Slides adapted from CD210 - Couchbase NoSQL Data Modeling, Querying, and Tuning Using N1QL

NOTES:

1. Many thanks to Couchbase for providing this material and letting us use it!
2. These slides have been subsetted and modified for CS122D use.

Copyright © 2018 Couchbase, Inc.
Introduction to Couchbase
Our perspective...
What is Couchbase Server?

- Couchbase Server
  - Core database has key/value based orientation
  - Has indexing and querying, FTS and analytics capabilities
  - Is geared for JSON
  - Has no tables and no fixed schema
  - Runs on a networked cluster of nodes
  - Extremely fast read/write
  - Has caching and persistence layers
  - Automatically fails-over

- Couchbase Server is aimed at fast-changing data items of relatively small size (i.e., the NoSQL equivalent of an OLTP workload)
"Denormalized" JSON directly represents domain object's structure

```json
{
  order: 123,
  customer: {
    name: "Acme",
    address: "9 W. 2nd",
    email: "al@acme"
  },
  items: [
    {
      qty: 2,
      name: "Widget",
      desc: "Fit for all uses."
    },
    {
      qty: 3,
      name: "Gadget",
      desc: "Just what I need."
    }
  ],
  paytype: {
    name: "PayPal"
  }
}
```
The term "key" is used in multiple contexts

- Couchbase, N1QL/SQL and JSON

JSON property – attribute/value or name/value or key/value

```
{ "type" : "Customer" }
```

attribute or key value

Couchbase item - key/value or docID/document

```
"user::0013" { "a1":3, "a2":4 }
```
docId or key value or document
Couchbase as Key-Value Store vs. Document Store

- Couchbase is capable of storing multiple data types.
  - Simple data types such as string, number, datetime, and boolean
  - Arbitrary binary data
  - For most of the simple data types, Couchbase offers a scalable, distributed data store that provides both key-based access as well as minimal operations on the values
- Document databases encapsulate stored data into “documents” that they can operate on
  - A document is simply an object that contains data in some specific format
  - For example, a JSON document holds data encoded in the JSON format
The Couchbase server stores the data as key value pairs
- Data lives in a “Bucket” as key/value pairs
- Create a named bucket on the cluster (logical construct)
- Its data is then managed for you as vBuckets
- Its data is spread around the cluster automatically
- Each key is mapped to a vBucket on a node
- Each node has multiple vBuckets (1024 / nodes)

Q: Any guesses as to why a bucket is divided into vBuckets?
Couchbase Bucket

- Works with In-memory and Disk
- Asynchronous persistence to disk (or other media)
- Data replication at bucket level
- 20 MB per item value limit, BLOBs allowed
- Configurable (disk space)

Memcached Bucket

- In-memory only
- High Availability
- Configurable (RAM)
- 1 MB per item value limit

Ephemeral Bucket

- Alternative to Couchbase buckets, to be used whenever persistence is not required
- 20 MB per item value limit
- Data replication
- Eventual successor to memcached buckets

Copyright © 2018 Couchbase, Inc.
Storage of Data in Couchbase Server (3 of 3)

Client

Cluster Map

<table>
<thead>
<tr>
<th>vBucket 1</th>
<th>Node 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>vBucket 2</td>
<td>Node 2</td>
</tr>
<tr>
<td>vBucket 3</td>
<td>Node 3</td>
</tr>
</tbody>
</table>

get ("Doc 3")

1024 vBuckets

A: *This is why!*
+ Fast access, and
+ Fast rebalancing
<table>
<thead>
<tr>
<th>RDBMS</th>
<th>Couchbase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Bucket(s)</td>
</tr>
<tr>
<td>Table</td>
<td>Documents with common shapes</td>
</tr>
<tr>
<td>Row</td>
<td>JSON document</td>
</tr>
<tr>
<td>Fixed Schema</td>
<td>Flexible Schema</td>
</tr>
</tbody>
</table>

Map RDBMS Concepts to Couchbase

Copyright © 2018 Couchbase, Inc.
**MDS Architecture Components**

- **Data Service**: Contains the **core data**.
- **Query Service**: Contains the N1QL **query engine**.
- **Index Service**: Global Secondary Indexes (GSI), which provides **indexing** for N1QL queries.
- **FTS Service**: Contains **text** indexes for Full Text Search.
- **Eventing Service**: Lets you process **changes** to data as they happen in real-time.
- **Analytics Service**: Lets you **analyze** the data without impacting KV/N1QL performance.

Copyright © 2018 Couchbase, Inc.
Multi-Dimensional Scaling (MDS) minimizes interference between services when you separate competing workloads into independent services and isolate the services in different zones (think subclusters), the interference among them is minimized.

