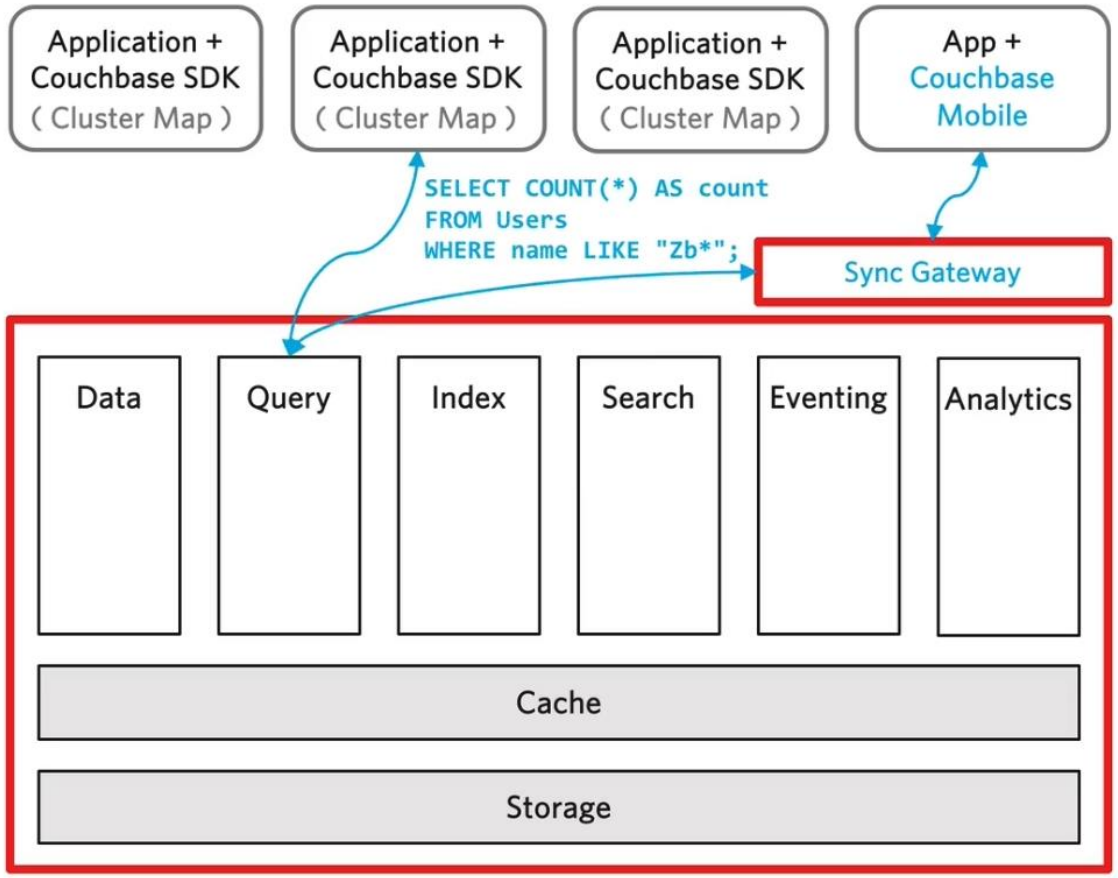
CRUD in couchbase

ANJU MUNOTH

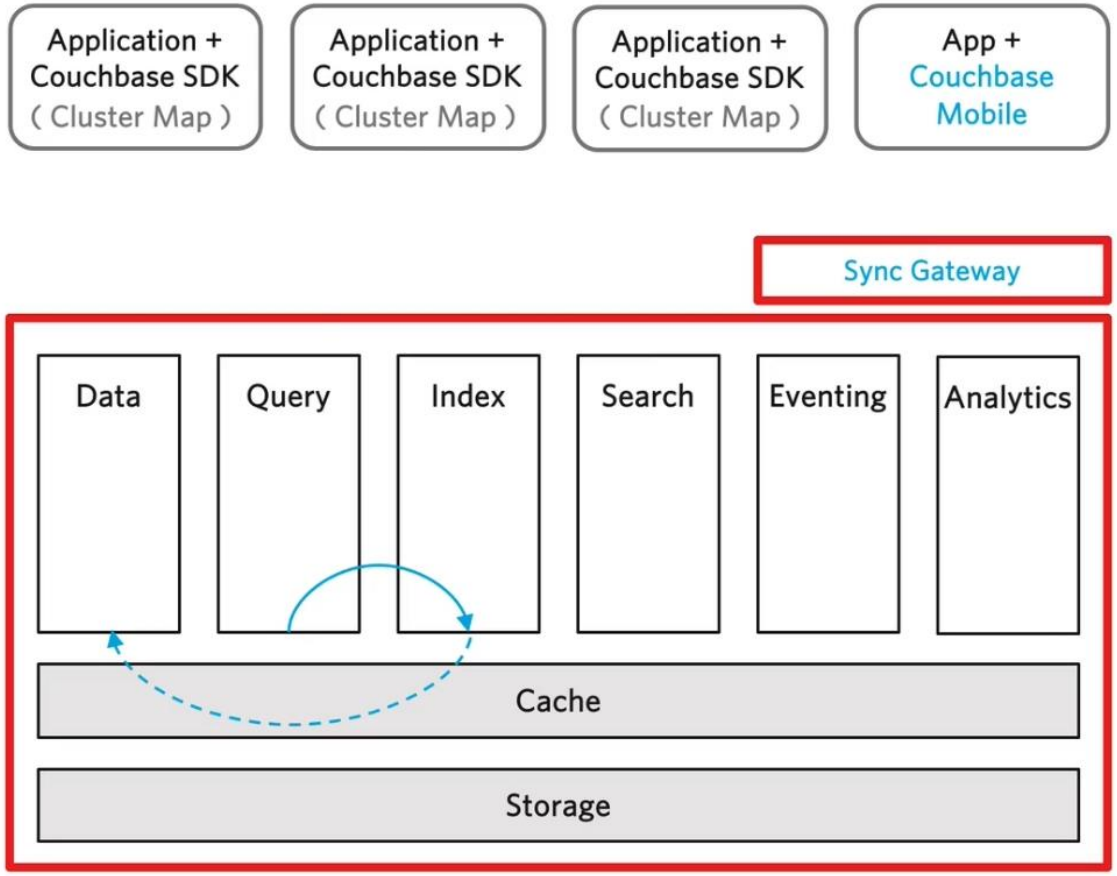
How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution



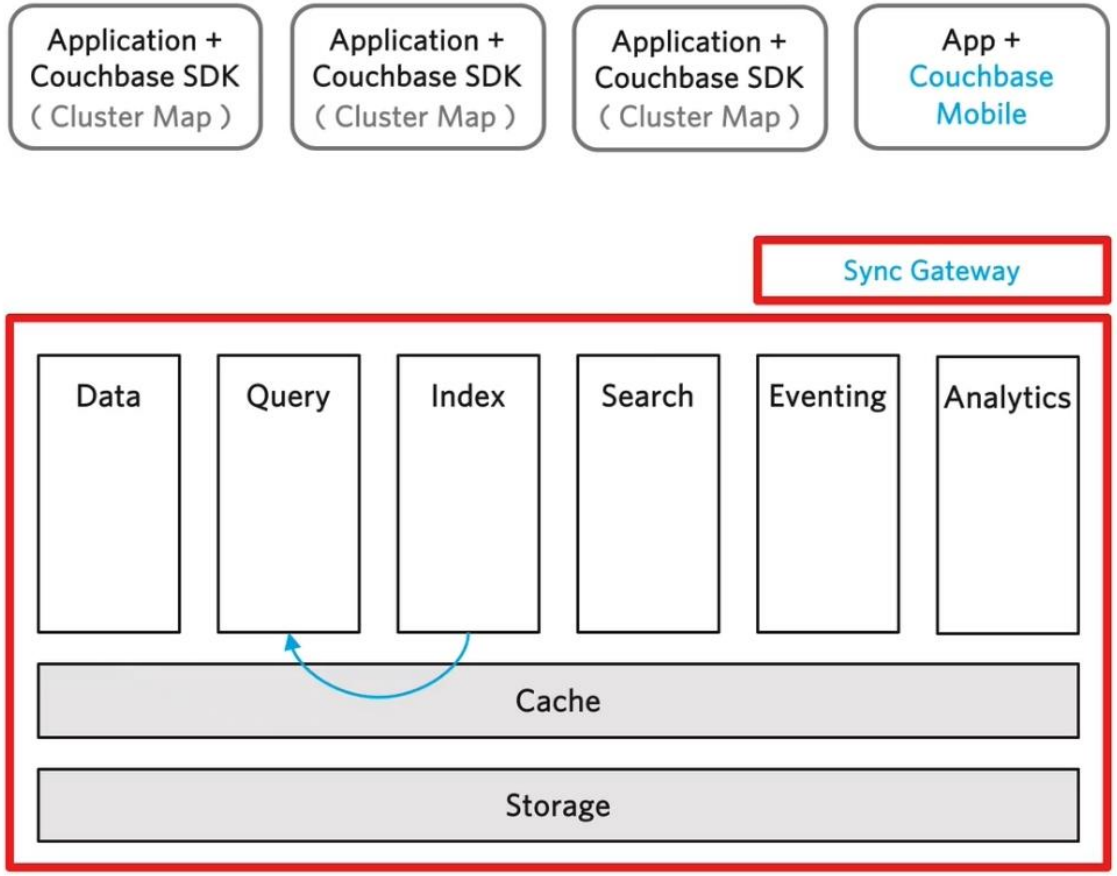
How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution
- ✓ Query Service sends request to Index Service
  Continuously maintains defined indexes



