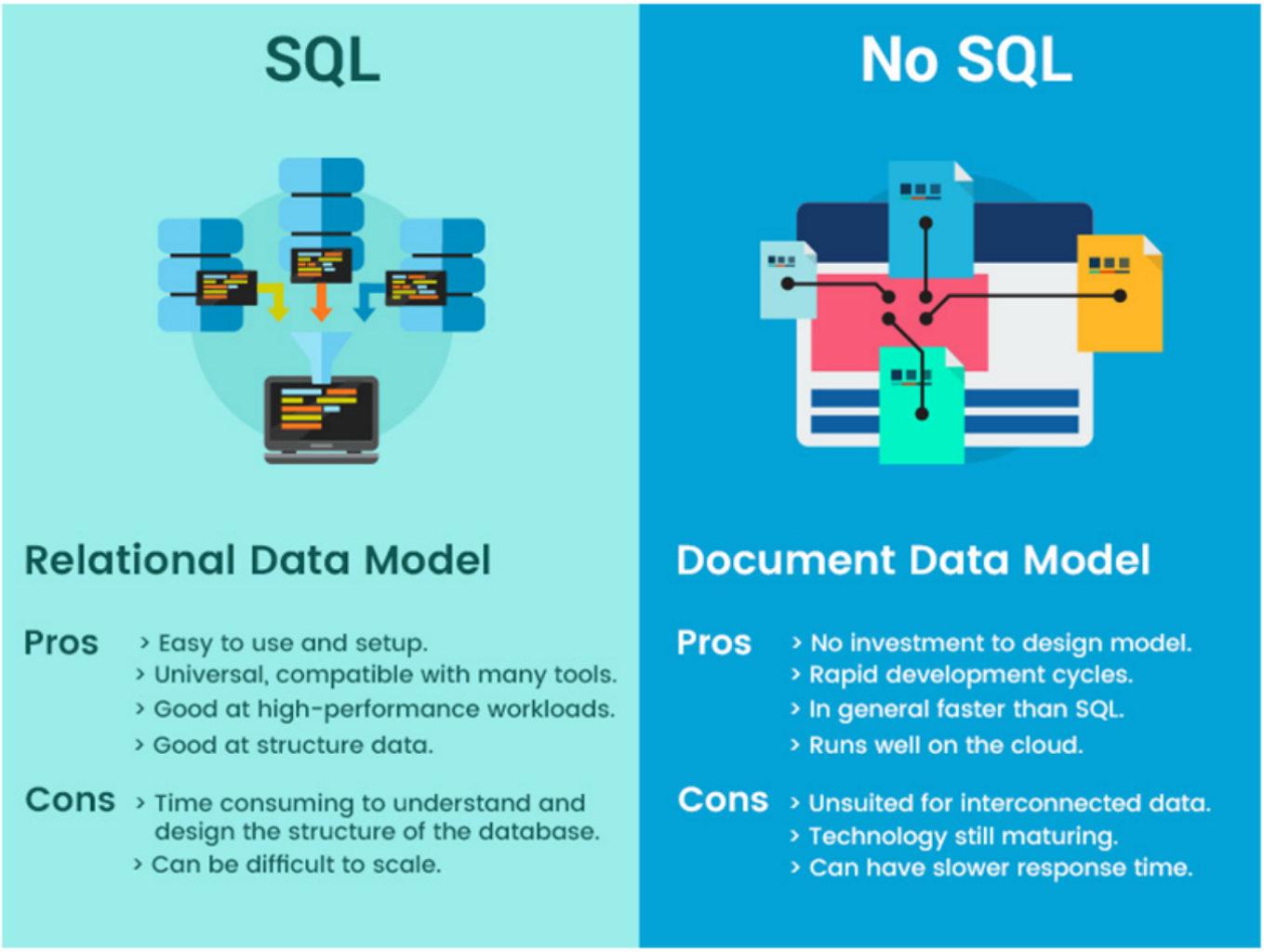


NOSQL SUMMARY

MongoDB and HBase

By Jonas Hellevang

Relational (SQL)	**MongoDB**
Database	Database
Table	Collection
Row	Json documents
Index	Index
Join	Embedded document