- Zones can be scaled independently.

Each service is deployed to an independent zone within the cluster.

No Full Text service if not needed.
Memory First Architecture

App Server

Managed Cache

Doc 1

upsert ("Doc 3", "{ ...}")

Replication Queue

Disk Queue

Disk

To other node

Couchbase Server Data Node

Copyright © 2018 Couchbase, Inc.
CB Server Data Flow (Under the Hood)

- In-memory streaming of updates to all components
- In-memory (cached) access to data and indexes
Applications use SDKs

- Couchbase offers a number of software development kits (SDKs) that enable programmatic access to Couchbase Server
Java Code for Data Service (KV)

- Procedure very similar between SDKs:
  - Create connection
  - Open bucket
  - Create JSON object
  - Create doc from object, assign key, and "upsert"
    - if exists, update
    - if not, insert
  - Get the document by key, and display

```java
// Connect to localhost
Cluster cluster = CouchbaseCluster.create();

// Open a bucket connection
Bucket bucket = cluster.openBucket("customers");

// Create a document
JsonObject user = JsonObject.empty()
    .put("first", "Walter")
    .put("last", "White")
    .put("job", "chemistry teacher")
    .put("age", 50);

// Store a document
JsonDocument stored =
    bucket.upsert(JsonDocument.create("walter", user));

// Get the document
JsonDocument walter = bucket.get("walter");
System.out.println("Found: " + walter.getString("first"));
```
Java Code for N1QL

- Very similar procedure:
  - Create connection to cluster
  - Open bucket
  - Define N1QL to execute
  - Issue query against bucket
  - Retrieve ResultSet

```java
class Main {
    public static void main(String[] args) {
        // connect to cluster
        Cluster cluster = CouchbaseCluster.create();

        // open a specific bucket on that cluster
        Bucket bucket = cluster.openBucket("travel-sample");

        // define a query
        N1qlQuery airlineQuery = N1qlQuery.simple("SELECT `travel-sample`.* FROM `travel-sample` WHERE type="airline" ");

        // issue query
        N1qlQueryResult queryResult = bucket.query(airlineQuery);

        for (N1qlQueryRow result: queryResult) {
            System.out.println(result.value());
        }
    }
}
```
The following are implemented across all Couchbase SDKs:

- **query** – execute simple, parameterized, or prepared N1QL query
- **set / insert** – add key/document or key/binary pair to bucket, **fail** if exists
- **upsert** – add key/document or key/binary pair to bucket, **success** if exists
- **replace** – replace document/binary at specified key, **fail** if it does not exist
- **remove** – delete (tombstone) value at specified key, **fail** if it does not exist
- **append** – append to current document at key
- **prepend** – prepend to current document at key
- **get / getAndLock** – retrieve document/binary at key, or get and lock (change CAS)
- **unlock** – unlock previously locked document/binary at key
- **touch / getAndTouch** – update time to live (TTL) for value at key, or get and update
- **counter** – increment or decrement a key’s numeric value
- **getFromReplica** – get replica document/binary
N1QL (for Query) Basics
Couchbase’s next generation query language

pronounced “nickel”

derives its name from non-1st normal form query language

aims to meet the needs of distributed document-oriented databases

N1QL for Query and N1QL for Analytics

is a superset and generalization of the relational language SQL (a.k.a. SQL++)

Copyright © 2018 Couchbase, Inc.
Couchbase Query Workbench

Fully Integrated with Couchbase

Let's go have a look...

cbq

Command line tool available in all Couchbase 4.x+ distributions
CBQ

- Command line tool for running and scripting N1QL queries

```bash
cbq -engine=http://<cbq-host>:8093
```

```sql
SELECT * FROM sample;
```

All the parameters with cbq are OPTIONAL

Copyright © 2018 Couchbase, Inc.
N1QL provides a rich set of features that let you retrieve, manipulate, transform, and create JSON document data.
Key features of N1QL (2 of 3)

SELECT Statement

Has the functionality of the **SQL SELECT** statement, but extends it to work with **JSON** documents

**IDEA:** Use your knowledge of **SQL** to work with the powerful **NoSQL** features of Couchbase that let you work with big data, big users, and cloud computing
Data Manipulation Statements

- INSERT
- DELETE
- UPDATE
- UPSERT
- MERGE

These statements allow you to create, delete, and modify the data stored in JSON documents by specifying and executing simple commands.
A N1QL (for Query) Query

N1QL Query

SELECT
FROM
WHERE
...
SELECT * FROM sample WHERE fname = 'Robert';

Only a SELECT clause is required in a query

The wildcard * selects all attributes of the document (without “flattening”...)