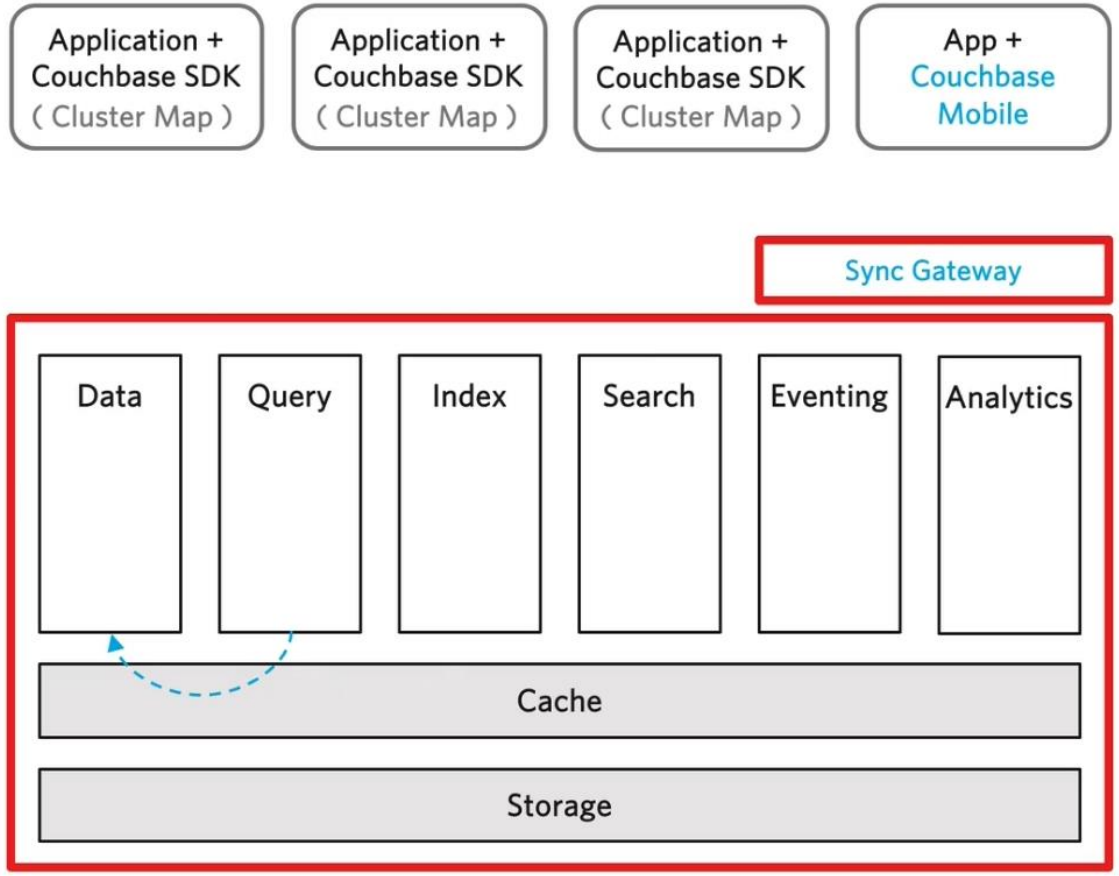
How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution
- ✓ Query Service sends request to Index Service
  Continuously maintains defined indexes
- ✓ Index Service returns Doc IDs & indexed data to Query Service



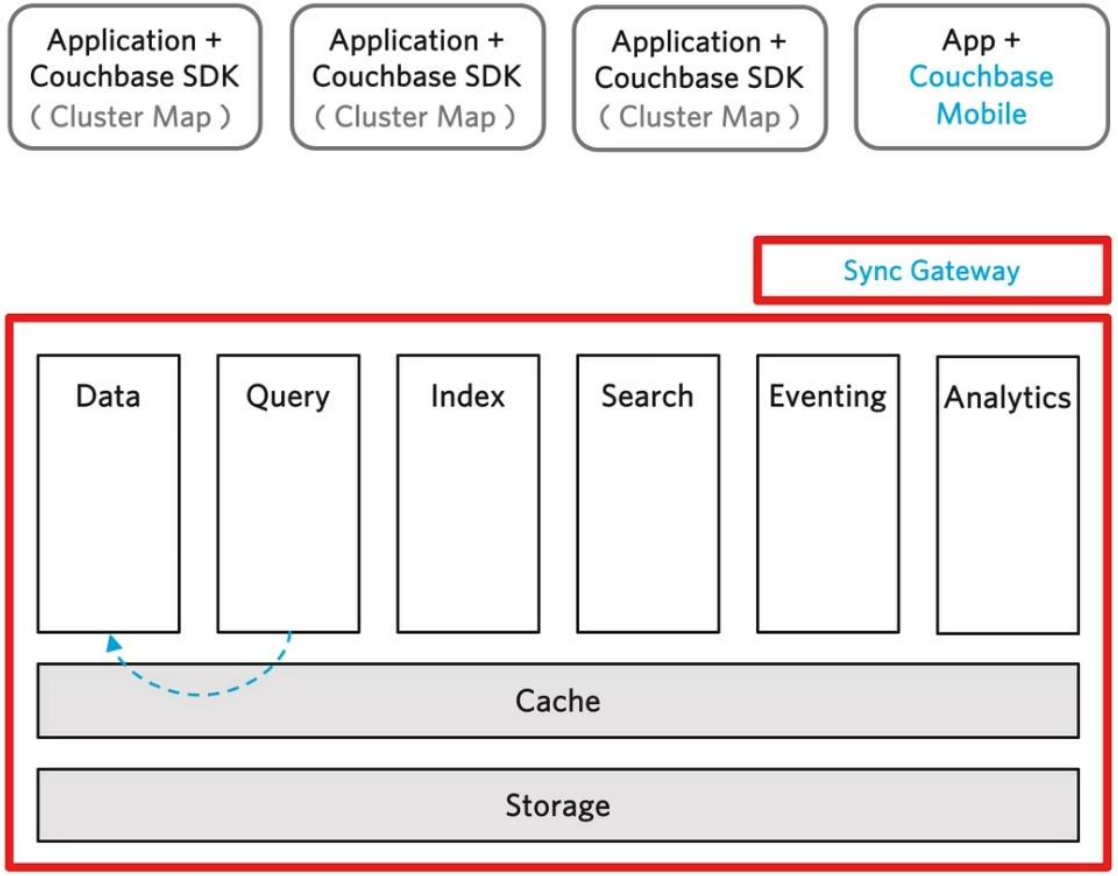
How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution
- ✓ Query Service sends request to Index Service
  Continuously maintains defined indexes
- ✓ Index Service returns Doc IDs & indexed data to Query Service
- ✓ If needed, Query Service fetches additional data from Data Service



