

BRINGING SQL TO NOSQL – RICH, DECLARATIVE

Isha Kandaswamy
Couchbase R&D



- MS: Johns Hopkins University
- Worked previously at Teradata
- Have been a software engineer at Couchbase for 3.5 years
- Worked on multiple features: CBQ, CURL(), Date time functions for N1QL
-



An abstract graphic in the top left corner showing a splash of liquid in vibrant blue, red, and yellow colors against a black background.

AGENDA

Overview

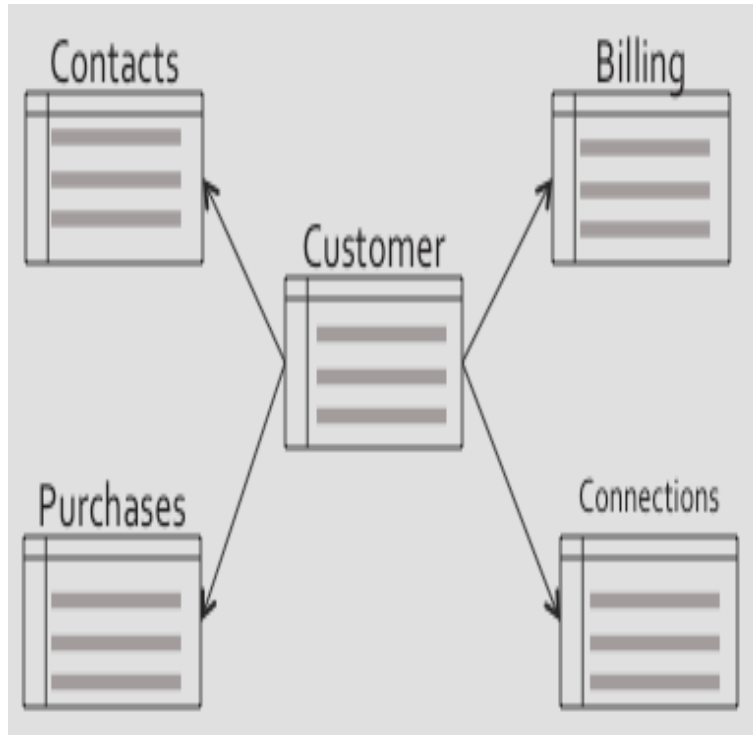
- Architecture
- Database Objects
- Data Types

N1QL In Depth

- Statements
- Indexes
- JOIN
- NEST
- UNNEST
- Security
- Tooling



Relations/Tuples



ResultSet



Give developers and enterprises an
expressive, powerful, and complete language
for querying, transforming, and manipulating
JSON data.

ResultDocuments

```
1 { "Name" : "Jane Smith",
2   "DOB" : "1990-01-30",
3   "Billing" : [
4     { "type" : "visa",
5       "cardnum" : "5827-2842-2847-3909",
6       "expiry" : "2019-03"
7     },
8     { "type" : "master",
9       "cardnum" : "6274-2842-2847-3909",
10      "expiry" : "2019-03"
11     }
12   ],
13   "Connections" : [
14     { "CustId" : "XYZ987",
15       "Name" : "Joe Smith"
16     },
17     { "CustId" : "PQR823",
18       "Name" : "Dylan Smith"
19     },
20     { "CustId" : "PQR823",
21       "Name" : "Dylan Smith"
22     }
23   ],
24   "Purchases" : [
25     { "id" : 12, "item" : "mac", "amt" : 2823.52 },
26     { "id" : 19, "item" : "ipad2", "amt" : 623.52 }
27   ]
28 }
```

