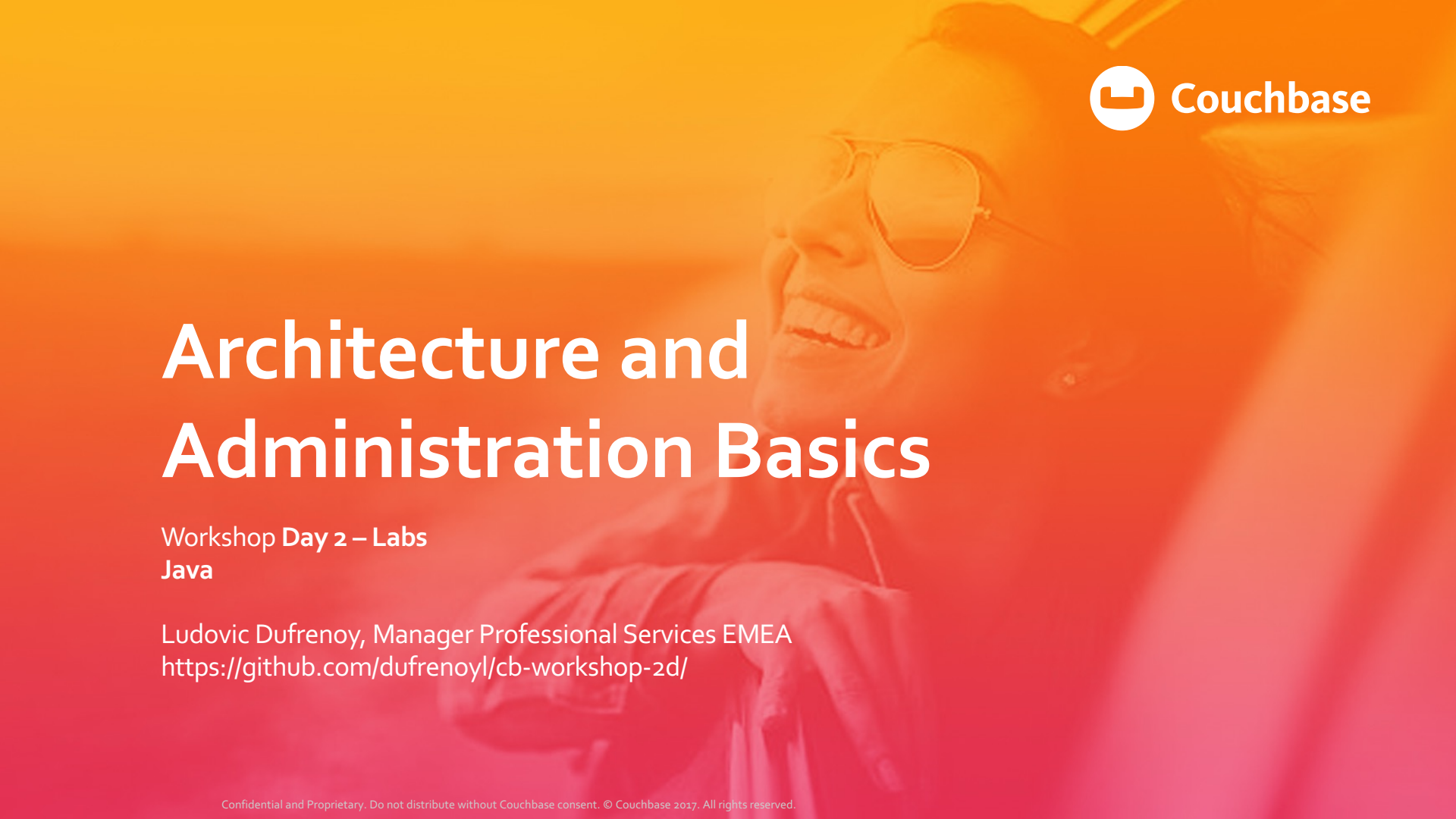


Architecture and Administration Basics

Workshop **Day 2 – Labs**
Java

Ludovic Dufrenoy, Manager Professional Services EMEA
https://github.com/dufrenoyl/cb-workshop-2d/

1 Installation & Configuration SDK

SDK Java - libcouchbase Installation

Perform the following steps in order to add a dependency to libcouchbase

- Clone the source repository in your home folder.

```
git clone https://github.com/vgasiunas/cb-workshop-2d.git
```