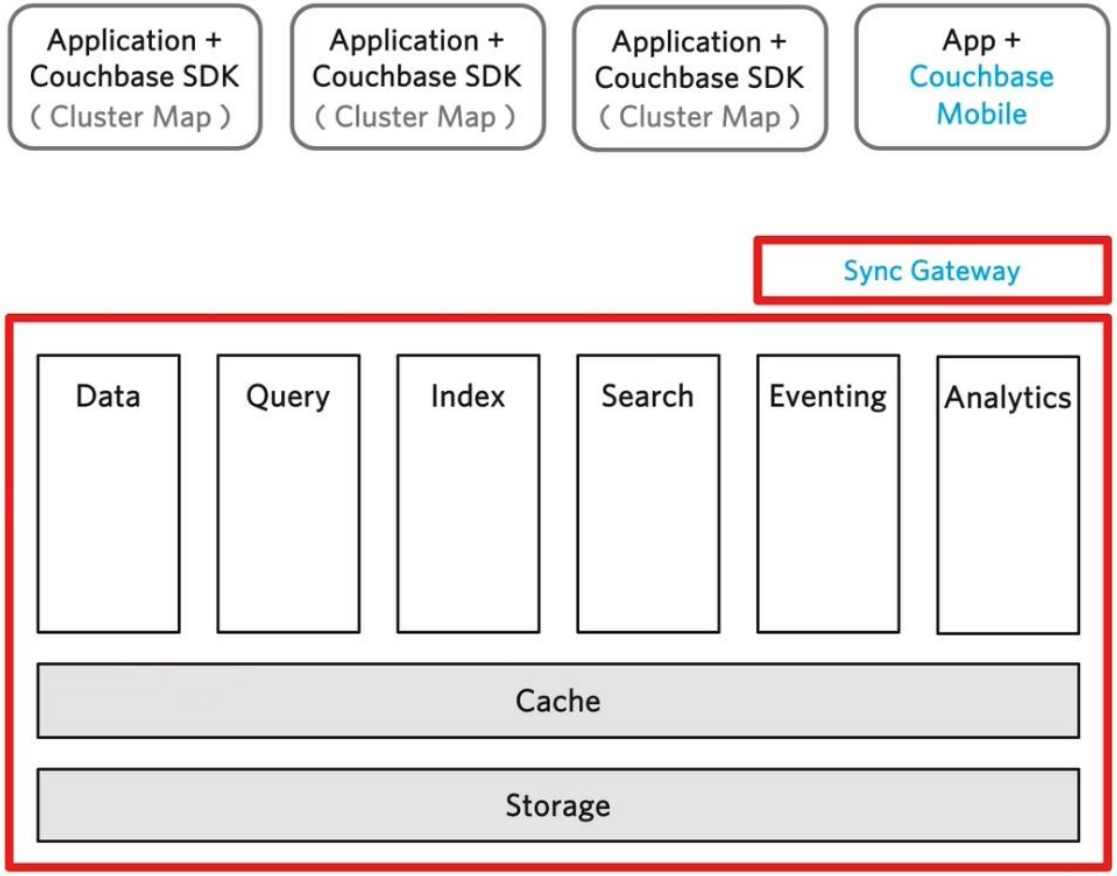
How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution
- ✓ Query Service sends request to Index Service
  Continuously maintains defined indexes
- ✓ Index Service returns Doc IDs & indexed data to Query Service
- ✓ If needed, Query Service fetches additional data from Data Service
  Skip if Index provides ("covers") all data



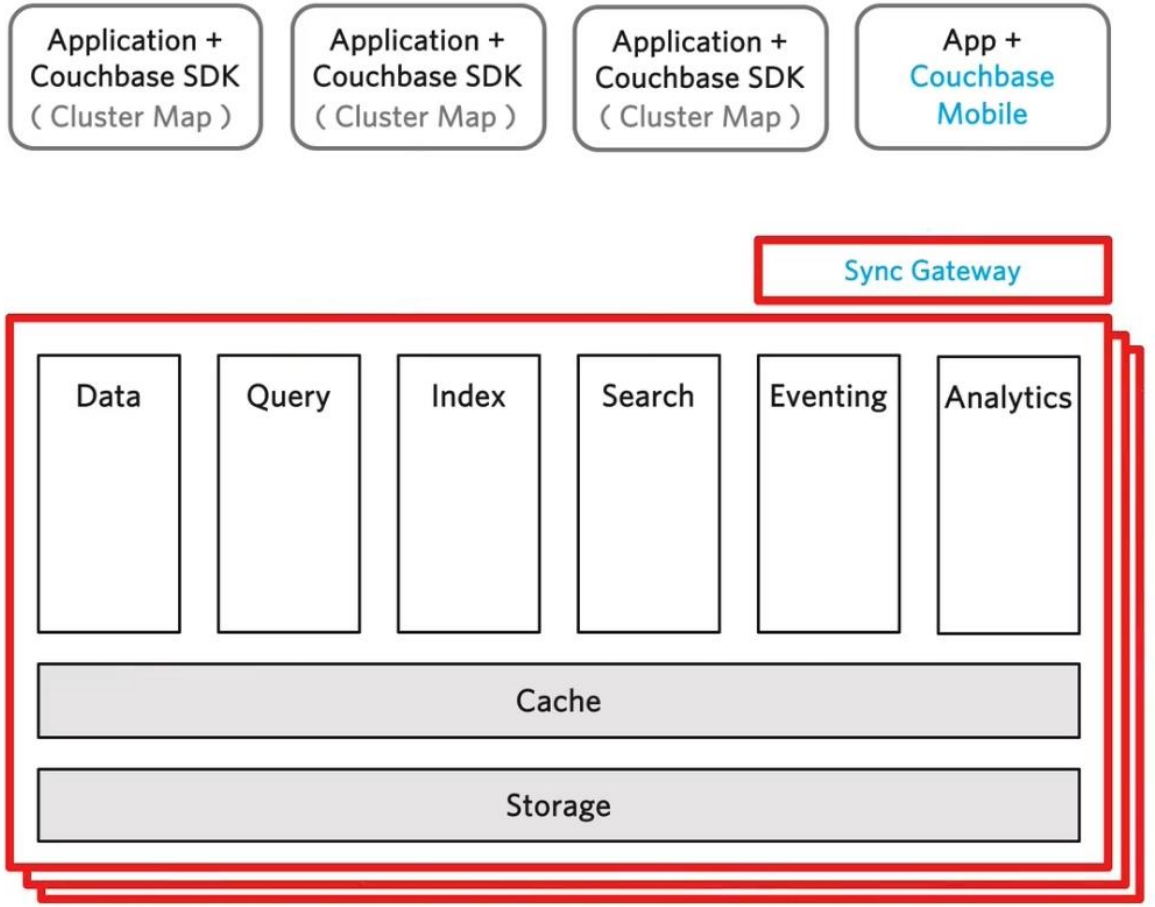
How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution
- ✓ Query Service sends request to Index Service
  Continuously maintains defined indexes
- ✓ Index Service returns Doc IDs & indexed data to Query Service
- ✓ If needed, Query Service fetches additional data from Data Service
  Skip if Index provides ("covers") all data
- ✓ Query Service formats results (`ORDER BY`, etc.)



How do N1QL queries flow through a node?

- ✓ N1QL sent to Query Service
  Parse, analyze & plan execution
- ✓ Query Service sends request to Index Service
  Continuously maintains defined indexes
- ✓ Index Service returns Doc IDs & indexed data to Query Service
- ✓ If needed, Query Service fetches additional data from Data Service
  Skip if Index provides ("covers") all data
- ✓ Query Service formats results (ORDER BY, etc.)
- ✓ Query Service returns results
- ✓ Nodes are involved across cluster



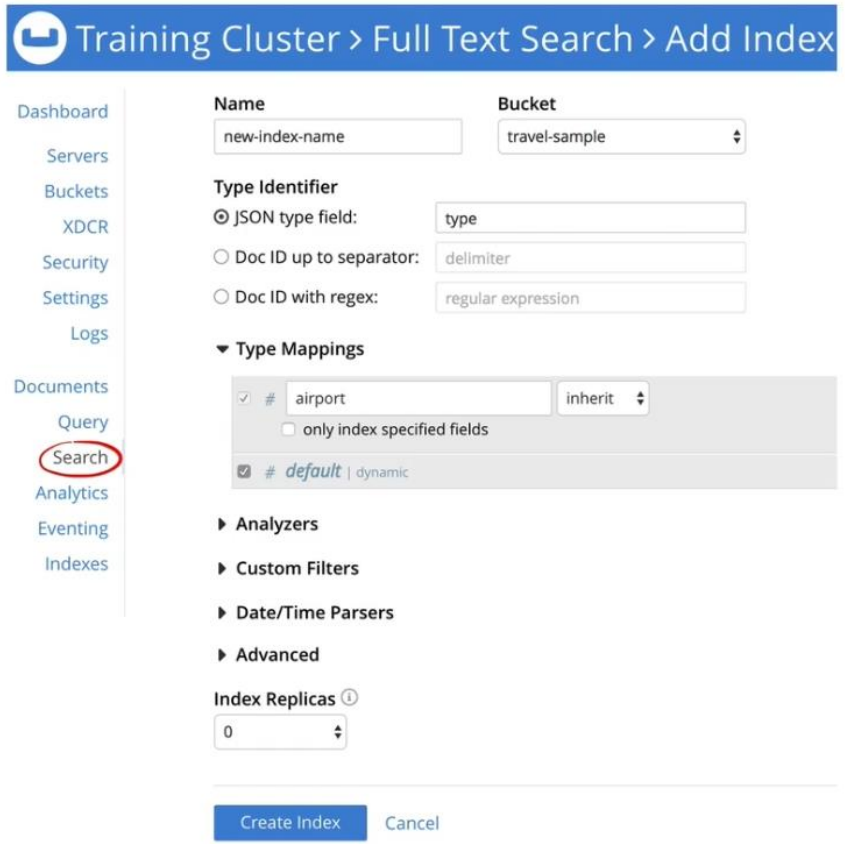
What does Couchbase Full Text Search offer?