LoyaltyInfo

CUSTOMER

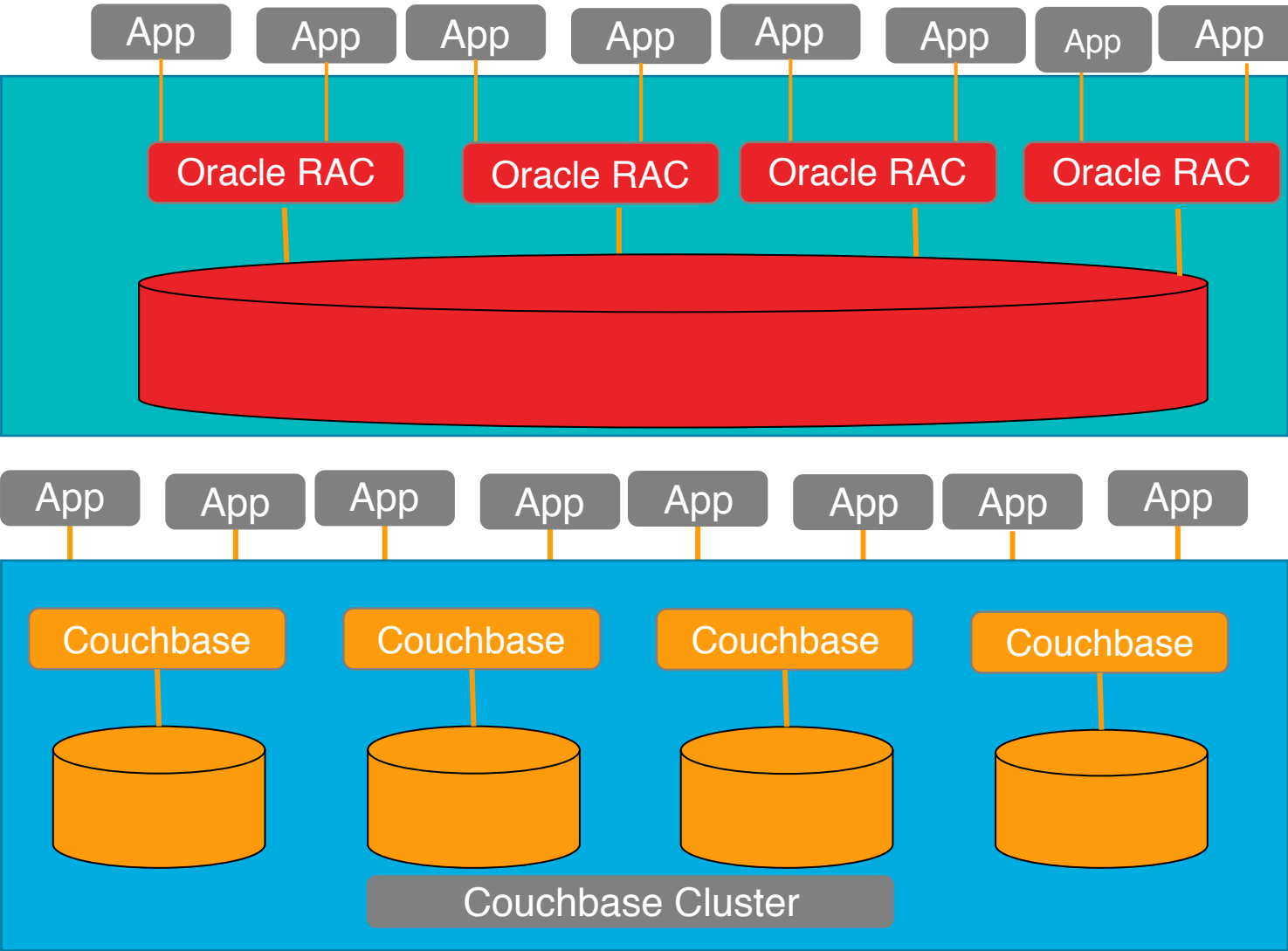
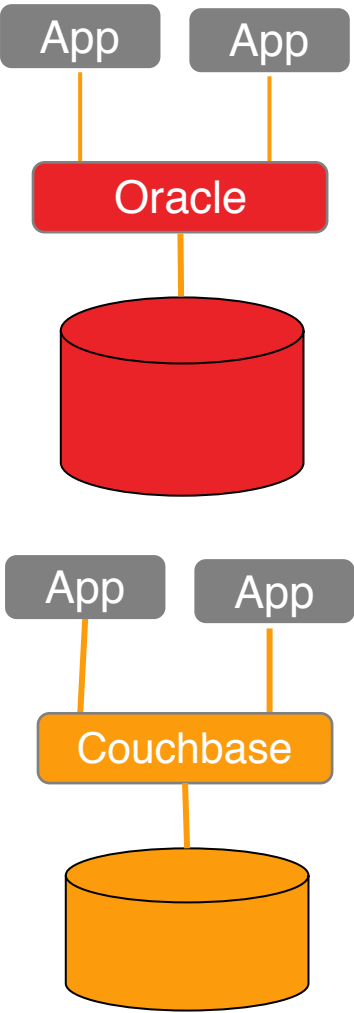
Orders

```
1 { "Name" : "Jane Smith",
2   "DOB" : "1990-01-30",
3   "Billing" : [
4     { "type" : "visa",
5       "cardnum" : "5827-2842-2847-3909",
6       "expiry" : "2019-03"
7     },
8     { "type" : "master",
9       "cardnum" : "6274-2842-2847-3909",
10      "expiry" : "2019-03"
11     }
12   ],
13   "Connections" : [
14     { "CustId" : "XYZ987",
15       "Name" : "Joe Smith"
16     },
17     { "CustId" : "PQR823",
18       "Name" : "Dylan Smith"
19     },
20     { "CustId" : "PQR823",
21       "Name" : "Dylan Smith"
22     }
23   ],
24   "Purchases" : [
25     { "id" : 12, "item" : "mac", "amt" : 2823.52 },
26     { "id" : 19, "item" : "ipad2", "amt" : 623.52 }
27   ]
28 }
```

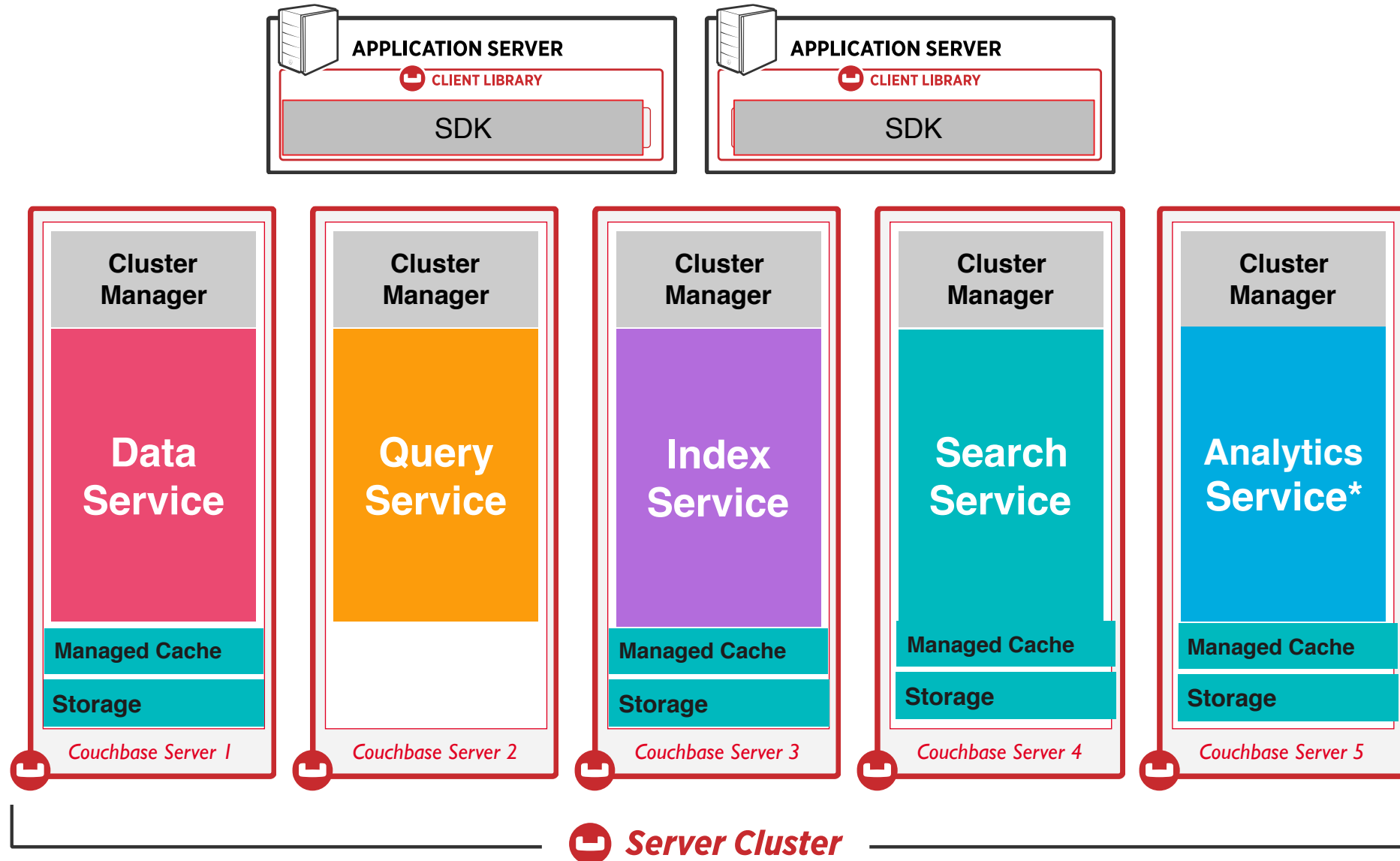
{N1QL}

```
{
  "Name" : "Jane Smith",
  "DOB" : "1990-01-30",
  "Billing" : [
    {
      "type" : "visa",
      "cardnum" : "5827-2842-2847-3909",
      "expiry" : "2019-03"
    },
    {
      "type" : "master",
      "cardnum" : "6274-2842-2847-3909",
      "expiry" : "2019-03"
    }
  ],
  "Connections" : [
    {
      "CustId" : "XYZ987",
      "Name" : "Joe Smith"
    },
    {
      "CustId" : "PQR823",
      "Name" : "Dylan Smith"
    },
    {
      "CustId" : "PQR823",
      "Name" : "Dylan Smith"
    }
  ],
  "Purchases" : [
    { "id":12, "item": "mac", "amt": 2823.52 },
    { "id":19, "item": "ipad2", "amt": 623.52 }
  ]
}
```