- Make sure Couchbase Server is running on your Machine.
- Go to the java/1 directory and edit the pom.xml
- Change Couchbase version to the latest. (2.7.2 on Feb 5th 2019)

```
<dependency>
    <groupId>com.couchbase.client</groupId>
    <artifactId>java-client</artifactId>
    <version>2.7.2</version>
</dependency>
```

https://developer.couchbase.com/documentation/server/current/sdk/java/start-using-sdk.html

Load the Project into NetBeans

Perform the following steps:

- Start a Server X on your Oracle Virtual Box from a Terminal.

```
startxfce4
```

- Start NetBeans IDE from the Desktop.
- Open the Existing projects "1" (from scratch) ; "2" (basics) ; "3" (solutions) in NetBeans IDE
- Create a bucket in Couchbase UI named "workshop".
  - 100MB
  - Flush enabled
  - No Replica.

- Build & Run your application => should be successful.

Documentation & Examples

Open the documentation for libcouchbase!

- https://developer.couchbase.com/documentation/server/current/sdk/java/sample-app-backend.html
- https://developer.couchbase.com/documentation/server/current/sdk/java/start-using-sdk.html
- http://docs.couchbase.com/sdk-api/couchbase-java-client-2.7.2/

- Open the "3" project => This is the solution.
- Open the "1" or "2" project => This is where you start.

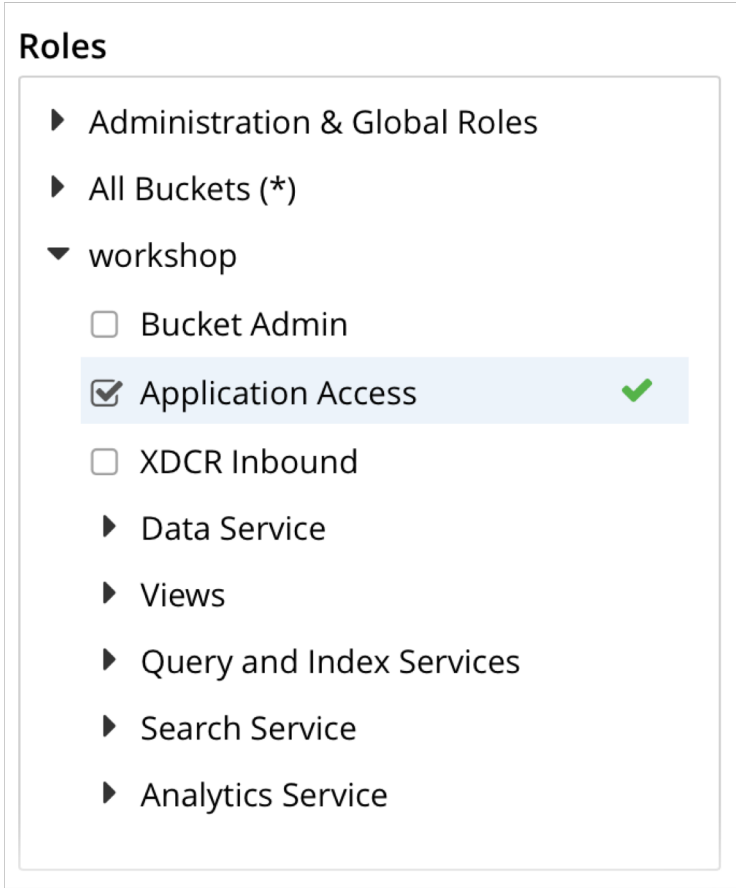
- Check the Sample App backend https://docs.couchbase.com/java-sdk/2.7/sample-application.html

2 | Managing Connections

Create an applicative User via RBAC

Perform the following operations:

- Go the Security Tab in Couchbase
- Create a new User.
- Username = “workshop”
- Password = “couchbase”
- Roles
  - Application Access on ‘workshop’