Natural language query on JSON data

- ✓ Multi-lingual and custom text analysis for both searching and indexing
- ✓ Term scoring (tf-idf) and highlighting
- ✓ Faceting by term, or numeric/date range

Search by required, optional, or boosted

- ✓ Term, Phrase, Match, Match Phrase, or Prefix
- ✓ Common Stem
- ✓ Compound conjunction, disjunction, and boolean queries



INSERT

- Insert one or more new documents into an existing keyspace.
- Each INSERT statement requires a unique document key and well-formed JSON as values.
- In Couchbase, documents in a single bucket must have a unique key.
- The INSERT statement can compute and return any expression based on the actual inserted documents.
- Use the UPSERT statement if you want to overwrite a document with the same key, in case it already exists.

Prerequisites

INSERT statement must include the following:

- Name of the keyspace to insert the document.
- Unique document key.
- A well-formed JSON document specified as key-value pairs, or the projection of a SELECT statement which generates a well-formed single JSON to insert.
- Optionally, you can specify the values or an expression to be returned after the INSERT statement completes successfully.

Examples

```
INSERT INTO `travel-sample` (KEY, VALUE)
VALUES ("key1", { "type" : "hotel", "name" : "new hotel" })
```

```
INSERT INTO `travel-sample` (KEY, VALUE)
VALUES ("key1", { "type" : "hotel", "name" : "new hotel" }) RETURNING *
```

```
INSERT INTO `travel-sample` (KEY foo, VALUE bar)
SELECT foo, bar FROM `beer-sample`
```

```
INSERT INTO `travel-sample` (KEY foo, VALUE bar)`
SELECT "foo" || meta().id, bar FROM `travel-sample` WHERE type = "hotel"
```

Result

The INSERT statement returns the requestID, the signature, results including the keyspace and JSON document inserted, status of the query, and metrics.

- requestID: Request ID of the statement generated by the server.
- signature: Signature of the fields specified in the returning clause.
- results: If the query specified the returning clause, then results contains one or more fields as specified in the returning clause. If not, returns an empty results array.
- errors: Returns the error codes and messages if the statement fails with errors. Returned only when the statement fails with errors. Errors can also include timeouts.
- status: Status of the statement - "successful" or "errors".
- metrics: Provides metrics for the statement such as elapsedTime, executionTime, resultCount, resultSize, and mutationCount.

Metrics

The INSERT statement returns the following metrics along with the results and status:

- elapsedTime: Total elapsed time for the statement.
- executionTime: Time taken by Couchbase Server to execute the statement. This value is independent of network latency, platform code execution time, and so on.
- resultCount: Total number of results returned by the statement. In case of INSERT without a RETURNING clause, the value is 0.
- resultSize: Total number of results that satisfy the query.
- mutationCount: Specifies the number of documents that were inserted by the INSERT statement

Example 1. Specify a key using an expression

- Can specify a key using an expression
- Query

```
INSERT INTO `travel-sample` ( KEY, VALUE )
                VALUES ( "airline" || TOSTRING(1234),
                { "callsign": "" } )
                RETURNING META().id;
```

Generate a unique key

- If you don't require the document key to be in a specific format, can use the function UUID() to generate a unique key
- Since the document key is auto-generated, can find the value of the key by specifying META().id in the returning clause.

```
INSERT INTO `travel-sample` ( KEY, VALUE )
        VALUES ( UUID(),
               { "callsign": "" } )
RETURNING META().id;
```

Insert an empty value

```
INSERT INTO `travel-sample` (KEY, VALUE)
    VALUES ( "airline::432",
            { "callsign": "",
              "country" : "USA",
              "type" : "airline"} )
RETURNING META().id as docid;
```

Insert a NULL value

```
INSERT INTO `travel-sample` (KEY, VALUE)
    VALUES ( "airline::1432",
           { "callsign": NULL,
             "country" : "USA",
             "type" : "airline"} )
RETURNING *;
```

Insert a MISSING value

```
INSERT INTO `travel-sample` (KEY, VALUE)
    VALUES ( "airline::142",
           { "callsign": MISSING,
             "country" : "USA",
             "type" : "airline"} )
RETURNING *;
```

Insert a NULL JSON document

```
INSERT INTO `travel-sample` (KEY, VALUE)
    VALUES ( "1021",
            { } )
            RETURNING *;
```

Insert a document with expiration

```
INSERT INTO `travel-sample` (KEY, VALUE, OPTIONS)
    VALUES ( "airline::ttl",
           { "callsign": "Temporary",
             "country" : "USA",
             "type" : "airline" },
           { "expiration": 5*24*60*60 } );
```

Insert a document into the travel-sample bucket using an expiration of 5 days.

Insert with SELECT

- Query the travel-sample bucket for documents of type "airport" and airportname "Heathrow", and then insert the projection (1 document) into the travel-sample bucket using a unique key generated using UUID().

```
INSERT INTO `travel-sample` (KEY UUID(), VALUE _airport)
    SELECT _airport FROM `travel-sample` _airport
     WHERE type = "airport" AND airportname = "Heathrow"
RETURNING *;
```

Insert with SELECT and set expiration

- Query the travel-sample bucket for documents of type "airport" and airportname "Heathrow", and then insert the projection into the travel-sample bucket using a unique key and an expiration of 2 hours.

```
INSERT INTO `travel-sample` (KEY UUID(), VALUE doc, OPTIONS {"expiration": 2*60*60})
    SELECT a AS doc FROM `travel-sample` a
     WHERE type = "airport" AND airportname = "Heathrow";
```

Insert with SELECT and preserve expiration

- To copy the expiration of an existing document to the inserted document, you can use a META().expiration expression in the SELECT statement

```
INSERT INTO `travel-sample` (KEY UUID(), VALUE doc, OPTIONS {"expiration": ttl})
   SELECT META(a).expiration AS ttl, a AS doc FROM `travel-sample` a
    WHERE type = "airport" AND airportname = "Heathrow";
```

Return the document ID and country

```
INSERT INTO `travel-sample` (KEY, VALUE)
    VALUES ( "airline_24444",
            { "callsign": "USA-AIR",
              "country" : "USA",
              "type" : "airline"})
RETURNING META().id as docid, country;
```

Return the document ID and an expression

- Use the UUID() function to generate the key and show the usage of the RETURNING clause to retrieve the generated document key and the last element of the callsign array with an expression.

```
INSERT INTO `travel-sample` (KEY, VALUE)
    VALUES ( UUID(),
            { "callsign": [ "USA-AIR", "America-AIR" ],
              "country" : "USA",
              "type" : "airline"} )
RETURNING META().id as docid, callsign[ARRAY_LENGTH(callsign)-1];
```

Return Doc Id and Document

- inserts a single JSON document into the travel-sample bucket with key "k001". The returning clause specifies the function META().id to return the key of the inserted document (metadata), and the wildcard (*) to return the inserted document.

```
INSERT INTO `travel-sample` ( KEY, VALUE )
  VALUES
  (
    "k001",
    { "id": "01", "type": "airline"}
  )
RETURNING META().id as docid, *;
```

Inserting a Single Document

- Insert a new document with key "1025" and type "airline" into the travel-sample bucket.

```
INSERT INTO `travel-sample` (KEY,VALUE)
  VALUES ( "1025",
          {    "callsign": "MY-AIR",
               "country": "United States",
               "iata": "Z1",
               "icao": "AQZ",
               "id": "1011",
               "name": "80-My Air",
               "type": "airline"
          } )
RETURNING *;
```