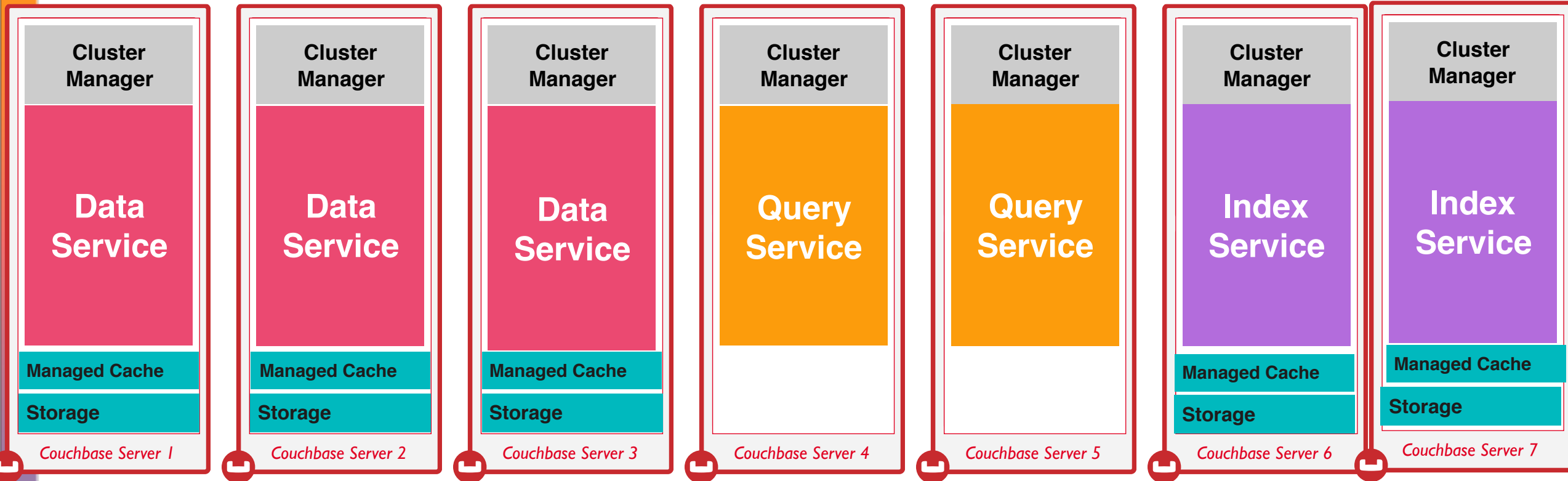
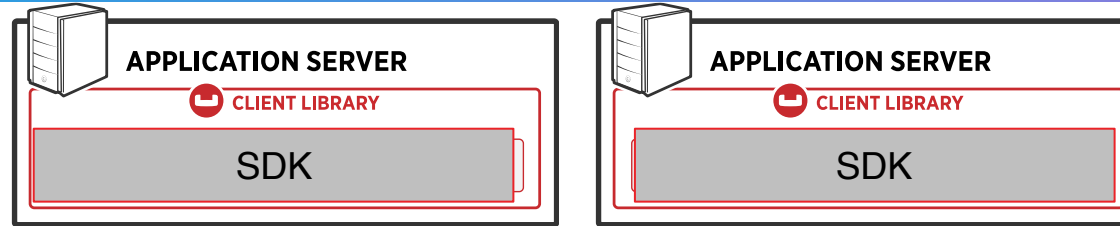
Architecture