Queries can return a collection of different document structures or fragments. However, they will all match the conditions in the WHERE clause

Remember – there IS NO SCHEMA in NoSQL-land...! (So: [ ] 😐...?)
Let’s Run Our First N1QL Query!

```sql
SELECT * FROM couchmusic2 WHERE type = "userprofile";
```

```
[{
  "code": 4000,
  "msg": "No index available on keyspace couchmusic2 that matches your query. Use CREATE INDEX or CREATE PRIMARY INDEX to create an index, or check that your expected index is online.",
  "query_from_user": "select * from couchmusic2 where type = "userprofile"
}
]
```

Copyright © 2018 Couchbase, Inc.
CREATE PRIMARY INDEX ON couchmusic2;

- This statement allows you to create a primary index
- Primary indexes contain a full set of keys in a given bucket
- Primary indexes are optional and only required for running ad hoc queries (similar to relational table scans) on a bucket if the query is not supported by an appropriate secondary index

*Note*: Best to drop primary indexes when development is done! (All of your application’s queries should have appropriate secondary indexes by then.)
SELECT lastName AS lname FROM sample
WHERE fname='Robert'

- Data in Couchbase Server is stored in the form of documents, not rows or columns
SELECT children[0].fname AS cname FROM sample
WHERE fname='Robert'

- Documents can have nested elements and embedded arrays, a few additional operators are needed
- The ‘.’ operator is used to refer to children, and the ‘[]’ is used to refer to an element in an array

```json
{
  "results": [
    {
      "cname": "Matthew"
    }
  ]
}
```
Couchbase always stores meta-data about a document outside of the document. The META() function is used to access the meta-data for each document. This is how you can access the documents' keys if needed:

```
SELECT META(sample).id ...
```
Specific document keys within a bucket are queried using the USE KEYS clause

```
SELECT fname, email
FROM sample
USE KEYS ["user:robert:12", "user:melisa:43"]
```

The query fetches the documents based on the list of keys from the “sample” bucket.
Data Manipulation Statements

allows you to create, delete, and modify the data stored in JSON documents by specifying and executing simple commands.

Copyright © 2018 Couchbase, Inc.
INSERT

- Inserts the document to the bucket

Remember: Couchbase stores **KEY:DOCUMEN T** pairs

```
INSERT INTO sample (KEY, VALUE)
VALUES ("melisa_2908", {"name":"Melisa Miranda", "type":"contact"})
```

Also, note the difference in the syntax with SQL's insert statement
The syntax for the UPSERT statement is similar to the INSERT:

```
INSERT INTO sample (KEY, VALUE)
VALUES ("melisa_2908", \{"name":"Melisa Miranda", "type":"contact"\})
```

What if the key already exists?

```
UPSERT INTO sample (KEY, VALUE)
VALUES ("melisa_2908", \{"name":"Melisa Miranda", "type":"contact"\})
```

For INSERT the key being inserted must not exist. If the key exists, an UPSERT behaves like an UPDATE.
- Documents (KV pairs) can be deleted using the DELETE clause

**DELETE FROM** sample s  
**WHERE** s.title = "Mr."

Remove all documents with title "Mr."

**DELETE FROM** sample s  
**USE KEYS** "melisa_2908" RETURNING s

The returning clause will return the list of keys that were deleted from the bucket

**DELETE FROM** sample s  
**WHERE** s.name like "Cla%"

Removes all documents with name Clarence, Clara, Clarissa
**UPDATE**

- UPDATE replaces a document that already exists with updated values

```
UPDATE sample
  USE KEYS "melisa_2908"
  SET type = "actor" RETURNING sample.type
```

The returning clause will return the list of keys that were updated from the bucket.

```
UPDATE sample s USE KEYS "melisa_2908" UNSET c.gender FOR c IN children END RETURNING s
```

unsets the "gender" attribute in the "children" array for the document with the key, "mel" in the "sample" keyspace/bucket.
N1QL SELECT (vs. SQL SELECT)

- Before we go there, a few quick comments:
  - Knowing SQL → knowing N1QL (so we won’t discuss N1QL from zero!)
  - N1QL for {Query, Analytics} has a rich function library you should explore
  - The two languages are mostly the same (roughly: Query ⊆ Analytics)
  - See Don Chamberlin’s book (PDF) in the course materials on the wiki!
  - Also see the documentation online at Couchbase

- Which N1QL should I use? (Since I seem to have 10¢ to spend?)
  - Many short application queries requiring ~milliseconds → N1QL for Query
  - Large ad hoc / reporting queries requiring parallelism → N1QL for Analytics

- Soon we’ll take a deeper look at N1QL’s vs. SQL’s SELECT!
  - And we’ll do it in the context of N1QL for Analytics...