Performing Bulk Inserts

```
INSERT INTO `travel-sample` (KEY,VALUE)
VALUES (
"airline_4444",  { "callsign": "MY-AIR",  "country": "United States",  "iata": "Z1",  "icao": "AQZ",  "name": "80-My Air",  "id": "4444",  "type": "airline"}
 ),
VALUES (
"airline_4445",  { "callsign": "AIR-X",  "country": "United States",  "iata": "X1",  "icao": "ARX",  "name": "10-AirX",  "id": "4445",  "type": "airline"}
)
RETURNING *;
```

Update - Set an attribute

- The following statement sets the nickname of the landmark "Tradeston Pedestrian Bridge" to "Squiggly Bridge".

```
UPDATE `travel-sample` USE KEYS "landmark_10090" SET nickname = "Squiggly Bridge"
RETURNING `travel-sample`.nickname;
```

Unset an attribute

- This statement removes the nickname attribute from the travel-sample keyspace for the document with the key landmark_10090.

```
UPDATE `travel-sample` USE KEYS "landmark_10090" UNSET nickname RETURNING `travel-sample`.name;
```

Set attributes in an array

- This statement sets the codeshare attribute for each element in the schedule array for document route_10003 in the travel-sample keyspace.

```
UPDATE `travel-sample` t USE KEYS "route_10003" SET s.codeshare = NULL FOR s IN schedule END RETURNING t;
```

Set nested array elements

Query:

```
UPDATE `travel-sample` AS h
USE KEYS "hotel_10025"

SET i.ratings =
OBJECT_ADD(i.ratings, "new",
"new_value" ) FOR i IN reviews
END

RETURNING
h.reviews[*].ratings;
```

Output:


```
[  {
    "ratings": [
     {
      "Cleanliness": 5,
      "Location": 4,
      "Overall": 4,
      "Rooms": 3,
      "Service": 5,
      "Value": 4,
      "new": "new_value"
     },
     {
      "Business service (e.g., internet access)": 4,
      "Check in / front desk": 4,
      "Cleanliness": 4,        "Location": 4,
      "Overall": 4,        "Rooms": 3,
      "Service": 3,        "Value": 5,
      "new": "new_value"
     }    ]  }]
```

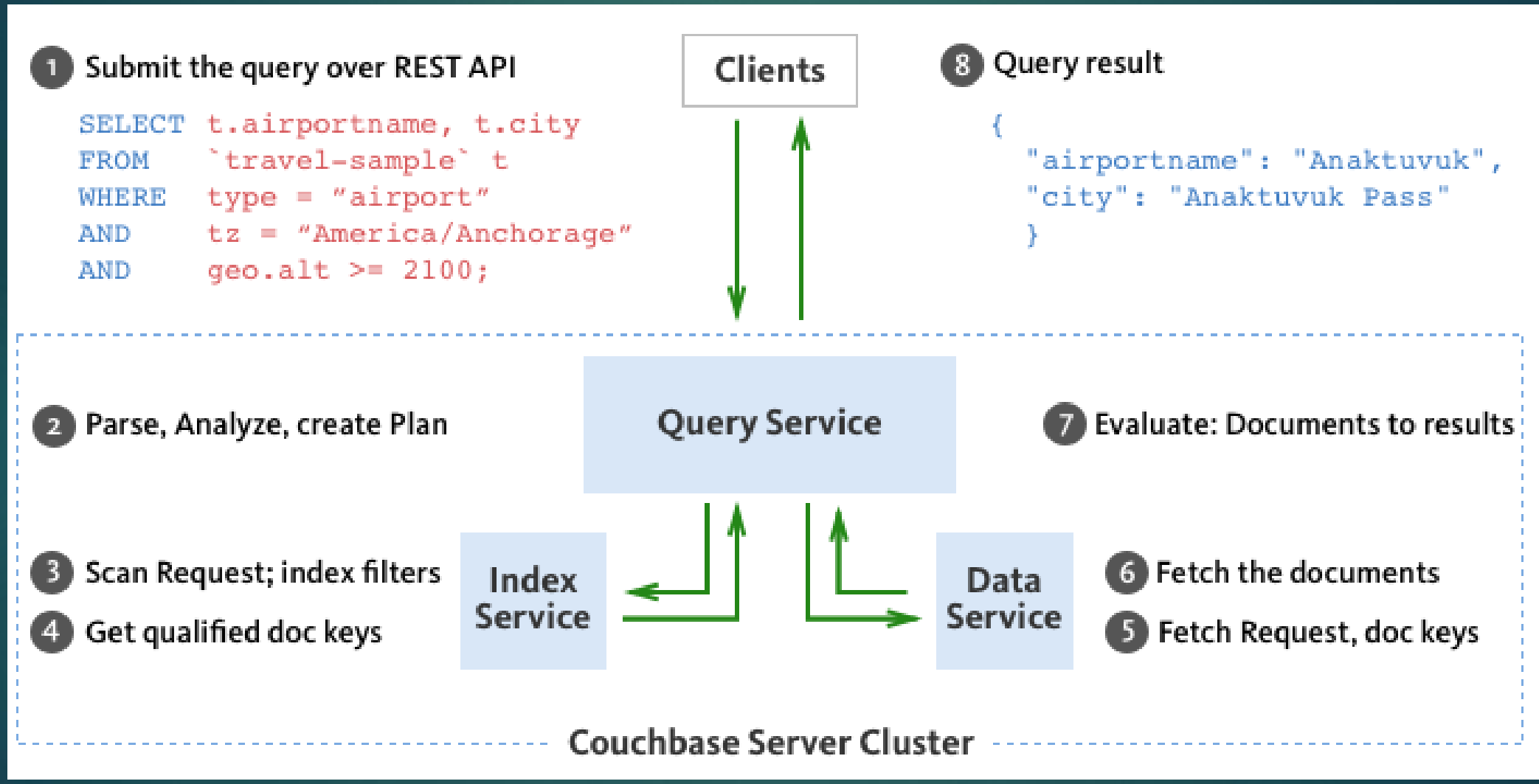
UPSERT

```
UPSERT INTO `travel-sample` (KEY, VALUE)
VALUES ("key1", { "type" : "hotel", "name" : "new hotel" })

UPSERT INTO `travel-sample` (KEY, VALUE)
VALUES ("key1", { "type" : "hotel", "name" : "new hotel" })
RETURNING *

UPSERT INTO `travel-sample` (KEY foo, VALUE bar)
SELECT foo, bar FROM `beer-sample`

UPSERT INTO `travel-sample` (KEY, VALUE)
VALUES ("upsert-1", { "name": "The Minster Inn", "type": "landmark-pub"}),
("upsert-2", {"name": "The Black Swan", "type": "landmark-pub"})
RETURNING VALUE name;
```