Couchbase Architecture



Couchbase Server Cluster Service Deployment



Architecture



Product	Oracle	Couchbase
System Architecture	SMP Shared disk RAC	MPP: Shared Nothing MDS: Multi Dimensional Scaling Data Service: MPP, Hash Partitioning Indexing, FTS: MDS Scale-out Query: MDS Scale-out
Query	SQL, search, XQuery, JSON extensions	N1QL, Key-value, Full text search
High Availability	Exadata storage, Golden Gate, Log Shipping, Standby server	Built-in intranode replication (up to 3 copies) Built-in XDCR (cross data center replication)
Transactions	ACID Multi-statement	Single document atomicity Data Service consistency, Index - eventual consistency optimistic locking (CAS) additional confirmation for durability
Drivers	JDBC, ODBC, .NET, LINQ	Couchbase SDK (Java, .NET, LINQ, PHP, Python, Go), Simba JDBC/ODBC
Data Model	Normalized, Denormalized	Denormalized JSON model

Database Objects



Data Feature/Type	Oracle	Couchbase
Database	Database	Bucket
Table	Table	Bucket, Keyspace
Row	Row	Document
Column	Column	Field/Attribute
Partition	Partition (manual)	Partition (hash automatic)

DATA TYPES



Data Type	Oracle	Couchbase	JSON
Numbers	INT, BIGINT, NUMBER, FLOAT, LONG, DECIMAL	JSON Number	{ "id": 5, "balance":2942.59 }
String	CHAR, VARCHAR, VARCHAR2 , NVARCHAR,	JSON String	{ "name": "Joe","city": "Morrisville" }
boolean	CHAR(1)	JSON Boolean	{ "premium": true, "pending": false}
datetime	DATE, TIMESTAMP WITH TIMEZONE, INTERVAL	JSON ISO 8601 String with extract, convert and arithmetic functions	{ "soldat": "2017-10-12T13:47:41.068-07:00" }
spatial data	SDO_Geometry, SDO_Topo_Geometry, SDO_GeoRaster	Supports nearest neighbor and spatial distance.	"geometry": {"type": "Point", "coordinates": [-104.99404, 39.75621]}
MISSING	Fixed Schema.	MISSING	
NULL	NULL	JSON Null	{ "last_address": Null }
Objects	Fixed pre-defined COLLECTION types	Flexible JSON Objects	{ "address": {"street": "1, Main street", "city": Morrisville, "zip":"94824"}}
Arrays		Flexible JSON Arrays	{ "hobbies": ["tennis", "skiing", "lego"]}



	Couchbase
MISSING	Value of a field absent in the JSON document or literal. {“name”:”joe”} Everything but the field “name” is missing from the document.
IS MISSING	Returns true if the document does not have status field FROM CUSTOMER WHERE status is MISSING;
IS NOT MISSING	Returns true if the document has status field FROM CUSTOMER WHERE status is NOT MISSING;
MISSING AND NULL	MISSING is a known missing quantity NULL is a known UNKNOWN. Valid JSON: {“status”: Null}
MISSING value	Simply make the field of any type to disappear by setting it to MISSING UPDATE CUSTOMER SET status = MISSING WHERE cxid = “xyz232”

Couchbase: 4-valued boolean logic



In N1QL boolean propositions can evaluate to NULL or MISSING. The following table describes how these values relate to the logical operators

A	B	A OR B	A AND B
TRUE	NULL	TRUE	NULL
FALSE	NULL	FALSE	NULL
TRUE	MISSING	TRUE	MISSING
FALSE	MISSING	TRUE	MISSING
NULL	MISSING	NULL	MISSING
NULL	NULL	NULL	NULL
MISSING	MISSING	MISSING	MISSING



N1QL IN DEPTH



Statements



Feature	Oracle	Couchbase
CREATE TABLE	CREATE TABLE	couchbase-cli bucket-create
ALTER TABLE	ALTER TABLE	UPDATE customer SET, UNSET
CREATE INDEX i1 on t(a, b, c DESC);	CREATE INDEX i1 on t(a, b, c DESC);	CREATE INDEX i1 on t(a, b, c DESC);
INSERT INTO	INSERT INTO	INSERT INTO
SELECT	SELECT	SELECT
JOINS	JOINS	JOIN – INNER JOIN, LEFT OUTER JOIN
GROUP BY, HAVING	GROUP BY, HAVING	GROUP BY, HAVING
ORDER BY a ASC, b DESC	ORDER BY a ASC, b DESC	ORDER BY a ASC, b DESC
OFFSET, LIMIT	OFFSET, FETCH FIRST/NEXT ROW ONLY	OFFSET, LIMIT
Subqueries	Subqueries	Subqueries

N1QL: SELECT Statement



```
SELECT *  
FROM customers c  
WHERE c.address.state = 'NY'  
      AND c.status = 'premium'  
ORDER BY c.address.zip
```

Project Everything

From the bucket customers

Predicate

Sort order

N1QL: SELECT Statement

```
SELECT  customers.id,  
        customers.NAME.lastname,  
        customers.NAME.firstname  
        Sum(orderline.amount)  
FROM    orders UNNEST orders.lineitems AS orderline  
        INNER JOIN customers ON KEYS orders.custid  
WHERE   customers.state = 'NY'  
GROUP BY customers.id,  
        customers.NAME.lastname,  
        customers.NAME.firstname  
HAVING  sum(orderline.amount) > 10000  
ORDER BY sum(orderline.amount) DESC
```

- Dotted sub-document reference
- Names are CASE-SENSITIVE

UNNEST to flatten the arrays

JOINS with Document KEY of customers

N1QL: Query Execution Flow



```
SELECT c_id,  
       c_first,  
       c_last,  
       c_max  
FROM   CUSTOMER  
WHERE  c_id = 49165;
```

```
{  
  "c_first": "Joe",  
  "c_id": 49165,  
  "c_last": "Montana",  
  "c_max" : 50000  
}
```