- database can have one or more collections
- collection, you can have none (empty collection), one or more JSON documents

show dbs	show the databases
use my_database	create or connect to a database
db	show the database I am using
show collections	shows the collections in the current database
db.my_collection.insert ({ attribute1:"hello", attribute2: 100 })	insert a document in a collection
db.other_collection.find ( )	
db.other_collection.find().pretty()	
db.other_collection.find().count()	
db.other_collection.find().pretty().count()	
db.agenda.find({age: 25})	filter the documents with a condition
db.agenda.find ({"address.province": "Murcia" }	filter the documents with a condition with Embedded Document
db.agenda.find ({children: "Ines" })	filter the documents with a condition with array field

$gt or $gte	Greater than or equal age: {$gt:25}
$lt or $lte	Lower than or equal
$eq or $ne	Equal or not equal
$in or $ni	In or not in

({name: /^Juan/ })	Begins with Juan
({name: /Juan$/ })	End with Juan
({name: /Juan/ })	Contains Juan
({name: /Juani/ })	Contains Juan(case insensitive)

db.agenda.find ({ $or: [{ "address.province":"Madrid" }, { "age":{$gte:30} }] })	or

db.agenda.find ({ $and:[{age:{$gte:30}} , {"address.province":"Madrid"}] }, { name:1, lastName:1, _id:0})	And (when using the same field many times in the conditions.)
db.agenda.find({… }, {name : 1 , "_id" : 0})	Filtering field
db.test.remove({}) or db.test4.deleteMany({ y:2 })	Remove all documents of test collection (don't forget : {})
db.test2.remove({ y:2 } , {justOne: true}) or db.test2.deleteOne({ y:2 })	Delete just 1 doc
db.test.drop()	Dropping a collection
db.<collection>.update ({ <conditions> }, {<modification> }, {<options> }) db.test5.update ({_id:1}, {y:12}) db.test6.update ({x:1}, {$set:{x:100}}, {multi:true, upsert:true})	Updating document You can also add a field that did not exist Multi : multiple updates (only partial update ($) Upsert : conditions does not match any document a new document is inserted instead works with partial and whole

$set:{}	Sets the value of a field in a document
$push:{}	Add value to an Array
$inc:{}	Increments the value of the field by the specified amount
$mul:{}	Multiplies the value of the field by the specified amount
$rename:{x : "z"}	Renames a field in a document
$unset:{}	Removes the specified field from a document.
$pull:{}	Remove an element from an array

db.test6.update ({x:1},{$set:{x:100}})
id field cannot be updated > get an error

db.test.insert ({name: "Luis", age: null })	In lowercase A string with the word null, isn't a null value. No value will also show up if age: null
db.test.find({ age: { $exists: true } }, { _id: 0 })	Show the documents that have the age field
db.test.find ({ age: {$exists: false} }, { _id: 0 })	Show the documents that do not have the age field
db.airplanes.find().sort({"passenger.name":1})	To order by (1 : ascending, -1 : descending)
db.tickets.find().limit(10)	Show only 10 doc

db.<collection>.help()	
db.countries.distinct ("field", {“condition”}) db.countries.distinct ("name", { continent: "AMERICA"}) db.countries.distinct ("name", { $and: [{continent: "AMERICA"},{population: {$gt:35}}] })	Find the the different values for a field. Find the different countries for the continent America. Find the different countries for the continent America with population greater than 35 millions
db.countries.aggregate([{ $group : { . . . } }]) db.countries.aggregate ([{ $group : { _id: "$continent", number_of_countries: {$sum: 1}, total_population: {$sum: "$population"}}}]) db.countries.aggregate([{$group : { _id: "$continent", max_population: {$max: "$population"}, avg_population: {$avg: "$population"}, min_population: {$min: "$population"} }}]) db.countries.aggregate([{$group : { _id : null, total_population:{$sum:"$population"} } }])	Aggregate Method for Grouping in MongoDB _id is null everything is just 1 group SELECT SUM(POPULATION)FROM COUNTRIES
db.countries.aggregate([{ $match : {<conditions>} } , { $group....}])	Condition can be established before doing the groups