Cluster and Bucket

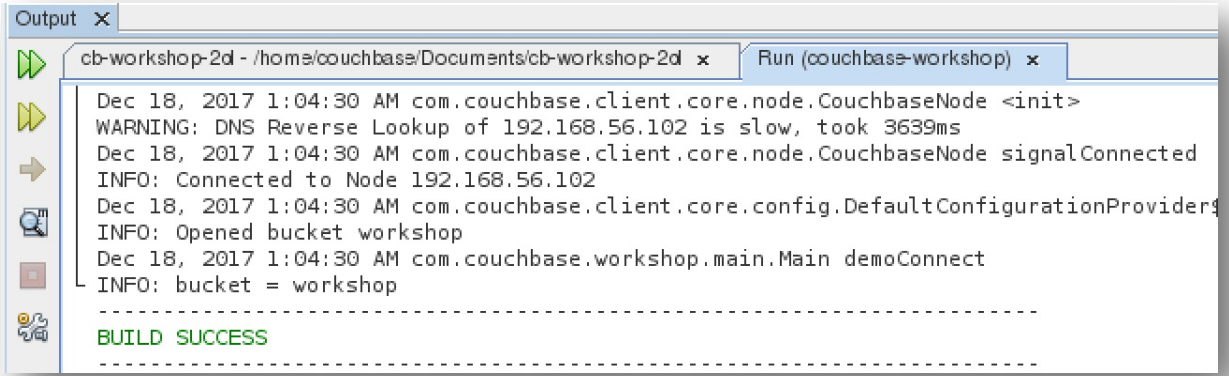
- Cluster
  - **Bootstrap the Connection**
  - **Just needs one or multiple IP-s / host names**
- Bucket
  - **Perform data operations**
  - **Thread-Safe**
  - **Reuse it (Singleton)**

Connecting to Couchbase with RBAC

Update cb.properties file with:

- Couchbase cluster IP
- Bucket name
- Username and password

Bootstrap your connection and run your application:



3 Working with Documents

Document

- Properties
  - **Content**
  - **Metadata (id, CAS value, expiry)**
- Types
  - **JsonDocument**
  - **BinaryDocument**
  - **SerializableDocument**
  - **RawJsonDocument (to use own JSON Serializer)**

Operations

- Get
- GetFromReplica
- GetAndLock
- Touch
- Upsert
- Insert
- Replace
- Append
- Prepend
- Unlock
- …

Management

- ClusterManager
  - **info**
  - **hasBucket**
  - **getBucket**
  - **insert/remove/updateBucket**
- BucketManager
  - **info**
  - **flush**
  - **get/insert/upsert/remove/publishDesignDocument**

The Patterns

- In the Observer pattern, Observers are used to observe a subject (Observable). The Observers are realizing (signaled by the Observable) if the state of the Observable is changing by being able to react on such an event. It's easy to see that you can use it also for asynchronous data access.
- The Iterator pattern allows to iterate through a collection of objects without the need to know the internal structure of the collection of objects. To combine the Observer pattern with the Iterator pattern means that you can receive a data stream (instead the whole collection of data) by then reacting on the arrival of new data items.
- The functional aspect is that you may pass functions (E.G. as anonymous classes or lambda expressions) as the arguments to an Observable.

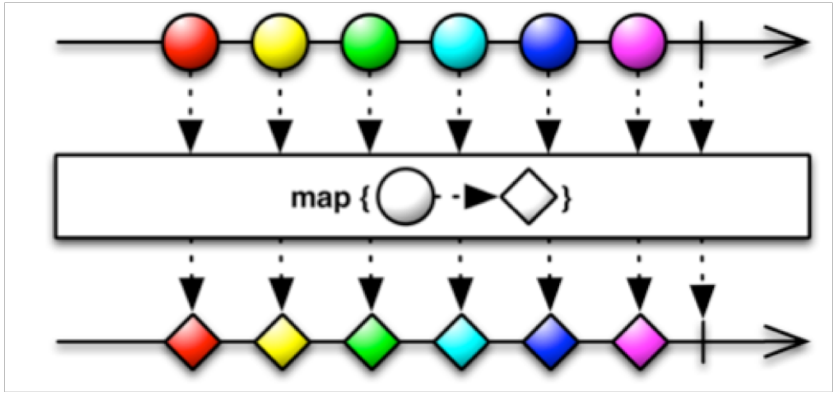
Observable

- Observable<T> and Observer<T>
- Observable = a Stream of Data
- Rx operators to manipulate the stream

	Single	Multiple
Sync ("pull")	T	Iterable<T>
Async ("push)	Future<T>	Observable<T>

