



UNIVERSITÀ  
di **VERONA**

# Fondamenti di Sistemi Informativi

## *Exam project*

---

Cristiano Di Bari

# Couchbase



Couchbase is a **distributed, multimodel NoSQL database**.

Couchbase' architecture supports a flexible JSON data model and uses familiar relational and multimodel data access services.

Couchbase was originally founded through the merger of two database companies, CouchOne and Membase. The first developed a highly-reliable document database (CouchDB), while the second employed developers of memcached, a high-performance, memory-first, key-value database. The merge led to the design of Couchbase, a reliable, scalable, fast in-memory, key-value database with document-based access and storage.

In this model, document identification “keys” store “value” data as a JSON document.



# JSON Data Model

Couchbase stores data as individual **JSON documents**. A document represents a single instance of an application object (or nested objects) and can also be considered analogous to a row in a relational table.

A JSON document's structure consists of its inner arrangement of key-value pairs.

Couchbase provides greater flexibility than the rigid schemas of relational databases, allowing JSON documents with varied schemas and nested structures.

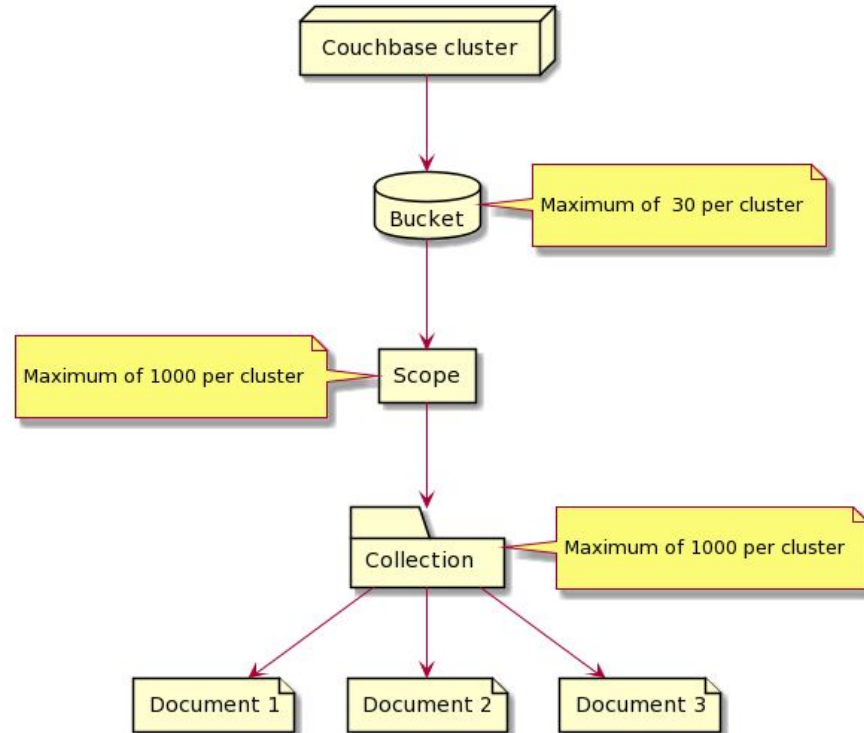
Schemas are defined by application developers and managed by applications. This is in contrast to the relational model where the database manages the schema.

# Data Model

Couchbase offers a multi-level data containment and organization structure to organize documents.

- **Buckets:** the topmost container in Couchbase. It is the logical equivalent of a database in relational systems.
- **vBuckets:** are shards or partitions of data that are automatically distributed across nodes. They are managed internally by Couchbase and not interacted with directly by the user.
- **Scopes:** are an intermediate data organization structure similar to a relational database schema. They simplify support for isolating data and access that data.
- **Collections:** are categorical or logically organized groups of documents. Collections offer a similar construct to relational tables, in that they contain documents that are alike.
- **Documents:** are the foundational construct of Couchbase and conform to JSON structural standards

# Data containment model





# Document access methods

Couchbase offers several ways for applications to access the data:

- **Key-value:** an application provides a document ID (the key), Couchbase returns the associated JSON or binary object.
- **N1QL Query:** SQL-based query syntax to interact with JSON data, similar to RDBMS, returns matching JSON results.
- **Full-text search:** using text analyzers with tokenization and language awareness, a search is made for a variety of field and boolean matching functions.
- **Eventing:** custom Javascript functions are executed within the database as data changes or based on timers




# Couchbase architecture

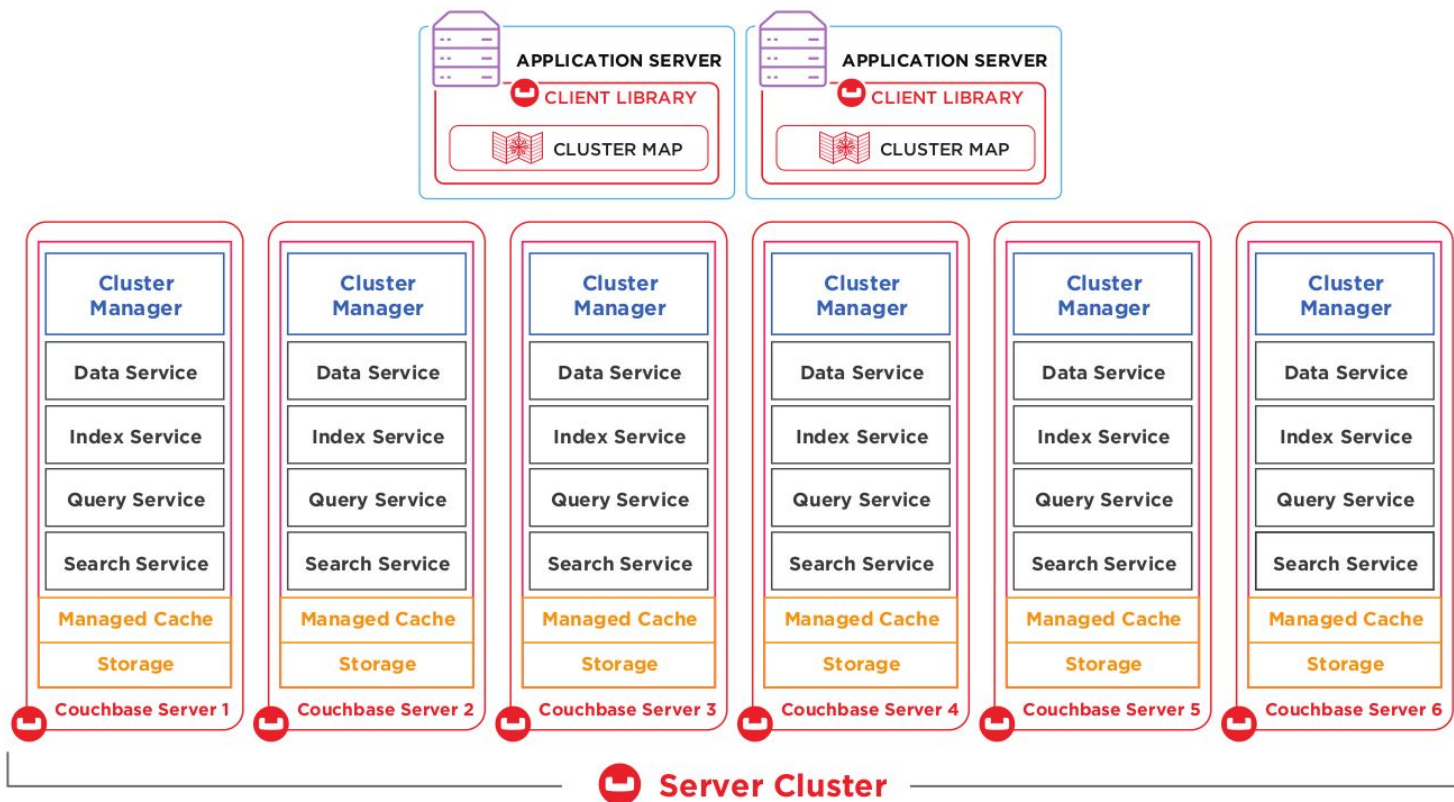
A Couchbase cluster consists of one or more instances of Couchbase Server, each running on an independent node, that operate in a **peer-to-peer topology**. Data and services are shared across the cluster.