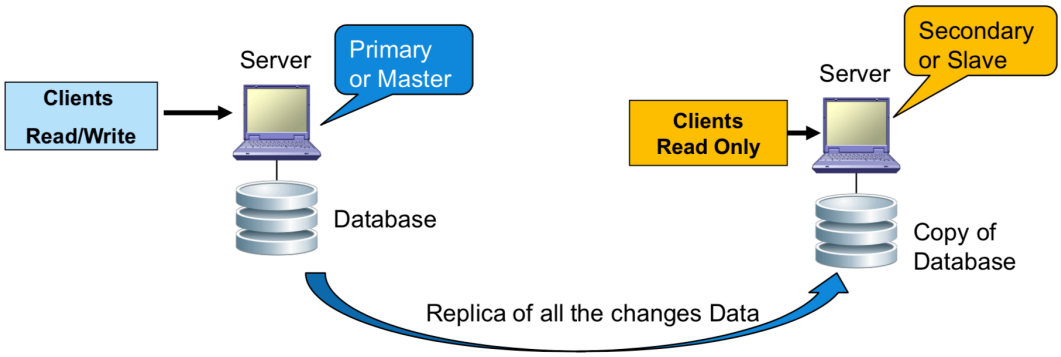
$sum	Returns a sum of numerical values. Ignores non-numeric values.
$avg	Returns an average of numerical values. Ignores non-numeric values.
$max	Returns the highest expression value for each group.
$min	Returns the lowest expression value for each group.
$first	Returns a value from the first document for each group.

$last	Returns a value from the last document for each group.
$push	Returns an array of expression values for each group. (an array with all the values of the field.)

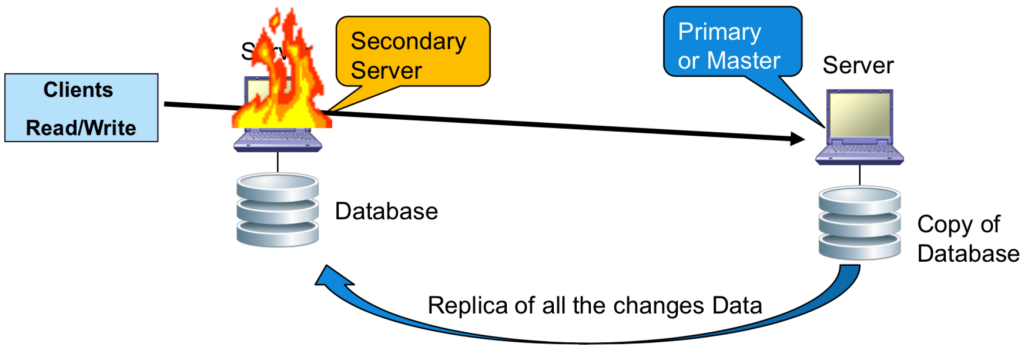
db.test.find ({phrase:/tailor/}) OR db.test.find ({phrase:{$regex:"tailor"}})	Regular Expressions
i	Case Insensitive
^	Beginning of a string
$	End of a string
\s	whitespace
\S	anything but whitespace
\d	1 digit (Find the documents that are only numbers)
\D	1 character that is not a digit (Find the documents that at least have 1 letter (or blank)
m[aeiou]	Find the String that contains ma, me, mi, mo, mu
mother|kitchen	Find the word mother or kitchen
\ db.test.find ({x:/\$/})	escape character Search for the documents containing $

Replication

- Make a **copy** (replica) of all the data in another server.
- When a **change** occur in the database the same change is propagated to the replica.
- The replication is **asynchronous** (it takes some time)



- When the Primary **fails** the secondary server becomes primary.
- Clients reconnect to secondary
- When failed Primary is back to life it can become Secondary.
- Receives the changes from the new primary server.