1. Submit the query over REST API

8. Query result

2. Parse, Analyze, create Plan

7. Evaluate: Documents to results

3. Scan Request; index filters

5. Fetch Request, doc keys

4. Get qualified doc keys

6. Fetch the documents

Clients

Query Service

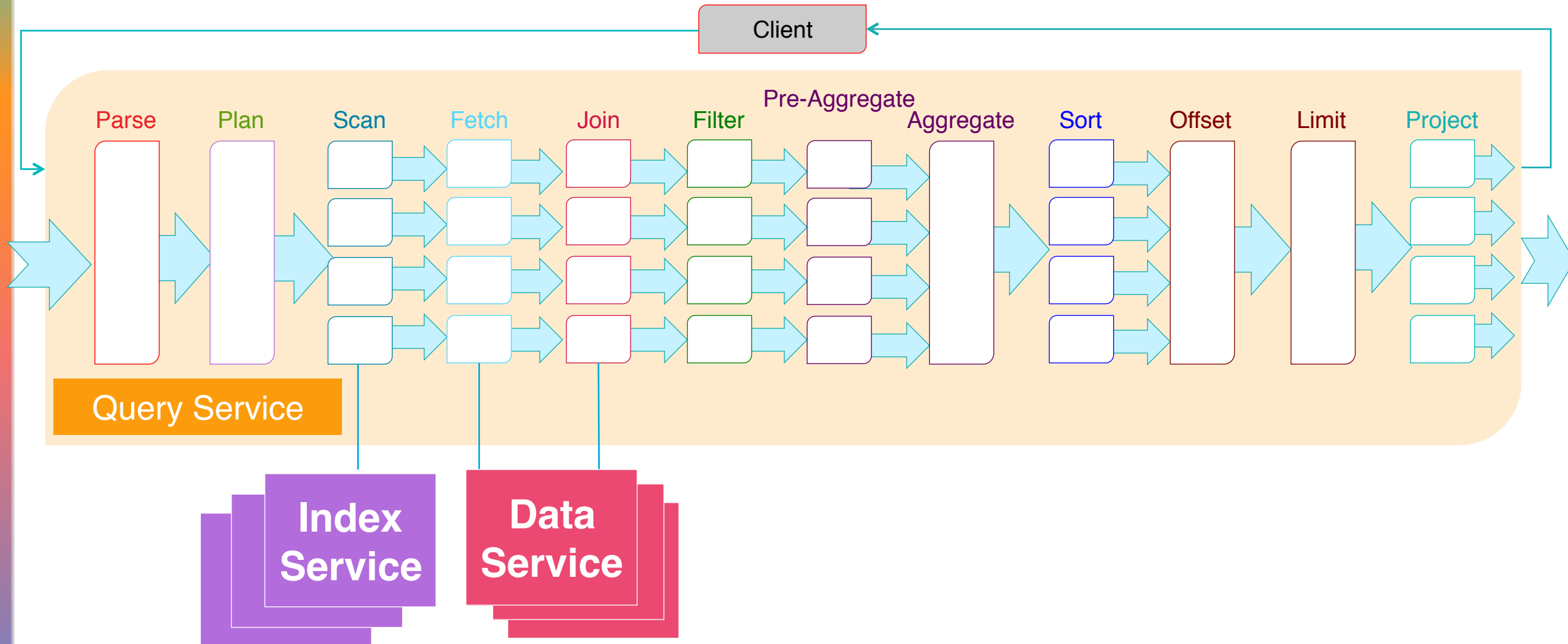
Index Service

Data Service



Server Cluster

N1QL: Inside the Query Service





Index Type	Oracle	Couchbase
Primary Index	Table Scans, Primary Index	Primary Index
Secondary Index	Secondary Index	Secondary Index
Composite Index	Composite Index	Composite Index
Functional Index (Expression Index)	Functional Index	Functional Index, Expression Index
Partial Index	Partial Index	Partial Index
Range Partitioned Index	Range partitioned, Interval, List, Ref, Hash, Hybrid partitioned Index	Manual range partitioned using partial Index
ARRAY Index	None	Yes. Nested array keys to the index is allowed.
Array Index on Expressions	None	Yes
Objects/Arrays	None	Yes



Primary Index

The primary index is simply an index on the document key on the entire bucket. The Couchbase data layer enforces the uniqueness constraint on the document key.

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

Secondary Index

The secondary index is an index on any key-value or document-key. This index can use any key within the document and the key can be of any type: scalar, object, or array.

```
CREATE INDEX def_city on `travel-sample` (city);  
SELECT * from `travel-sample` where city = "Los Angeles";
```

Composite Index

It's common to have queries with multiple filters (predicates). In such cases, you want to use indexes with multiple keys so the indexes can return only the qualified document keys.

```
CREATE INDEX travel_info ON `travel-sample` (name,type,id,icoo,iata);  
SELECT * FROM `travel-sample` WHERE name="Air France";
```



Functional Index

Create an index based on a function.

```
CREATE INDEX travel_cxname ON `travel-sample` (LOWER(name));  
SELECT * FROM `travel-sample` WHERE LOWER(name) = "american airlines";
```

Indexes on nested objects

Create an index on a nested objects using dot notation.

```
CREATE INDEX travel_geo on `travel-sample` (geo.lat);  
SELECT * FROM `travel-sample` where geo.lat is not null;
```

Partial Index

Create a composite index filtered using a where clause.

```
CREATE INDEX travel_info_partial ON `travel-sample` (name,type,id,icoo,iata)  
WHERE type="airline";  
SELECT * FROM `travel-sample` WHERE name="Air France" and type="airline";
```



Range partitioned index

Use multiple partial indexes and place them on distinct indexer nodes.

```
CREATE INDEX over5 ON `beer-sample` (abv) WHERE abv > 5 USING GSI WITH {"nodes":  
["192.0.2.1:8091"]};
```

```
CREATE INDEX over6 ON `beer-sample` (abv) WHERE abv > 10 USING GSI WITH  
{"nodes": ["192.0.2.2:8091"]};
```

- You can use complex predicates in the WHERE clause of the index. Here are some examples where you can use partial indexes: Partitioning a large index into multiple indexes using the mod function.
- Partitioning a large index into multiple indexes and placing each index into distinct indexer nodes.
- Partitioning the index based on a list of values. For example, you can have an index for each state.



Array Index

Global indexes on array elements that optimizes the execution of queries involving array elements.

