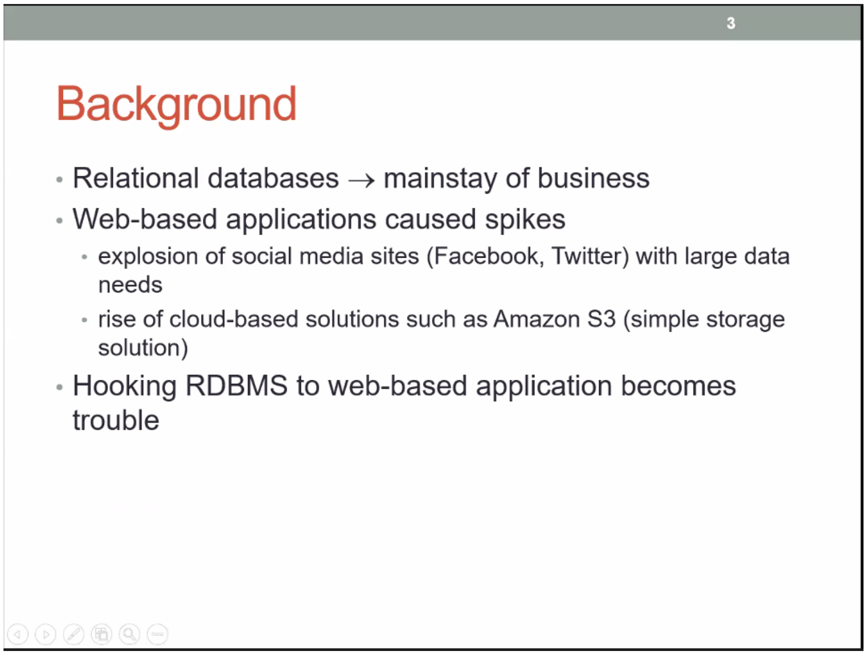
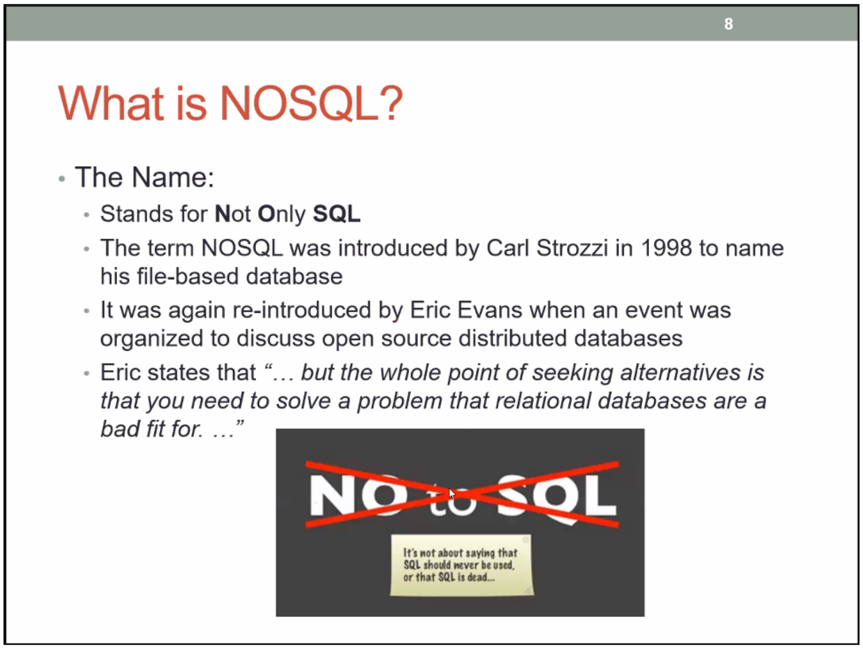
→