- Multiples slaves : MongoDB
- Every server primary: IBM Cloudant

Sharding

To increase computer power I can do it 2 ways:

- **Vertically**: Adding more CPU, Memory or disk to the same server
- **Horizontally**: Adding more Servers and dividing the documents among

<u>Column Family: HBase (columns family</u>)

Facebook uses it for messaging (Powered by HBASE)

- History
  - Google created in 2004 a new database called Big Table.
  - Google published “Big Table” paper in 2006:

        - "Bigtable: A Distributed Storage System for Structured Data".
    - HBase (and Casandra) are based in **Big Table**:
        - Manage high volume of data.
    - Hbase(and Casandra) are **open source software**.
        - Hbase (and Casandra) are **Apache Top Level Projects**: http://hbase.apache.org/

- HBase Table
    - The basic element in Hbase
    - With Columns and Rows (Similar to Relational).
    - The Hbase Table is a bit different from a relational database table:
        - Even It has rows and columns as well.
        - It has other components like new **Column-family** concept.
        - You can **not** use SQL to access it (NoSQL).
        - You can **not** do Join (only 1 table at the time).
            - Similar than with MongoDB Collections.
            - Resolve the Join your self programming.

Important HBase Concept

- **Column Family**
    - A group of columns.
    - A table can have several columns familiy.
    - A column is represented by the
        - Column family: Column name:
        - Personal_data:Name
        - Personal_data:Age
        - Internet_data:Twitter
    - In RDBMS there are no column-families.

Table Persons:

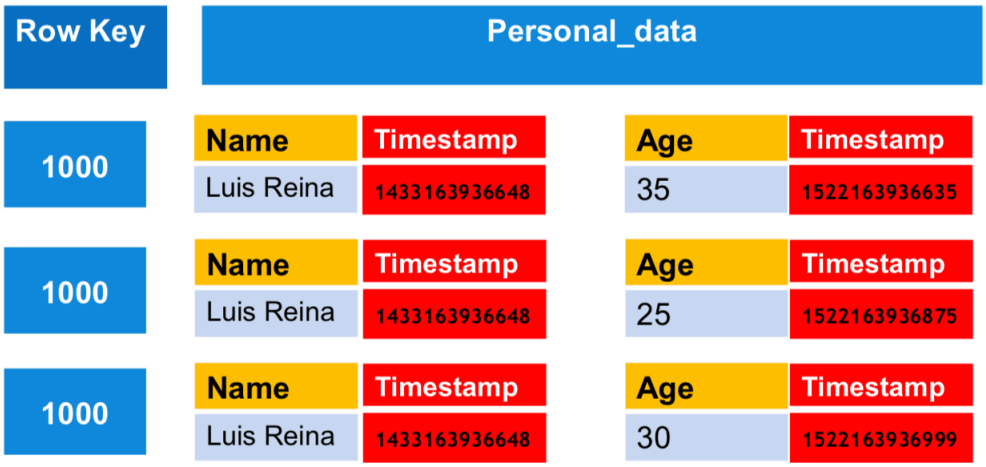
Column-family →	Personal_data			Internet_data	
Columns →	Name	Address	Age	Twitter	E-mail
Rows	John	White	25	@lg	lg@x.com
	Will	Smith	30	@wsmith	w@y.es
	Bruce	Lee	35	@lee	lee@p.tv

- **Row Key**
    - Every row in the table has a unique value that identifies one row from the others of rows.
    - Rows are stored sorted by Row Key.
    - When inserting a row the user/applications must give a value for the row key.
    - Similar to the primary key/unique key concept in RDBMS.