Indexing simplified array expressions

Creating an index on the whole array.

```
CREATE INDEX idx ON `travel-sample` (DISTINCT `schedule`) WHERE type =  
"airline";
```



Indexing full array expressions

```
CREATE INDEX idx_sched ON `travel-sample` ( DISTINCT ARRAY v.flight FOR v IN
schedule END );
```

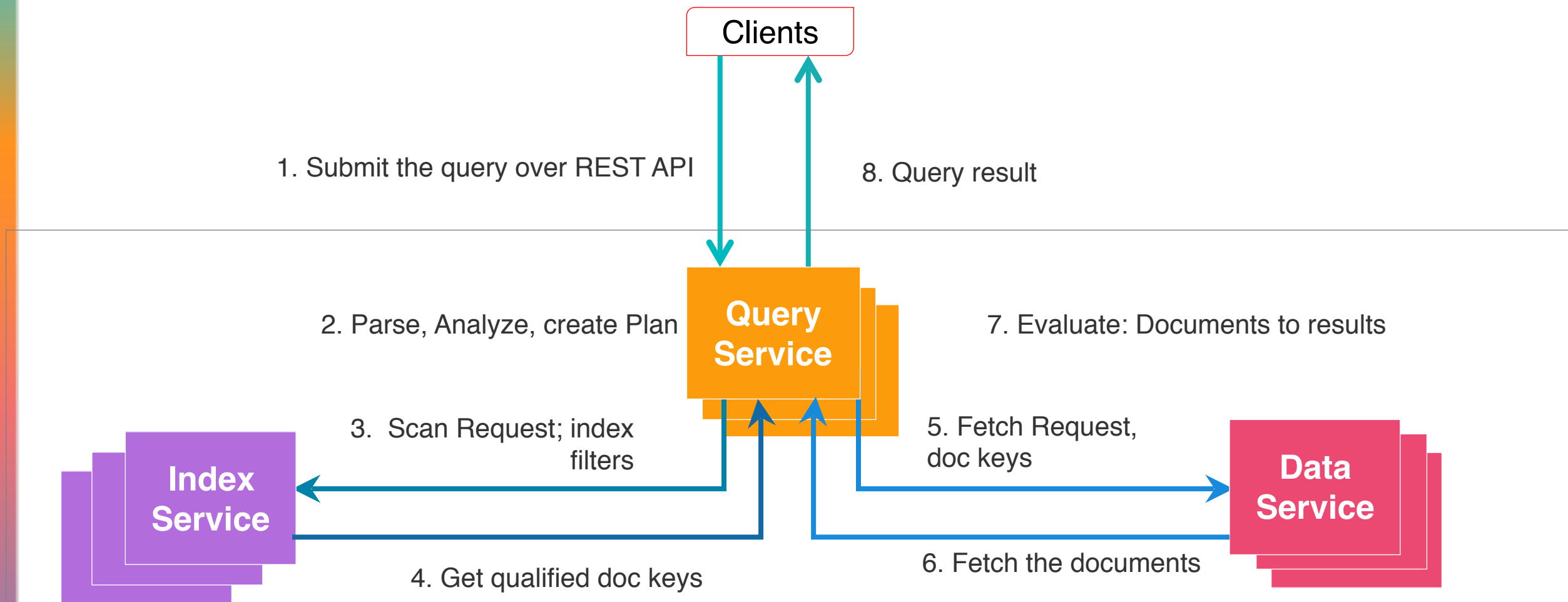
```
SELECT * FROM `travel-sample` WHERE ANY v IN schedule SATISFIES v.flight LIKE
'UA%' END;
```

```
CREATE INDEX idx_flight_day ON `travel-sample` ( ALL ARRAY v.flight FOR v IN
schedule WHEN v.day < 4 END ) WHERE type = "route";
```

```
SELECT * FROM `travel-sample` WHERE type = "route" AND ANY v IN schedule
SATISFIES (v.flight LIKE 'UA%') AND (v.day=1) END;
```

- predicate type = "route" matches that of the partial index WHERE clause.
- The ANY operator uses the index key v.flight on which the index is defined.
- The ANY-SATISFIES condition v.day=1 in the query is sargable to that in the index definition WHEN clause v.day<4.

N1QL: Query Execution Flow





Covering Index

- When an index includes the actual values of all the fields specified in the query, the index covers the query and does not require an additional step to fetch the actual values from the data service.
- An index, in this case, is called a covering index and the query is called a covered query.
- As a result, covered queries are faster and deliver better performance.

```
CREATE INDEX idx_state on `travel-sample` (state,type) USING GSI;
```

```
SELECT state FROM `travel-sample` WHERE type="hotel" AND state = "CA";
```

```
SELECT state, city FROM `travel-sample` WHERE state = "CA" and city="Los Angeles" and type="hotel";
```

```
CREATE INDEX idx_state2 on `travel-sample` (state,type,city) USING GSI;
```

Statements



Statement	Oracle	Couchbase
SELECT	<code>SELECT * FROM CUSTOMER WHERE zip = 94040</code>	<code>SELECT * FROM CUSTOMER WHERE zip = 94040;</code>
INSERT	<code>INSERT INTO CUSTOMER(id, name, status, zip) VALUES ('xyz124', 'Joe Montana', 'Premium', 94040)</code>	<code>INSERT INTO CUSTOMER(KEY, VALUE) VALUES('xyz124', {"id": "xyz124", "name": "Joe Montana", "status": "Premium", "zip": 94040})</code>
UPDATE	<code>UPDATE CUSTOMER SET zip = 94587 WHERE id = 'xyz124'</code>	<code>UPDATE CUSTOMER SET zip = 94587 WHERE id = 'xyz124'</code>
DELETE	<code>DELETE FROM CUSTOMER WHERE id = 'pqr482'</code>	<code>DELETE FROM CUSTOMER WHERE id = 'pqr482'; DELETE FROM CUSTOMER WHERE META().id = 'pqr482';</code>
MERGE	<code>MERGE into CUSTOMER using (select id from Cx where x < 10) as CN on (CUSTOMER.id = CN.id) when matched then update set CUSTOMER.o4=1;</code>	<code>merge into CUSTOMER using (select id from CN where x < 10) as CN on key CN.id when matched then update set CUSTOMER.o4=1;</code>
DESCRIBE	<code>DESCRIBE CUSTOMER</code>	<code>INFER CUSTOMER</code>
EXPLAIN	<code>EXPLAIN PLAN SELECT * FROM CUSTOMER WHERE zip = 94040</code>	<code>EXPLAIN SELECT * FROM CUSTOMER WHERE zip = 94040;</code>