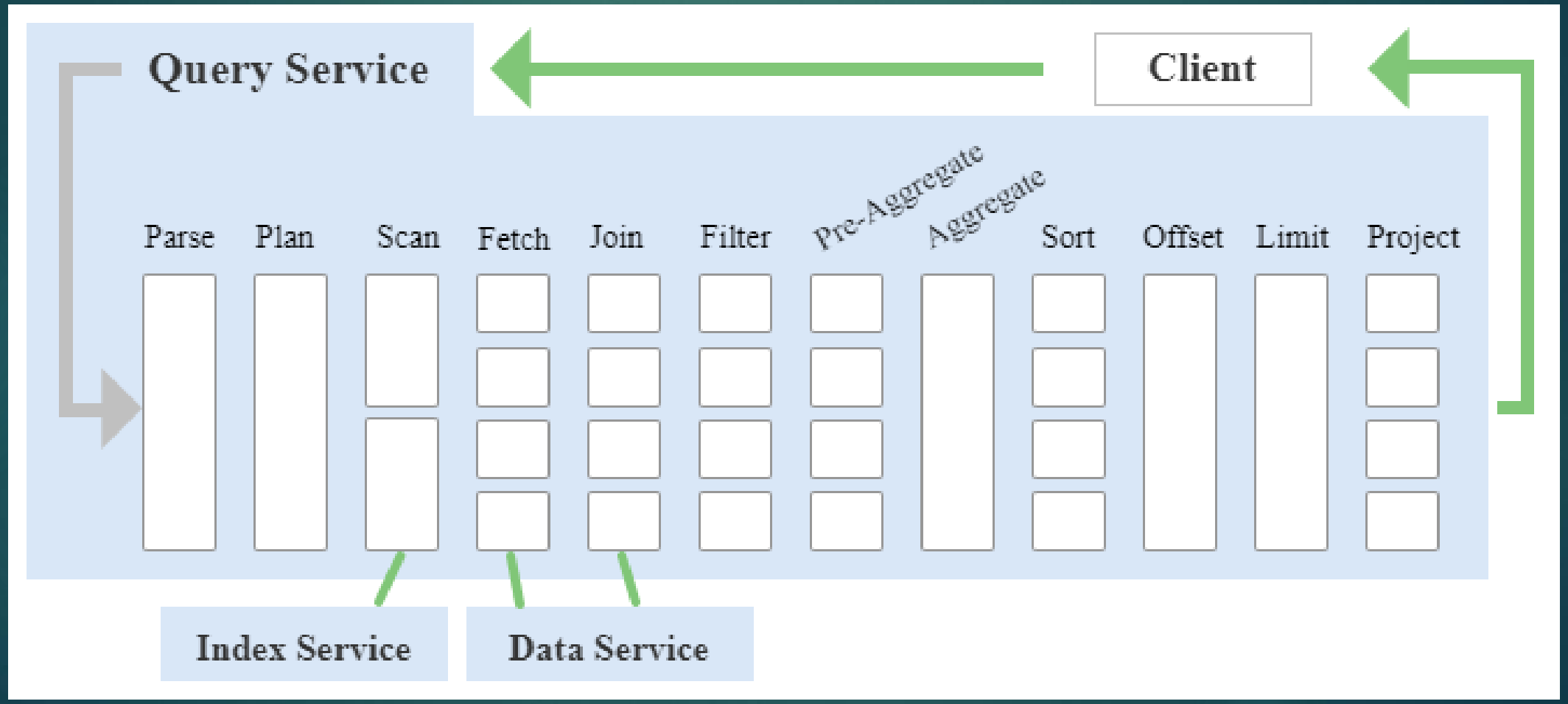
SELECT Clause

```
SELECT select_clause
[FROM from_clause] [JOIN join_clause]
[USE INDEX useindex_clause]
[LET let_clause]
[WHERE where_clause ( [AND where_clause2] )* ]
[GROUP BY groupby_clause] [LETTING | HAVING letting_clause]
[UNION | INTERSECT | EXCEPT union_clause]
[ORDER BY orderby_clause]
[LIMIT limit_int]
[OFFSET offset_clause]
;
```

SELECT Statement Processing



Query Execution Phases



Query Phases

Query Phase	Description
Parse	Analyzes the query and available access path options for each keyspace in the query to create a query plan and execution infrastructure.
Plan	Selects the access path, determines the Join order, determines the type of Joins, and then creates the infrastructure needed to execute the plan.
Scan	Scans the data from the Index Service.
Fetch	Fetches the data from the Data Service.
Join	Joins the data from the Data Service.
Filter	Filters the result objects by specifying conditions in the WHERE clause.
Pre-Aggregate	Internal set of tools to prepare the Aggregate phase.
Aggregate	Performs aggregating functions and window functions.
Sort	Orders and sorts items in the resultset in the order specified by the ORDER BY clause
Offset	Skips the first n items in the result object as specified by the OFFSET clause.
Limit	Limits the number of results returned using the LIMIT clause.
Project	Receives only the fields needed for final displaying to the user.

Elements and operations in a query

- Specifying the keyspace that is queried.
- Specifying the document keys or using indexes to access the documents.
- Fetching the data from the data service.
- Filtering the result objects by specifying conditions in the WHERE clause.
- Removing duplicate result objects from the resultset by using the DISTINCT clause.
- Grouping and aggregating the result objects.
- Ordering (sorting) items in the resultset in the order specified by the ORDER BY expression list.
- Skipping the first n items in the result object as specified by the OFFSET clause.
- Limiting the number of results returned using the LIMIT clause

```
SELECT city
FROM `travel-sample`
WHERE type="airport"
ORDER BY city LIMIT 5;
```

```
Results:
[
  {
    "city": "Abbeville"
  },
  {
    "city": "Aberdeen"
  },
  {
    "city": "Aberdeen"
  },
  {
    "city": "Aberdeen"
  },
  {
    "city": "Abilene"
  }
]
```

```
SELECT RAW city
FROM `travel-sample`
WHERE type="airport"
ORDER BY city LIMIT 5;
```

```
Results:
[
  "Abbeville",
  "Aberdeen",
  "Aberdeen",
  "Aberdeen",
  "Abilene"
]
```

```
SELECT DISTINCT RAW city
FROM `travel-sample`
WHERE type="airport"
ORDER BY city LIMIT 5;
```

```
Results:
[
  "Abbeville",
  "Aberdeen",
  "Abilene",
  "Adak Island",
  "Addison"
]
```

Order by Return values

- If no ORDER BY clause is specified, the order in which the result objects are returned is undefined.

- Objects are sorted first by the left-most expression in the list of expressions.
- Any items with the same sort value will be sorted with the next expression in the list.
- This process repeats until all items are sorted and all expressions in the list are evaluated.

- When a field has a mix of data types, the different JSON types are sorted in the following order, from first to last:

Order by Return values

ASC NULLS FIRST	ASC NULLS LAST	DESC NULLS FIRST	DESC NULLS LAST
MISSING	FALSE	NULL	BINARY
NULL	TRUE	MISSING	OBJECT
FALSE	NUMBER	BINARY	ARRAY
TRUE	STRING	OBJECT	STRING
NUMBER	ARRAY	ARRAY	NUMBER
STRING	OBJECT	STRING	TRUE
ARRAY	BINARY	NUMBER	FALSE
OBJECT	MISSING	TRUE	NULL
BINARY	NULL	FALSE	MISSING

Query to find the names of (at a maximum) ten hotels that accept pets, in the city of Medway:.

```
SELECT name FROM `travel-sample` WHERE type="hotel" AND city="Medway" and pets_ok=true LIMIT 10;
```

Query the keyspace for airports that are in the America/Anchorage timezone and at an altitude of 2100ft or higher, and returns an array with the airport name and city name for each airport that satisfies the conditions.

```
SELECT t.airportname, t.city
FROM   `travel-sample` t
WHERE  type = "airport"
       AND tz = "America/Anchorage"
       AND geo.alt >= 2100;
```

Query the keyspace using the document key

```
SELECT * FROM `travel-sample` USE KEYS "airport_3469";
```

Query to Select multiple documents by their document keys

```
SELECT *
FROM `travel-sample`
USE KEYS ["airport_1254","airport_1255"];
```

Query to Create an index of airlines and destination airports, and then use it in a query for flights originating in San Francisco.

```
CREATE INDEX idx_destinations
ON `travel-sample` (airlineid, airline, destinationairport)
WHERE type="route";
```

```
SELECT airlineid, airline, sourceairport, destinationairport
FROM `travel-sample` USE INDEX (idx_destinations USING GSI)
WHERE sourceairport = "SFO";
```

Query to list airports in France

```
SELECT airportname, city, country
FROM `travel-sample`
WHERE type = "airport"
AND country = "France"
LIMIT 4;
```

Query to list only the landmarks that start with the letter "C" or "K".

```
SELECT name
FROM `travel-sample`
WHERE type = "landmark"
AND ( CONTAINS(SUBSTR(name,0,1),"C")
   OR CONTAINS(SUBSTR(name,0,1),"K") )
LIMIT 4;
```

Query to List landmark restaurants, except Thai restaurants..

```
SELECT name, activity
FROM `travel-sample`
WHERE type = "landmark"
AND activity = "eat"
AND NOT CONTAINS(name,"Thai")
LIMIT 4;
```

Query to List cities in descending order and then landmarks in ascending order.

```
SELECT city, name
FROM `travel-sample`
WHERE type = "landmark"
ORDER BY city DESC, name ASC;
```

Query to List the names in ascending order of hotels and landmarks resulting from a UNION query.

```
SELECT name
  FROM `travel-sample`
  WHERE type = "landmark"
UNION SELECT name
  FROM `travel-sample`
  WHERE type = "hotel"
ORDER BY name ASC;
```

Query to group the unique landmarks by city and list the top 4 cities with the most landmarks in descending order.

```
SELECT city City, COUNT(DISTINCT name) LandmarkCount
FROM `travel-sample`
WHERE type = "landmark"
GROUP BY city
ORDER BY LandmarkCount DESC
LIMIT 4;
```