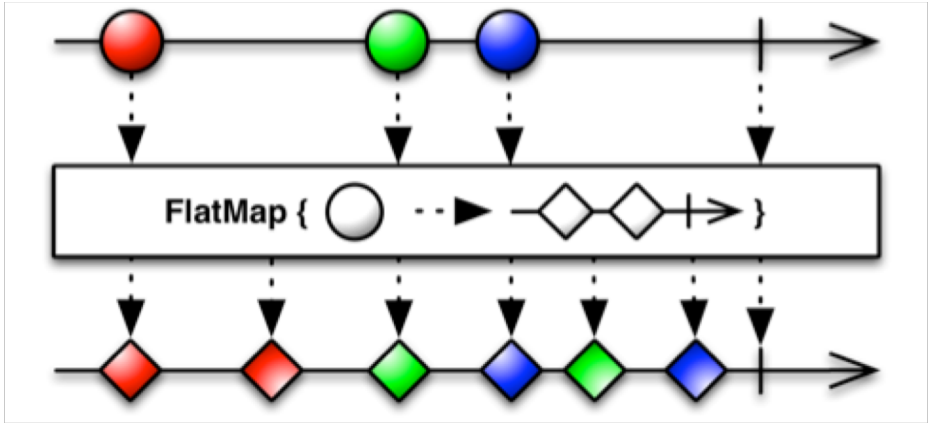
Transformation

- Observable<T> source
- Observable<R> transformed = source,map(function)
- map(function) transforms each T into R



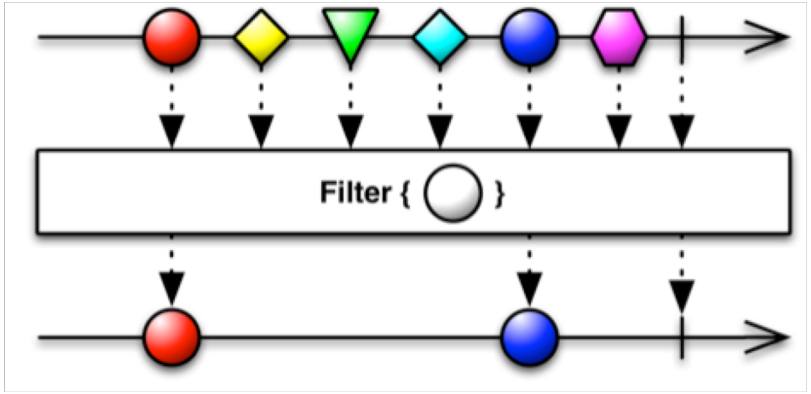
Transformation

- flatMap is like map but emits again an Observable which is then flattened into one Observable (output stream)
- When the transformation must be also async.



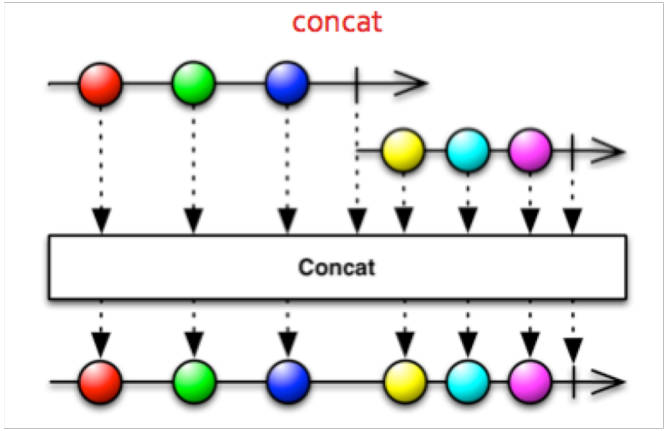
Filter

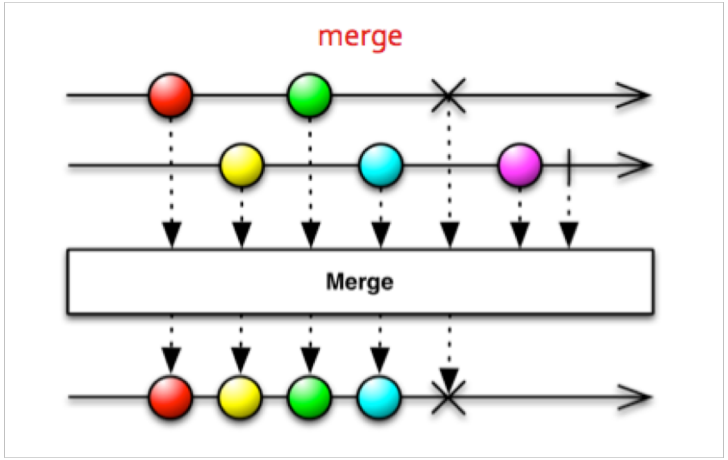
- Observable<T> filter(predicate)
- E.G.
  - **first()**
  - **take(n)**
  - **skip(n)**
  - **…**