Statements



Statement	Oracle	Couchbase
PREPARE	<pre>EXEC SQL PREPARE p1 FROM SELECT * FROM CUSTOMER</pre>	<pre>PREPARE p1 FROM SELECT * FROM CUSTOMER where \$1 > 5 and \$amt < 25</pre>
EXECUTE	<pre>EXECUTE IMMEDIATE :p1</pre>	<pre>\SET -args ["6"]; \SET -\$amt 23; EXECUTE P1</pre>
GRANT	<pre>GRANT SELECT ON oe.customers_seq TO hr;</pre>	<pre>GRANT select ON orders, customers TO bill, linda;</pre>
REVOKE	<pre>REVOKE UPDATE ON hr.employees FROM oe;</pre>	<pre>REVOKE update ON `travel-sample` FROM debby</pre>

SELECT



Statement	Oracle	Couchbase
GROUP BY	GROUP BY a, b, c	GROUP BY a, b, c
ORDER BY	ORDER BY SUM(d) DESC, b	ORDER BY SUM(d) DESC, b
OFFSET, LIMIT	OFFSET 400 ROWS FETCH NEXT 50 ROWS ONLY;	OFFSET 400 LIMIT 50
JOINS	INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER	INNER, LEFT OUTER. ON clause is equi joins with one side document key. (ON KEYS CLAUSE)
NEST, UNNEST	--	JSON encapsulation and decomposing JSON arrays.
Subqueries	Projection sub queries can return only one (scalar or collection) value	Projection subqueries can return zero, one or more values, treated as MISSING, scalar, object or arrays
Table Expressions (subqueries in FROM clause)	Table query expressions	Query expressions can be the first expression in FROM clause



NEST

- Nesting performs a join across two keyspaces (or a keyspace with itself). But instead of producing a cross-product of the left and right hand inputs, a single result is produced for each left hand input, while the corresponding right hand inputs are collected into an array and nested as a single array-valued field in the result object.
- Special JOIN that embeds external child documents under their parent
- Ideal for JSON encapsulation (Another way of grouping)

```
SELECT * FROM `travel-sample` route INNER NEST `travel-sample` airline ON KEYS  
route.airlineid WHERE route.type = "route" LIMIT 1;
```




UNNEST

- If a document or object contains a nested array, UNNEST conceptually performs a join of the nested array with its parent object. Each resulting joined object becomes an input to the query.
- Flattening JOIN that surfaces nested objects as top-level documents
- Ideal for decomposing JSON hierarchies

```
SELECT t.airlineid,sched FROM `travel-sample` t UNNEST schedule sched WHERE  
sched.day = 1 AND t.type="route";
```

JOINS, NEST and UNNEST can be cascaded.

JOINS



JOIN Type	Oracle	Couchbase
INNER JOIN	Full ANSI join	ON clause requires document key reference (pre 5.5). For 5.5 we have introduced ANSI JOINS syntax
LEFT OUTER JOIN	Full ANSI join	ON clause requires document key reference (pre 5.5). For 5.5 we have introduced ANSI JOINS syntax
RIGHT OUTER JOIN	Full ANSI join	Limited supportability
FULL OUTER JOIN	Full ANSI join	Unsupported
LATERAL JOIN, +++	Full ANSI join	Unsupported



Lookup JOIN



In lookup join the left-hand side of the join (“route”) needs to produce document keys for the right-hand side of the join (“airline”), this is achieved by the ON KEYS clause.

```
SELECT airline.name FROM `travel-sample` route
      JOIN `travel-sample` airline
      ON KEYS route.airlineid
WHERE route.type = "route" AND route.sourceairport = "SFO" AND
route.destinationairport = "JFK";
```

ANSI JOIN



In 5.5 we support ANSI JOIN. You can join arbitrary expressions as long as there is an index on the JOIN condition.

```
CREATE INDEX hotel_country ON `travel-sample`(free_internet,country,city) WHERE  
type = "hotel";
```

```
CREATE INDEX airport_location ON `travel-sample`(country, city) WHERE type =  
"airport";
```

```
SELECT * FROM `travel-sample` hotel  
      JOIN `travel-sample` airport  
        ON hotel.country = airport.country AND hotel.city = airport.city  
        AND airport.type = "airport"  
WHERE hotel.type = "hotel" AND hotel.free_internet = true
```

Developer Tooling



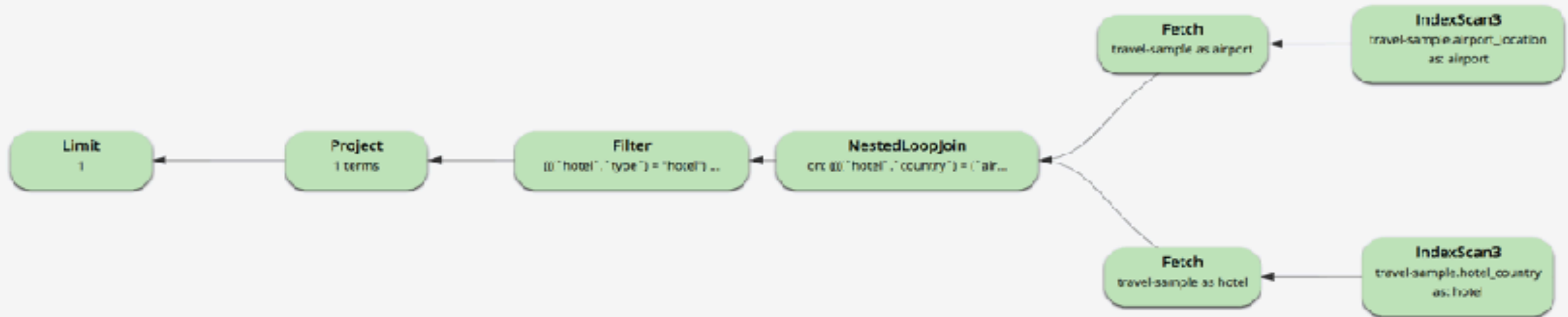
Query Results

JSON Table Tree **Plan** Plan Text

Indexes
travel-sample.hotel_country

Buckets
travel-sample

Fields
travel-sample.type travel-sample.free_internet travel-sample.*



CBQ Shell for scripting – See Dzone articles.