Query to Use LETTING to find cities that have a minimum number of things to see

```
SELECT city City, COUNT(DISTINCT name) LandmarkCount
FROM `travel-sample`
WHERE type = "landmark"
GROUP BY city
LETTING MinimumThingsToSee = 400
HAVING COUNT(DISTINCT name) > MinimumThingsToSee;
```

Query to find cities that have more than 180 landmarks

```
SELECT city City, COUNT(DISTINCT name) LandmarkCount
FROM `travel-sample`
WHERE type = "landmark"
GROUP BY city
HAVING COUNT(DISTINCT name) > 180;
```

Query to find landmarks that begin with an "S" or higher

```
SELECT city City, COUNT(DISTINCT name) LandmarkCount
FROM `travel-sample`
WHERE type = "landmark"
GROUP BY city
HAVING city > "S";
```

Query with case in group.

```
SELECT Hemisphere, COUNT(DISTINCT name) AS LandmarkCount
FROM `travel-sample` AS l
WHERE type="landmark"
GROUP BY CASE
  WHEN l.geo.lon <0 THEN "West"
  ELSE "East"
END AS Hemisphere;
```

Query to look for the schedule, and access the first flight id for destinationairport=ALG..

```
SELECT t.schedule[0].flight AS flightid
FROM `travel-sample` t
WHERE type="route"
  AND destinationairport="ALG"
LIMIT 1;
```

Query to find the average number of public likes for each record. Then find all hotels with a greater than average number of public likes..

```
WITH avgLikeCount AS (
  SELECT VALUE AVG(DISTINCT ARRAY_COUNT(cte.public_likes))
  FROM `travel-sample` AS cte
)
SELECT hotel.name, ARRAY_COUNT(hotel.public_likes) AS likeCount
FROM `travel-sample` AS hotel
WHERE ARRAY_COUNT(hotel.public_likes) > avgLikeCount[0]
LIMIT 5;
```

Create a recordset of hotel names and their Cleanliness ratings. Then use this recordset to find the names all hotels whose average Cleanliness rating is greater than 4.5.

```
WITH hotels AS (
  SELECT name, reviews[*].ratings[*].Cleanliness
  FROM `travel-sample`
  WHERE type = "hotel"
)
SELECT hotels.name
FROM hotels
WHERE ARRAY_AVG(hotels.Cleanliness) > 4.5
LIMIT 5;
```

Query to list all landmark names from a subset of all landmark names and addresses..

```
SELECT name, city
FROM (SELECT id, name, address, city
    FROM `travel-sample`
    WHERE type = "landmark") as Landmark_Info
WHERE city = "Gillingham";
```

Query to find the name and phone fields for up to 10 documents for hotels in Manchester, where directions are not missing, and orders the results by name:.

```
SELECT name,phone FROM `travel-sample` WHERE type="hotel" AND city="Manchester" and directions IS NOT MISSING ORDER BY name LIMIT 10;
```

Example of let.

```
SELECT count(*) FROM `travel-sample` t
LET x = t.geo
WHERE (SELECT RAW y.alt FROM x y)[0] > 6000;
```

Query to find all the cities with landmarks that have airports.

```
SELECT t1.city
FROM `travel-sample` t1
WHERE t1.type = "landmark" AND
    t1.city IN (SELECT RAW city
                FROM `travel-sample`
                WHERE type = "airport");
```

Query to to find total number of airports by country where each city has more than 5 airports

```
SELECT t1.country, array_agg(t1.city), sum(t1.city_cnt) as apnum
FROM (SELECT city, city_cnt, array_agg(airportname) as apnames, country
    FROM `travel-sample` WHERE type = "airport"
    GROUP BY city, country LETTING city_cnt = count(city) ) AS t1
WHERE t1.city_cnt > 5
GROUP BY t1.country;
```

Query to find the landmark that are named with corresponding city name.

```
SELECT t1.city, t1.name
FROM `travel-sample` t1
WHERE t1.type = "landmark" AND
      t1.city IN SPLIT((SELECT RAW t2.name
                        FROM t1 AS t2)[0]);
```

```
SELECT t1.city, t1.name
FROM `travel-sample` t1
WHERE t1.type = "landmark" AND
      t1.city IN SPLIT(t1.name);
```

```
SELECT count(*) FROM `travel-sample` t
WHERE (SELECT RAW t.geo.alt FROM t t1)[0] > 6000 ;
```

```
SELECT count(*) FROM `travel-sample` t
WHERE (SELECT RAW alt FROM t.geo.alt)[0] > 6000;
```

```
SELECT count(*) FROM `travel-sample` t
LET x = t.geo
WHERE (SELECT RAW y.alt FROM x y)[0] > 6000;
```

```
SELECT count(*) FROM `travel-sample` t
WHERE (SELECT RAW geo.alt FROM t.geo)[0] > 6000;
```

Query to find airports that are at altitudes more than 4000ft

```
SELECT t1.city, t1.geo.alt
FROM `travel-sample` t1
WHERE t1.type = "airport" AND
    (SELECT RAW t2.alt FROM t1.geo t2)[0] > 4000;
```

JOINS

- JOIN clause is used within the FROM clause.
- Creates an input object by combining two or more source objects.
- Couchbase Server supports three types of JOIN clause: ANSI JOIN, Lookup JOIN, and Index JOIN.

ANSI JOIN Clause

- To be closer to standard SQL syntax, ANSI JOIN can join arbitrary fields of the documents and can be chained together.
- ANSI JOIN and ANSI NEST clauses have much more flexible functionality than their earlier INDEX and LOOKUP equivalents.
- Since these are standard compliant and more flexible, it is recommended to use ANSI JOIN and ANSI NEST exclusively, where possible.

Type of joins

- INNER
  - For each joined object produced, both the left-hand side and right-hand side source objects of the ON clause must be non-MISSING and non-NULL.
- LEFT [OUTER]
  - [Query Service interprets LEFT as LEFT OUTER]
  - For each joined object produced, only the left-hand source objects of the ON clause must be non-MISSING and non-NULL.
- RIGHT [OUTER]
  - [Query Service interprets RIGHT as RIGHT OUTER]
  - For each joined object produced, only the right-hand source objects of the ON clause must be non-MISSING and non-NULL.

[INNER] JOIN … ON

```
SELECT *
FROM `travel-sample` r
JOIN `travel-sample` a
ON r.airlineid = META(a).id
WHERE a.country = "France"
```

LEFT [OUTER] JOIN ... ON

```
SELECT *
FROM `travel-sample` r
LEFT JOIN `travel-sample` a
ON r.airlineid = META(a).id
WHERE r.sourceairport = "SFO"
```

RIGHT [OUTER] JOIN … ON

```
SELECT *
FROM `travel-sample` r
RIGHT JOIN `travel-sample` a
ON r.airlineid = META(a).id
WHERE r.sourceairport = "SFO"
```

Important

In Couchbase Server 6.5 and later, if you create either of the following:

- A LEFT OUTER JOIN where all the NULL or MISSING results on the right-hand side are filtered out by the WHERE clause or by the ON clause of a subsequent INNER JOIN, or

- A RIGHT OUTER JOIN where all the NULL or MISSING results on the left-hand side are filtered out by the WHERE clause or by the ON clause of a subsequent INNER JOIN,
- Then the query is transformed internally into an INNER JOIN for greater efficiency.

Query to List the source airports and airlines that fly into SFO, where only the non-null documents join with matching documents.

```
SELECT route.airlineid, airline.name, route.sourceairport, route.destinationairport
FROM `travel-sample` route
INNER JOIN `travel-sample` airline
ON route.airlineid = META(airline).id
WHERE route.type = "route"
AND route.destinationairport = "SFO"
ORDER BY route.sourceairport;
```

Query to List the airports and landmarks in the same city, ordered by the airports.

```
SELECT DISTINCT  MIN(aport.airportname) AS Airport__Name,
                 MIN(lmark.name) AS Landmark_Name,
                 MIN(aport.tz) AS Landmark_Time
FROM `travel-sample` aport
LEFT JOIN `travel-sample` lmark
  ON aport.city = lmark.city
  AND lmark.country = "United States"
  AND lmark.type = "landmark"
WHERE aport.type = "airport"
GROUP BY lmark.name
ORDER BY lmark.name;
```

Query to List the airports and landmarks in the same city, ordered by the landmarks.

```
SELECT DISTINCT  MIN(aport.airportname) AS Airport_Name,
                 MIN(lmark.name) AS Landmark_Name,
                 MIN(aport.tz) AS Landmark_Time
FROM `travel-sample` aport
RIGHT JOIN `travel-sample` lmark
  ON aport.city = lmark.city
  AND aport.type = "airport"
  AND aport.country = "United States"
WHERE lmark.type = "landmark"
GROUP BY lmark.name
ORDER BY lmark.name;
```

Lookup JOIN Clause

- Lookup joins allow only left-to-right joins, which means the ON KEYS expression must produce a document key which is then used to retrieve documents from the right-hand side keyspace.
- Couchbase Server version 4.1 and earlier supported only lookup joins.
- Join Type
  - Inner
    - For each joined object produced, both the left-hand and right-hand source objects must be non-MISSING and non-NULL.
  - LEFT [OUTER]
    - [Query Service interprets LEFT as LEFT OUTER]
    - For each joined object produced, only the left-hand source objects must be non-MISSING and non-NULL.
  - This clause is optional. If omitted, the default is INNER.