Combine

- E.G.
  - **merge(observableOfT)**
  - **concat(observableOfT)**
  - **…**





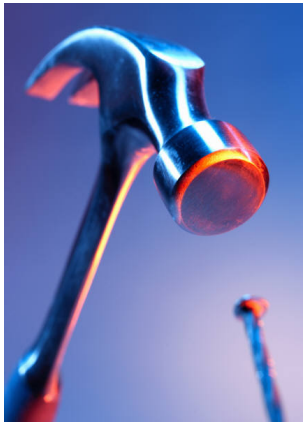
Subscribe

- By passing an Observer or the following Actions
  - **onError**
  - **onNext**
  - **onComplete**

CRUD Operations

Create a company document which has the following properties:

- Id
- Type (“company”)
- Name
- Address



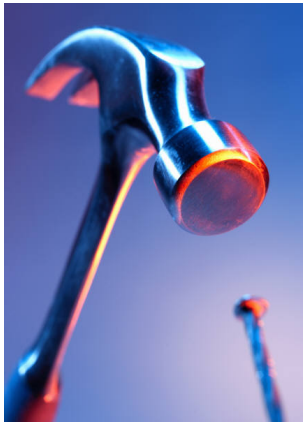
Create 7 user documents those have the following properties:

- Uid
- Type (“user”)
- First name
- Last name
- Email address
- Birthday

CRUD Operations

Assign every user to a company!

For every user create an email lookup document! (Optional)