Even though the services running on each node can be managed as required, there exists only one type of node: there is no concept of master nodes, slave nodes or config nodes.

Components of a node include the cluster manager and, optionally, the data, query, index, analytics, search, eventing and backup services.

Nodes can be added or removed easily through a rebalance process, which redistributes the load evenly across all nodes.









# Couchbase architecture

When a cluster has multiple nodes, the Couchbase **cluster manager** runs on each node. The cluster manager is a service that supervises server configuration and interaction between servers in a cluster. It also manages replication and rebalancing operations.

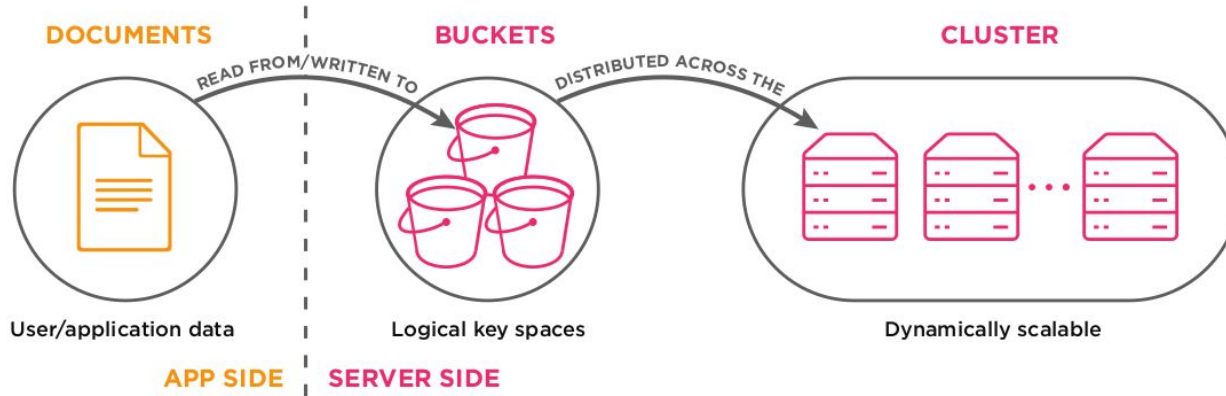
Although the cluster manager is executed locally on each node, a clusterwide **orchestrator** is elected to oversee cluster conditions and carry out appropriate cluster management functions.

When a machine crashes or become unavailable, the cluster orchestrator notifies all other nodes and promotes to active status all the replicas associated with the unavailable node. The cluster map is then updated on all nodes and clients.

If the orchestrator crashes then other nodes will detect it and will elect a new orchestrator.

# Data distribution

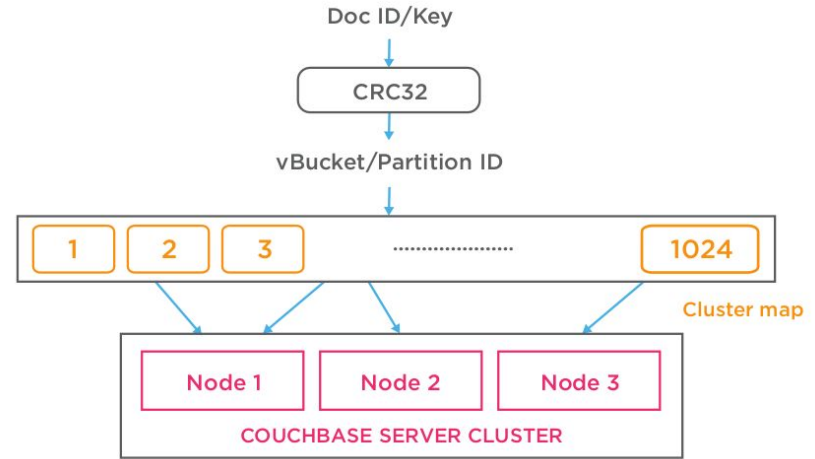
Couchbase partitions data into **vBuckets** (synonymous to shards) to automatically distribute data across nodes. Shards do not have a fixed physical location on nodes, therefore, there exists a mapping of the shards to nodes which is known as cluster map.