Optimizer



Feature	Oracle	Couchbase
Optimizer Type	Rule Based Cost Based (default)	Rule based
Query Rewrite	Yes	No
JOIN Order	Cost based	User Specified (Left to Right)
HINTS	Yes	Yes (USE INDEX)
EXPLAIN	EXPLAIN FOR	EXPLAIN
Visual Explain	Oracle Studio	Built-in to web console
Query Profiling	Oracle Studio	Built into query engine and web console





- RBAC
 - GRANT [select | insert | delete | update] ON bucket-name TO user
 - REVOKE [select | insert | delete | update] ON bucket-name FROM user
- X509 – CA signed certs as of 5.5
 - curl --cacert ./rootdir/ca.pem --cert-type PEM --cert ./chain.pem --key-type PEM --key ./nodedir/pkey.key <https://172.23.99.211:18093/query/service> -d "statement=select * from system:keyspaces"





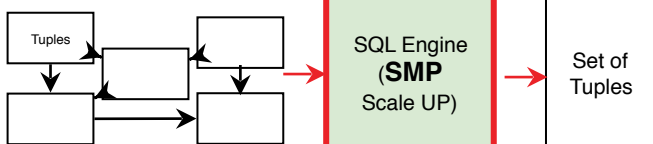
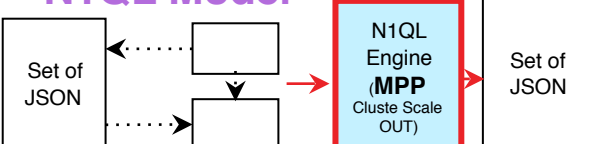
SDK	Oracle	Couchbase
Java	JDBC Driver	Couchbase Java SDK, Simba & CDATA JDBC
C	ODBC	Couchbase C SDK, Simba & CDATA ODBC
.NET, LINQ	Oracle .NET provider LINQ provider	Couchbase .NET provider LINQ provider
PHP, Python, Perl, Node.js	SDK on all these languages	SDK on all these languages
golang	Nothing official	SDK available

Deployment Options:



Deployment	Oracle	Couchbase
Your laptop	Linux, Windows, OS X	Linux, Windows, OS X
Single Node	Yes	Yes
Multi-node	Shared-disk Cluster (RAC)	Shared Nothing Cluster
VM, DOCKER	Only the non RAC version.	Yes
Kubernetes	Yes	Yes in Developer Preview
Openshift	Yes	Yes in Developer Preview
Cloud Deployment	AWS, Azure, GC, Oracle	AWS, Azure, GC
Application Upgrade without downtime	Downtime < 15 min	No downtime



SQL is English for Relational Database SQL Invented by Don Chamberlin & Raymond Boyce at IBM	N1QL is English for JSON N1QL was invented by Gerald Sangudi at Couchbase	SQL Instance Database Table Row Column Index Datatypes	N1QL Cluster Bucket Bucket, Keyspace Document Attribute Index JSON Datatypes	SQL Input and Output: Set(s) of Tuples N1QL Input and Output: Set(s) of JSON	SQL STMT CREATE TABLE CREATE INDEX ALTER TABLE SELECT INSERT UPDATE DELETE MERGE Subqueries JOIN GROUP BY ORDER BY OFFSET, LIMIT EXPLAIN PLAN PREPARE EXECUTE GRANT REVOKE DESCRIBE PREPARE EXECUTE TRUNCATE	N1QL STMT CREATE BUCKET CREATE INDEX None SELECT INSERT UPDATE DELETE MERGE Subqueries JOIN GROUP BY ORDER BY OFFSET, LIMIT EXPLAIN PREPARE EXECUTE GRANT ROLE REVOKE ROLE INFER PREPARE EXECUTE FLUSH
SQL Model 		N1QL Model 		Additional SQL Features Triggers Stored Procedures XML Constraints RAC		
SQL Optimizer Rule Based Cost Based Index Selection Query Rewrites NL, Hash, Merge join	N1QL Optimizer Rule based Index Selection NL Join	SQL Datatypes Numeric Decimal Character Date Time Timezone BLOB Spatial JSON	N1QL Datatype Numeric Boolean Character Array Object Null JSON Conversion Functions	N1QL Index Scan Consistency* Unbounded AT_PLUS REQUEST_PLUS		
SQL Logic 3 valued logic TRUE, FALSE, NULL/ UNKNOWN	N1QL Logic 4 valued logic TRUE, FALSE, NULL/ UNKNOWN, MISSING	SQL ACID ATOMIC Consistent Isolated Durable	N1QL BASE Single doc Atomic Consistent Data* Optimistic Concurrency	SQL Tooling ODBC, JDBC, .NET Hibernate BI Tools erwin TOAD		
SQL Indexes Primary Key Secondary Key Composite Range Partitioned Expression (Functional) Spatial Search	N1QL Indexes Primary Secondary Composite Partial Expression (Functional) Array Index Replica(HA) Adaptive Spatial	SQL Transactions ACID Multi-Statement Savepoints Commit/Rollback Redo, Undo	N1QL Transactions Single Document atomicity	N1QL Tooling Web Console Monitoring Profiling Dev workbench SDK Simba, Cdata BI Slamdata		
N1QL Resources query.couchbase.com						



Give developers and enterprises an
expressive, powerful, and complete language
for querying, transforming, and manipulating
JSON data.

Resources



- N1QL Tutorial - <https://query-tutorial.couchbase.com/tutorial/#1>
- Blogs – blogs.couchbase.com
- Book - <http://blog.couchbase.com/wp-content/uploads/2017/03/N1QL-A-Practical-Guide-v2.pdf>
- Forums - <https://forums.couchbase.com/c/n1ql>
- Documentation - <https://developer.couchbase.com/documentation/server/current/n1ql/n1ql-language-reference/index.html>

START A MIGRATION

