
                                        {
                                            "#operator": "Filter",
                                            "condition": "(((cover ((`t
ravel-sample`.`type`)) = \"airline\") and (cover ((`travel-sample`.`cou
```

```
ntry`)) = \"United States\")) and (cover ((`travel-sample`.`id`)) betwe
en 0 and 1000))"
                                        },
                                        {
                                            "#operator": "InitialProjec
t",
                                            "result_terms": [
                                                {
                                                    "expr": "cover ((`t
ravel-sample`.`country`))"
                                                },
                                                {
                                                    "expr": "cover ((`t
ravel-sample`.`id`))"
                                                },
                                                {
                                                    "expr": "cover ((`t
ravel-sample`.`name`))"
                                                }
                                            ]
                                        }
                                    ]
                                }
                            }
                        ]
                    },
                    {
                        "#operator": "Offset",
                        "expr": "5"
                    },
                    {
                        "#operator": "Limit",
                        "expr": "10"
                    },
                    {
                        "#operator": "FinalProject"
                    }
                ]
            },
            "text": "SELECT country, id, name FROM `travel-sample` WHER
E type = \"airline\" AND country = \"United States\" AND id BETWEEN 0 A
ND 1000 ORDER BY id LIMIT 10 OFFSET"
        }
    ]
}
```

The query uses covering partial index and pushes all the predicates down to the indexer. This results in 18+ items.

The index has all the information required by the query.

Query avoids fetch and answers from the index scan.

This query is called a Covered Query and the index is called Covering Index.

Step 6: Query Exploit Index Order

Index stores data pre-sorted by the index keys. When the ORDER BY list matches the INDEX keys list order left to right, the query can exploit index order.

Query ORDER BY expressions match index keys from left to right.

IndexScan has a single span and the query doesn't have any JOINs, GROUP BY, or other clauses that can change the order or granularity of items produced by indexer.

The query can exploit index order and avoid expensive sort and fetch.

The EXPLAIN (step 5) will not have an Order section.

Step 7: OFFSET Pushdown to Indexer

Providing pagination hints to the indexer tells the indexer how many items it can produce and reduce unnecessary work. This allows scaling the workload in the cluster.

Currently, the offset is not pushed down to the indexer separately.

If the LIMIT is present, the Query Engine pushes down limit as limit + offset and applies limit and offset separately.

Step 8: LIMIT Pushdown to Indexer

Providing pagination hints(LIMIT) to indexer. This tells indexer maximum items it should produce. This reduces unnecessary work by indexer and Query Engine, helping to scale.

With the technique used in Step 5, LIMIT is not pushed to indexer. Let's see how we can push the LIMIT to indexer.

LIMIT can be pushed down to the indexer:

- When the query does not have ORDER BY clause OR query uses index order.
- When exact predicates are pushed down to the indexer and the Query Engine will not eliminate any documents (i.e., no false positives from indexer).

In the case of a composite key, if leading keys predicates are non-equality, the indexer will produce false positives.

The id is the range predicate; LIMIT will not be pushed to the indexer. The EXPLAIN (step 5) will not have a "limit" in the IndexScan section.

The query has an equality predicate on "country"; if "country" is the leading index key followed by the id, the indexer will not produce any false positives and allows

LIMIT push down to the indexer. However, the query ORDER BY expressions and index keys order are different; this disallows LIMIT push-down and requires sorting.

By closely looking at the query, we have "country" as an equality predicate. If we change the query to include the ORDER BY country, the id query outcome will not change and we can use index order and push the LIMIT down to indexer. In 4.6.0, this is detected automatically and no query changes are required.

Let's drop the index created in the previous step and create a covering partial composite index.

```
DROP INDEX `travel-sample`.ts_ix1;

CREATE INDEX ts_ix1 ON `travel-sample`(country, id, name) WHERE type =
"airline";

EXPLAIN
SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
        AND country = "United States"
        AND id BETWEEN 0 AND 1000
ORDER BY country, id
LIMIT 10
OFFSET 5;

{
    "requestID": "922e2b13-4152-4327-be02-49463e150ced",
    "signature": "json",
    "results": [
        {
            "plan": {
                "#operator": "Sequence",
                "~children": [
                    {
                        "#operator": "Sequence",
                        "~children": [
                            {
                                "#operator": "IndexScan",
                                "covers": [
                                    "cover ((`travel-sample`.`country`)
)",
                                    "cover ((`travel-sample`.`id`))",
                                    "cover ((`travel-sample`.`name`))",
                                    "cover ((meta(`travel-sample`).`id`
))"
                                ],
                                "filter_covers": {
                                    "cover ((`travel-sample`.`type`))":
 "airline"
                                },
                                "index": "ts_ix1",
                                "index_id": "a51d9f2255cfb89e",
                                "keyspace": "travel-sample",
```

```
                                "limit": "(5 + 10)",
                                "namespace": "default",
                                "spans": [
                                    {
                                        "Range": {
                                            "High": [
                                                "\"United States\"",
                                                "successor(1000)"
                                            ],
                                            "Inclusion": 1,
                                            "Low": [
                                                "\"United States\"",
                                                "0"
                                            ]
                                        }
                                    }
                                ],
                                "using": "gsi"
                            },
                            {
                                "#operator": "Parallel",
                                "maxParallelism": 1,
                                "~child": {
                                    "#operator": "Sequence",
                                    "~children": [
                                        {
                                            "#operator": "Filter",
                                            "condition": "(((cover ((`t
ravel-sample`.`type`)) = \"airline\") and (cover ((`travel-sample`.`cou
ntry`)) = \"United States\")) and (cover ((`travel-sample`.`id`)) betwe
en 0 and 1000))"
                                        },
                                        {
                                            "#operator": "InitialProjec
t",
                                            "result_terms": [
                                                {
                                                    "expr": "cover ((`t
ravel-sample`.`country`))"
                                                },
                                                {
                                                    "expr": "cover ((`t
ravel-sample`.`id`))"
                                                },
                                                {
                                                    "expr": "cover ((`t
ravel-sample`.`name`))"
                                                }
                                            ]
                                        }
                                    ]
                                }
                            }
                        ]
                    },
                    {
                        "#operator": "Offset",
```

```
                        "expr": "5"
                    },
                    {
                        "#operator": "Limit",
                        "expr": "10"
                    },
                    {
                        "#operator": "FinalProject"
                    }
                ]
            },
            "text": "SELECT country, id, name FROM `travel-sample` WHER
E type = \"airline\" AND country = \"United States\" AND id BETWEEN 0 A
ND 1000 ORDER BY country, id LIMIT 10 OFFSET 5"
        }
    ]
}
```

The EXPLAIN will have “limit” in the index section and will not have order section, The value of the “limit” is the value of limit + offset.

Final Query and Index

```
CREATE INDEX ts_ix1 ON `travel-sample`(country, id, name) WHERE type =
"airline";

SELECT country, id, name
FROM `travel-sample`
WHERE type = "airline"
AND country = "United States"
    AND id BETWEEN 0 AND 1000
ORDER BY country, id
LIMIT 10
OFFSET 5;
```

Summary of the Lab

While designing the index, explore all index options. Include as many query predicates as possible in the leading index keys (equality, IN, less than, (less than or equal), greater than, (greater than or equal)), followed by other attributes used in your query. Use partial indexes to focus your queries on the relevant data.