# Client connectivity

Clients are continually aware of the cluster topology through **cluster map** updates from the cluster manager. They automatically send requests from applications to the appropriate nodes for access, query, etc.

When creating documents, clients apply a hash function (CRC32) to every document that needs to be stored in Couchbase, and the document is sent to the server where it should reside.





# Couchbase and CAP Theorem

Couchbase can be configured in different ways: if transaction support is enabled then the system will privilege **consistency** over availability. Otherwise when the system is geographically dispersed and **availability** is preferable against consistency then the system can be configured to satisfy that need.

By default, writes in Couchbase are async: replication and persistence happen in the background, and the client is notified of a success or failure. The updates are stored in memory, and are flushed to disk and replicated to other Couchbase nodes asynchronously. You can choose to have the update replicated to other nodes or persisted to disk, before the client responds back to the app.

# Couchbase consistency

Read and writes for a single document (key value access) are strongly consistent. Meanwhile when collections are queried then Couchbase guarantees an eventual consistency.

Couchbase provides several levels of control over query consistency, allowing the application to choose between faster queries (ignoring pending mutations) and stronger consistency:

- **not\_bounded** (default): return the query response immediately.
- **at\_plus**: block the query until its indexes have been updated to the timestamp of the last update. This can be used to implement “read-your-own-writes”.
- **request\_plus**: block the query until its indexes are updated to the timestamp of the current query request. This is a strongly consistent query.