    - Flexible schema:

        - Columns are “created” when inserting data in a row.
        - When creating a table you only give the name of the table and the column families names, not the columns.
        - Column names can be the same in different column families.
        - **Null values does not exist** => nothing is stored for a missing column (“sparse data”).

- **Cell and Timestamp**
    - Timestamp=Time when the value was inserted or modified.
    - Everytime the value changes the timestamp is inserted.
        - It is inserted by Hbase but you can insert instead.
    - **Versioning**
        - Keeping past values of the data: When application do an update I keep the old value before the update.
        - By default Hbase keeps only 1 version but it can be customized as you wish.
        - In older versions of Hbase (until v0,96) default was 3 values.
        - When quering data if not specified I will obtain the last value but I can ask for a past value.
        - The timestamp value mentioned below is the one used for versioning:
            - The highest timestamp is the last value.
            - Using this timestamp I can obtain a past value.
        - The timestamp is a long integer in UTC format: Number of milliseconds since midnight, January1, 1970 UTC

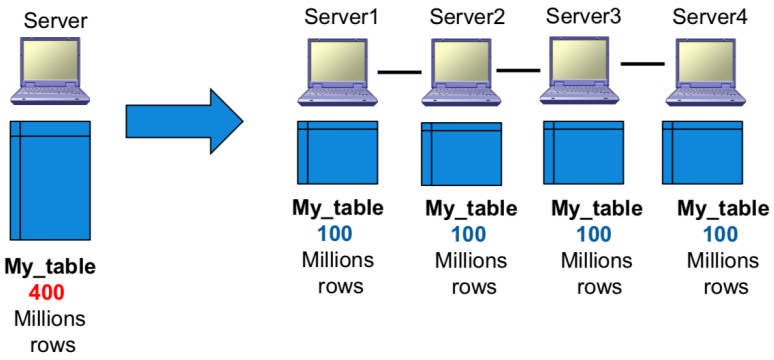


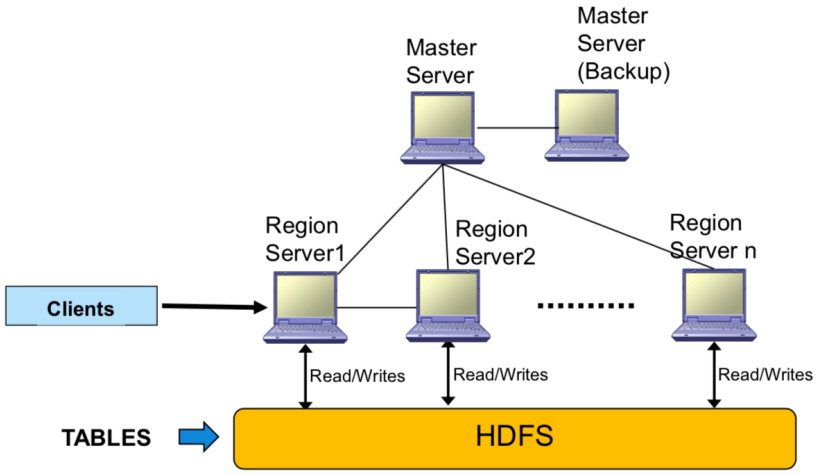
HBase Architecture

- HBase uses HDFS from Hadoop: Data in HBASE is **stored** in **HDFS**.
    - The standalone allows to run out of HDFS but is only for development never for a production system.
- This way data is **replicated** among different servers
    - In MongoDB I need to specify the replicas.
    - In Hbase is done by HDFS **automatically**: Thanks to HDFS redundancy of the blocks.
- Because of this Hbase is recognized as the **Hadoop database**.