Query to list the schedule of flights from Boston to San Francisco on JETBLUE in the keyspace.

```
SELECT DISTINCT airline.name, route.schedule
FROM `travel-sample` route
  JOIN `travel-sample` airline
  ON KEYS route.airlineid
WHERE route.type = "route"
AND airline.type = "airline"
AND route.sourceairport = "BOS"
AND route.destinationairport = "SFO"
AND airline.callsign = "JETBLUE";
```

For each country, find the number of airports at different altitudes and their corresponding cities..

```
SELECT t1.country, num_alts, total_cities
FROM (SELECT country, geo.alt AS alt,
        count(city) AS num_cities
    FROM `travel-sample`
    WHERE type = "airport"
    GROUP BY country, geo.alt) t1
GROUP BY t1.country
LETTING num_alts = count(t1.alt), total_cities = sum(t1.num_cities);
```

Query to find the distinct airline details which have routes that start from SFO.

```
SELECT DISTINCT airline.name, airline.callsign, route.destinationairport, route.stops, route.airline
FROM `travel-sample` route
  JOIN `travel-sample` airline
  ON KEYS route.airlineid
WHERE route.type = "route"
  AND airline.type = "airline"
  AND route.sourceairport = "SFO"
LIMIT 2;
```

Query to get a list of the flights on Monday

```
SELECT ARRAY item FOR item IN schedule WHEN item.day = 1 END AS Monday_flights
FROM `travel-sample`
WHERE type = "route"
AND ANY item IN schedule SATISFIES item.day = 1 END
LIMIT 3;
```

Query to find all the cities with landmarks that have airports.

“UNNEST”-ing Nested Structures

- Breaking an array into individual elements

```
"1" : {
  "order_id": "1",
  "type": "order",
  "customer_id": "24601",
  "total_price": 30.3,
  "lineitems": [
    { "item_id": 576, "quantity": 3, "item_price": 4.99, "base_price": 14.97, "tax": 0.75,
       "final_price": 15.72 },
    { "item_id": 234, "quantity": 1, "item_price": 12.95, "base_price": 12.95, "tax": 0.65,
       "final_price": 13.6 },
    { "item_id": 122, "quantity": 2, "item_price": 0.49, "base_price": 0.98, "final_price": 0.98 }
  ]          }
"5" : {
  "order_id": "5",
  "type": "order",
  "customer_id": "98732",
  "total_price": 428.04,
  "lineitems": [
    { "item_id": 770, "quantity": 3, "item_price": 95.97, "base_price": 287.91, "tax": 14.4,
       "final_price": 302.31 },
    { "item_id": 712, "quantity": 1, "item_price": 125.73, "base_price": 125.73,
       "final_price": 125.73 }
  ]    }
```

SELECT * FROM demo UNNEST lineitems


```
[....,
  {
     {
    "demo": {
      "customer_id": "24601",
      "lineitems": [
        { "base_price": 14.97, "final_price": 15.72, "item_id": 576, "item_price": 4.99, "quantity": 3,
            "tax": 0.75        },
        { "base_price": 12.95, "final_price": 13.6, "item_id": 234, "item_price": 12.95, "quantity": 1,
            "tax": 0.65        },
        { "base_price": 0.98, "final_price": 0.98, "item_id": 122, "item_price": 0.49, "quantity": 2 }      ],
      "order_id": "1",            "total_price": 30.3,            "type": "order"
    },
    "lineitems": { "base_price": 12.95, "final_price": 13.6, "item_id": 234, "item_price": 12.95,
       "quantity": 1, "tax": 0.65 }
  },....
]
```

To find the tax payable for each order

```
SELECT demo.order_id, SUM(lineitems.tax) AS total_tax
FROM demo UNNEST lineitems
GROUP BY demo.order_id
```

Output:


```
[
  {
    "order_id": "1",
    "total_tax": 1.4
  },
  {
    "order_id": "5",
    "total_tax": 14.4
  }
]
```

“NEST”-ing an Unnested Structure

- What if the data starts out with orders and line items as separate entries in the database (unnested), and we want to group them together with line items under the documents. For example, we may want for each order, the items included and the quantity of each item.

"1" : { "order_id": "1", "type": "order", "customer_id": "24601", "total_price": 30.3,
"lineitems": ["11", "12", "13"] }

"11" : { "lineitem_id": "11", "type": "lineitem", "item_id": 576, "quantity": 3, "item_price": 4.99,
"base_price": 14.97, "tax": 0.75, "final_price": 15.72 }

"12" : { "lineitem_id": "12", "type": "lineitem", "item_id": 234, "quantity": 1, "item_price": 12.95,
"base_price": 12.95, "tax": 0.65, "final_price": 13.6 }

"13" : { "lineitem_id": "13", "type": "lineitem", "item_id": 122, "quantity": 2, "item_price": 0.49,
"base_price": 0.98, "final_price": 0.98 }

"5" : { "order_id": "5", "type": "order", "customer_id": "98732", "total_price": 428.04,
"lineitems" : ["51", "52"] }

"51" : { "lineitem_id": "51", "type": "lineitem", "item_id": 770, "quantity": 2, "item_price": 95.97,
"base_price": 287.91, "tax": 14.4, "final_price": 302.31 }

"52" : { "lineitem_id": "52", "type": "lineitem", "item_id": 712, "quantity": 1, "item_price": 125.73,
"base_price": 125.73, "final_price": 125.73 }

SELECT * FROM demo ordr NEST demo li ON KEYS ordr.lineitems


```
[
  {
    "li": [
      { "base_price": 14.97, "final_price": 15.72, "item_id": 576, "item_price": 4.99,
        "lineitem_id": "11", "quantity": 3, "tax": 0.75, "type": "lineitem" },
      { "base_price": 0.98, "final_price": 0.98, "item_id": 122, "item_price": 0.49, "lineitem_id": "13",
        "quantity": 2, "type": "lineitem" },
      { "base_price": 12.95, "final_price": 13.6, "item_id": 234, "item_price": 12.95,
        "lineitem_id": "12", "quantity": 1, "tax": 0.65, "type": "lineitem" }
    ],
    "ordr": {
      "customer_id": "24601",
      "lineitems": [
        "11",
        "12",
        "13"
      ],
      "order_id": "1",
      "total_price": 30.3,
      "type": "order"
    }
  },....
```

```
SELECT ordr.order_id, ARRAY {"item_id": l.item_id, "quantity" : l.quantity} FOR l
IN li END as items
FROM demo ordr NEST demo li ON KEYS ordr.lineitems
```


```
[
  {
    "items": [
      { "item_id": 576, "quantity": 3 },
      { "item_id": 234, "quantity": 1 },
      { "item_id": 122, "quantity": 2 }
    ],
    "order_id": "1"
  },
  {
    "items": [
      { "item_id": 712, "quantity": 1 },
      { "item_id": 770, "quantity": 2 }
    ],
    "order_id": "5"
  }
]
```

- In the travel-sample keyspace travel-sample, flatten the schedule array to get a list of the flights on Monday

```
SELECT sched
FROM `travel-sample`
UNNEST schedule sched
WHERE  sched.day = 1
LIMIT 3;
```

UNNEST

- Use UNNEST to collect items from one array to use in another query
- In this example, the UNNEST clause iterates over the reviews array and collects the author names of the reviewers who rated the rooms less than a 2 to be contacted for ways to improve. r is an element of the array generated by the UNNEST operation

```
SELECT RAW r.author
FROM `travel-sample`
UNNEST reviews AS r
WHERE `travel-sample`.type = "hotel"
AND r.ratings.Rooms < 2
LIMIT 4;
```