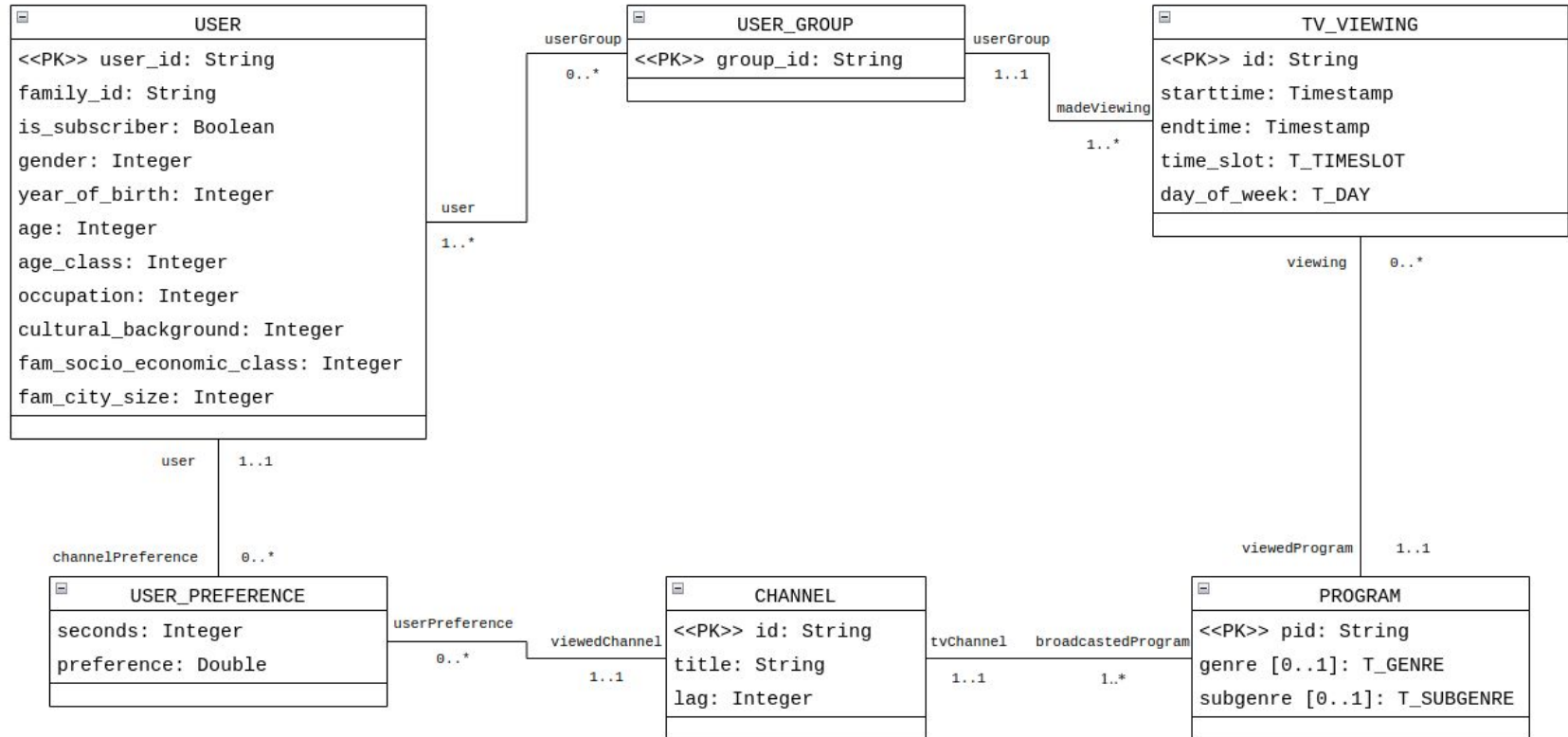
# Auditel Dataset

The chosen dataset contains data on Italian television audience divided by user groups.


It is possible to identify some entities that characterize the information contained in the reports:


- Users who watch television programs, demographic information and socio-economic status information are reported for each user together with preferences with respect to channels.
- Channels available on TV.
- Programs broadcast by each television channel, each belonging to a specific genre.
- Viewings of television programs made by groups of users who were together in front of the same television.


# Conceptual Design




# Conceptual Design

|  <<Enumeration>><br>T_GENRE |
|--|
| entertainment<br>information<br>kids and music<br>movie<br>society<br>sport<br>other programs                |

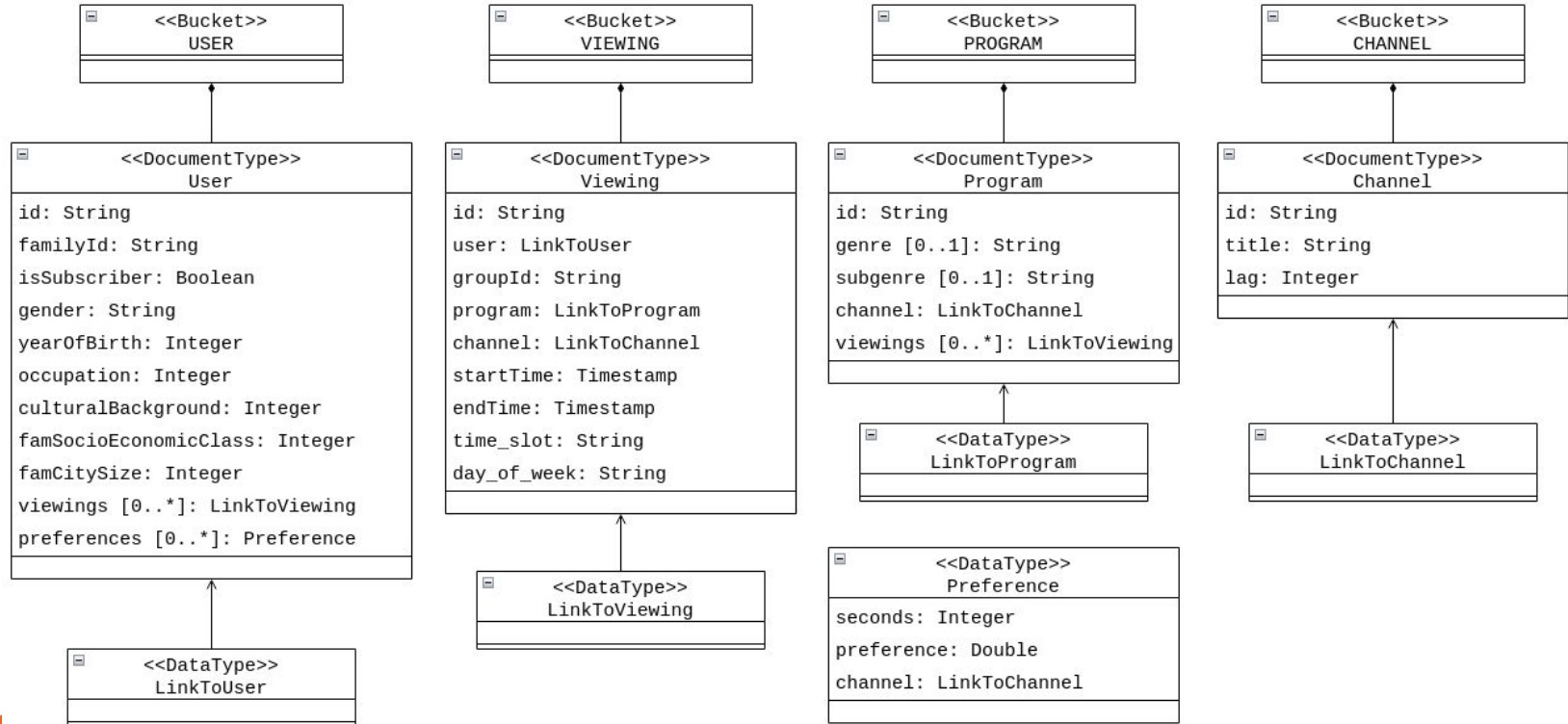
|  <<Enumeration>><br>T_SUBGENRE |
|---|
| music<br>basket<br>comedy<br>adventure<br>documentary<br>art and culture<br>cinema<br>...                       |