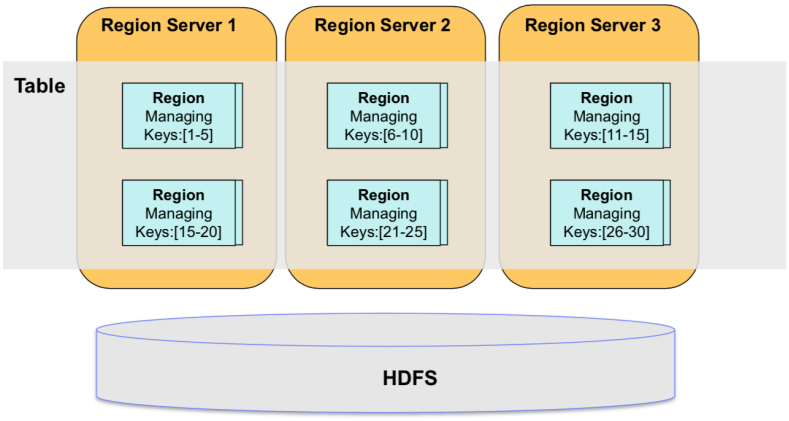
- It does not use **Map/Reduce**.
- Map/Reduce: Batch reads (slow for queries accesing little data).
    - Quick random Access (Real Time) to the data (read and writes).

- **HBase Sharding**
    - As mentioned sharding allows to split the data among different servers to scale horizontally (adding servers):
        - In MongoDB we Split the **json documents** among the different servers
        - In Hbase we splits the **rows** among the servers.



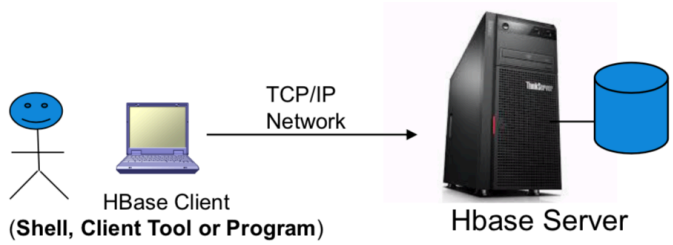


- Region Servers and Regions
    - HBase Cluster is a number of Region Servers.
    - Every Region Server has 1 or more Regions.
    - Data is divided (sharded) in the different Regions.



HBASE SHELL

- The Hbase Shell is a client where we can write the HBase commands to interact with an Hbase Server.
    - Remember the Client/server Architecture:



- use the AWS Cluster where Hbase is installed.

CREATING HBASE TABLES

- TO OPEN THE SHELL:
    - Open a ssh connection to the Linux Server (Edge node 1) Connect to the Linux Edge Node using ssh client.
    - Execute in Linux this command to open the HBase Shell
        - hbase shell

create'Table_name','Column_family1','Column_family2' or create'Table_name',[{NAME => 'Column_family1'},{NAME=>'Column _family2'}]	creating a hbase table
list	List all tables
exists'Table_name'	see if a table exists
Alter'Table_name','Column _family1'	Add a New Column Family
Disable'Table_name' Drop'Table_Name'	Dropping a Column
Describe'Table_name'	Show the "structure" and Information of a Table
Alter'Table_name',{NAME=>'Column _family1' ,VERSIONS=>3	Changing the number of versions for a Column Family

HBASE NAMESPACE

- logical way of grouping tables:
    - Similar to Relational Schema.
    - A way to group tables together
    - For Namespace=jonas
        - jonas:table1
        - jonas:table2
    - For Namespace=John

- John:table1

➔ Same Table name in a different Namespace

creating_namespace 'Jonas'	create the Namespace first
create 'namespace : table_name' ,'column_families' create'Luis:Table1','col_fam1'	Then create the table under the Namespace
List_namespace	Listing namespaces There are 2 predefined namespaces: ✓ hbase: system namespace, used to contain HBase internal tables. ✓ default: tables created with no explicit specified namespace.
Drop_namespace'namespace' Drop_namespace'Jonas'	Dropping a namespace ✓ You can drop a namespace if it doesn't contain any table in it. You must first drop all tables and then the namespace.