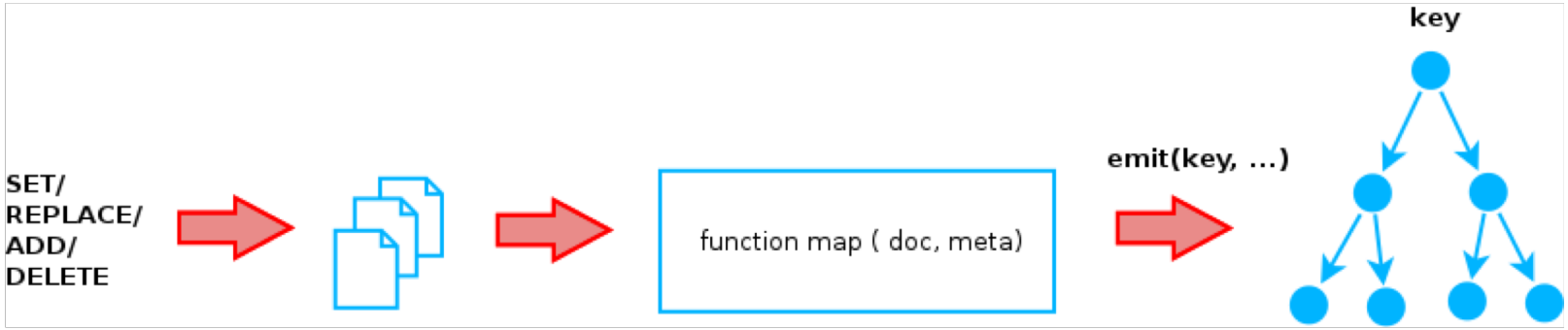


4 | Indexes using VIEWS

Views

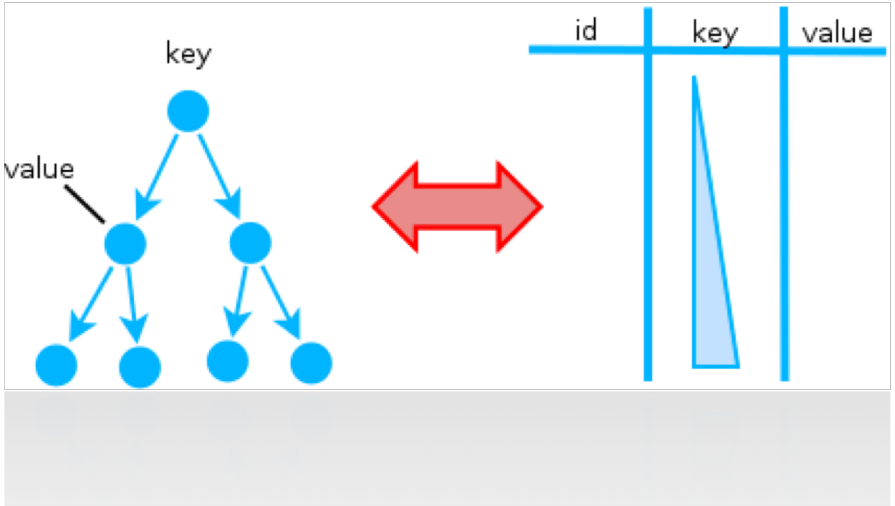
- Organized in Design Documents
- Incremental Map-Reduce
- Spread indexing load across nodes

Map	Reduce
Process, filter, map and emit a row	Aggregate mapped data Built in: _count, _sum, _stats



Views

- Multiple roles
  - A Primary Index to access all document id-s
  - A Secondary Index as an alternative access path
  - A View provides you an alternative view on your data

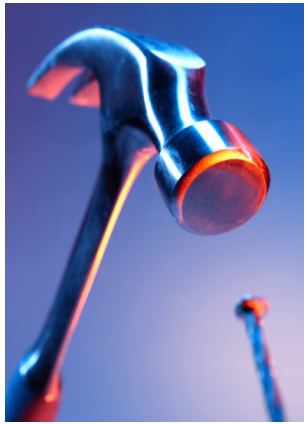


Querying via Views

Create a new Design Document with the name 'persons'!

Create a View with the name 'by_birthday'

Implement a View which:



- Filters users
- Emits the birthday as a date array and so as a compound key
- Emits the first name and the last name as the value

Query the View by using the web browser:

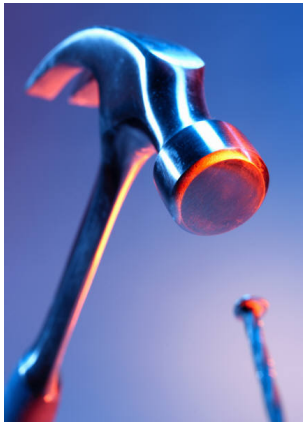
- Use the built-in reduce function _count and group the result by the year of birth

Querying via Views

Query the View by using the web browser:

- Use the built-in reduce function _count and group the result by the year of birth

Query the View by using the Java client!

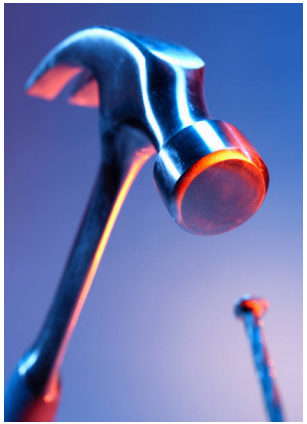


4 | Indexes using GSI (Plasma)

Querying via N1QL

Make sure that Primary Index is created!
Also create a Secondary Index on the type attribute!
Create a Secondary Index on the last name!

- Create the indexes programmatically by using the Index API



Querying via N1QL

Query for users by their last name!

- Use the builder API for building the N1QL statement

Query users of a specific company and with a specific last name by using a JOIN!

- Use a query string for this purpose

5 | Error Management & Logging

Error Handling

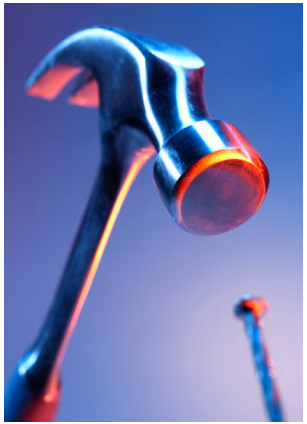
- In the subscribe
  - **onError(function)**
- Return default value
  - **onErrorReturn(T defaultValue)**
- Defer to a backup stream on error
  - **onErrorResumeNext(Observable<T> backup)**
- Retry on error
  - **retry(predicate)**

6 | FTS Full Text Search

Querying via FTS

Make sure the default FTS is created!
Give it the following name: "idx_default"

- Search for an email in the "workshop" bucket.



Thank you

Couchbase