|  <<Enumeration>><br>T_TIMESLOT |
|---|
| DayTime<br>EarlyFringe<br>EarlyMorning<br>GraveyardSlot<br>LateFringe<br>Morning<br>PrimeAccess<br>PrimeTime      |

|  <<Enumeration>><br>T_DAY |
|--|
| weekday<br>weekend   |



# Document DB Design



# Query 1

Given a user (id) and a period of time (start-end), provide the list of the viewings made during that period.

```
1  SELECT W
2  FROM `user-record-collection` AS U USE KEYS $USER_ID
3  UNNEST U.viewings V
4      JOIN `viewings-record-collection` AS W ON V = META(W).id
5  WHERE W.startTime >= $START_PERIOD
6      AND W.endTime <= $END_PERIOD
```

## Query 2

Given a period of time (start-end), find the most viewed program in terms of seconds by reporting the id of the program and the channel.

```
1 CREATE INDEX viewings_startTime_idx ON `viewings-record-collection`(startTime);
2 CREATE INDEX viewings_endTime_idx ON `viewings-record-collection`(endTime);
```

```
1 SELECT V.programId, SUM(DATE_DIFF_STR(V.endTime, V.startTime, 'second')) AS duration
2 FROM `viewings-record-collection` AS V
3 WHERE V.startTime >= $START_PERIOD
4       AND V.endTime <= $END_PERIOD
5 GROUP BY V.programId
6 ORDER BY duration DESC
7 LIMIT 1
```

## Query 3

Given a program, find the largest group of users who have viewed that specific program.

```
1  WITH groups_count AS (  
2      SELECT W.groupId, COUNT(W.userId) AS numUsers  
3      FROM `auditel-bucket`._default.`program-record-collection` AS P USE KEYS $PROGRAM_ID  
4      UNNEST P.viewings V  
5      JOIN `auditel-bucket`._default.`viewings-record-collection` AS W ON V = META(W).id  
6      GROUP BY W.groupId  
7  )  
8  SELECT G.groupId, G.numUsers  
9  FROM groups_count AS G  
10 WHERE EVERY groupCount IN groups_count SATISFIES groupCount.numUsers <= G.numUsers END
```