INSERT DATA INTO HBASE TABLES

put 'table_name', 'row_key, 'column_family:column', 'value' put 'my_table', 1, 'cf1:c1', 'hello'	insert data
put 'table_reina', 1, 'private:name' ,'Luis Reina' put 'table_reina', 1, 'private:age' , 30 put 'table_reina', 1, 'public:email', 'lreina@faculty.ie.edu'	Row key 1 column name value your name and age column your age both columns in column familty private and column email with value your_email in column family public
put 'table_reina', 2, 'private:name' ,'Griezmann' put 'table_reina', 2, 'private:nationality' , 'french' put 'table_reina', 2, 'public:email', 'jsmith@faculty.ie.edu'	Row key 2 column name value your_friend and nationality column his nationality in columna family private and column email with value his_email in column family public

QUERYING DATA IN HBASE

scan ‘my_table’	showing all the rows of a table
scan ‘my_table ', {COLUMN => 'cf1' }	Specifying the colum family required, option COLUMN:
scan ‘my_table ', {COLUMN => 'cf1:c1' }	Specifying the column required, option COLUMN
scan ‘my_table ', {COLUMNS => ['cf1','cf2'] }	Specifying **several** column families required, option COLUMNS
scan ‘my_table ', {COLUMNS => ['cf1:c1','cf2:c1'] }	Specifying several columns required, option COLUMNS
scan ‘my_table ', {COLUMNS => ['cf1:c1','cf2'] }	Specifying a mix of columns and column families, option COLUMNS:
scan ‘my_table ‘, {LIMIT => 10}	**Limiting** the Number of Rows Returned, option LIMIT:
scan ‘my_table ‘, {STARTROW => '2'}	Identifiying the **first** row to return, option STARTROW
scan ‘my_table ‘, { COLUMNS => ['cf1:c1','cf2:c1’], LIMIT => 1 , STARTROW => '2'}	**Multiple** Options together (separate by commas)
get ‘<table_name>’ , ‘<row_key> get ‘<table_name>’ , ‘ <row_key>’, ‘<column families or columns>’ get 'my_table', 1, {COLUMNS => 'cf1:c10'}	get command to read 1 row only Specifing the column families or columns I need Some **options** are also valid for get
scan 'table_reina', {LIMIT=>1, STARTROW=>’2’} OR get 'table_reina', '2’	Read the second row

UPDATING DATA IN HBASE

No specific command for updating a value.
As commented Hbase does versioning:
– Using the Timestamp when the data was inserted.
– Hbase will keep the old values of the data: if we specify versions bigger tan 1 when creating the table.

Put ‘my_table’, 1, ‘cf1:c1’, ‘hello’ put ‘my_table’, 1, ‘cf1:c1’, ‘new value’	same put command is used with same row key

RETRIEVING PAST DATA

create 'my_table', {NAME=> 'cf1',VERSIONS=>3}	The column family must have VERSIONING enabled if it is disabled by default
alter ‘my_table’,NAME => ‘cf1’, VERSIONS => 3	The column family must have VERSIONING enabled if it is disabled by default
get 'my_table', 1,{TIMESTAMP=> 1433701796440}	There is an option TIMESTAMP
put 'my_table', 1, 'cf1:c1', 'hello'	timestamp1
put 'my_table', 1, 'cf1:c1', 'new value'	timestamp2
get 'my_table',1,{TIMESTAMP=> timestamp1}	ask for past data specfing a past timestamp
scan 'my_table',{VERSIONS=>3}	Show all the versions
scan 'my_table', {COLUMNS=>'cf1:c1', TIMERANGE=>[1390651029025,1490651031644]}	Show the newest version for a Time Range
scan 'my_table', {COLUMNS=>'cf1:c1', TIMERANGE=>[1390651029025,1490651031644], VERSIONS=>2}	Show 2 versions for a Time Range