Background

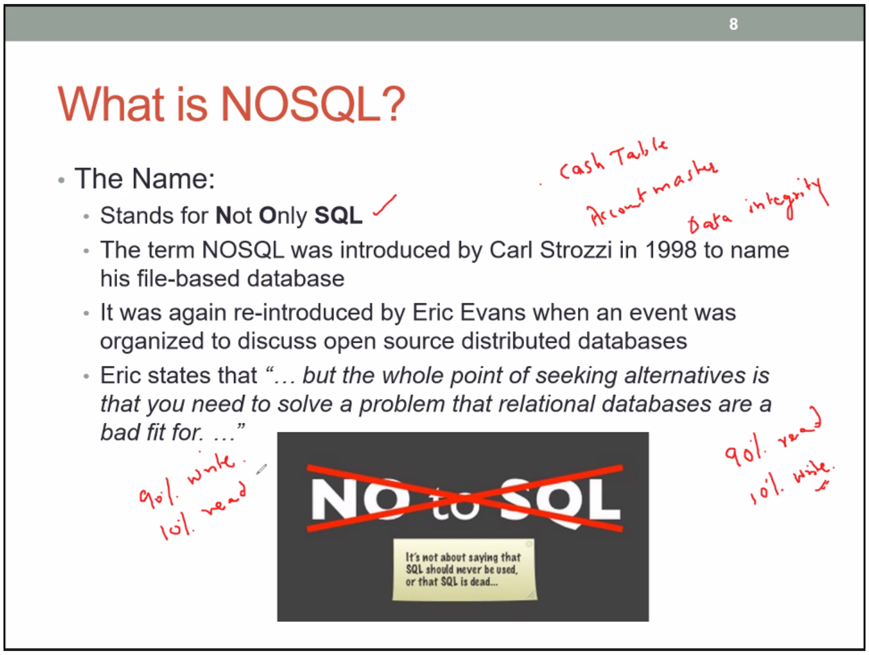
- Relational databases → mainstay of business
- Web-based applications caused spikes
  - explosion of social media sites (Facebook, Twitter) with large data needs
  - rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Hooking RDBMS to web-based application becomes trouble



What is NOSQL?

- The Name:
  - Stands for **N**ot **O**nly **SQL**
  - The term NOSQL was introduced by Carl Strozzi in 1998 to name his file-based database
  - It was again re-introduced by Eric Evans when an event was organized to discuss open source distributed databases
  - Eric states that *"… but the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for. …"*

~~NO to SQL~~
It's not about saying that SQL should never be used, or that SQL is dead...



What is NOSQL?

- The Name:
  - Stands for **N**ot **O**nly **SQL** ✓
  - The term NOSQL was introduced by Carl Strozzi in 1998 to name his file-based database
  - It was again re-introduced by Eric Evans when an event was organized to discuss open source distributed databases
  - Eric states that *"… but the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for. …"*

Cash Table

Account master

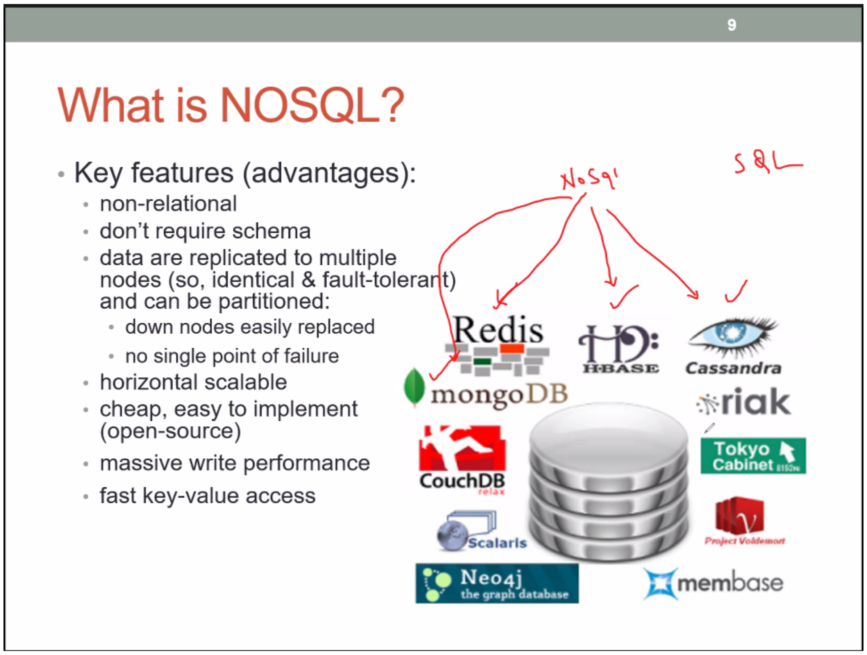
Data integrity

90% write.
10% read

90% read
10% write.

~~NO to SQL~~
It's not about saying that SQL should never be used, or that SQL is dead...

a. It more write, less read, and you want to maintain data integrity, use SQL [like in banking]
b. If more read, less write, you want speed, and there is no certainty about how many users will connect , you use NOSQL [like in e-commerce]



What is NOSQL?

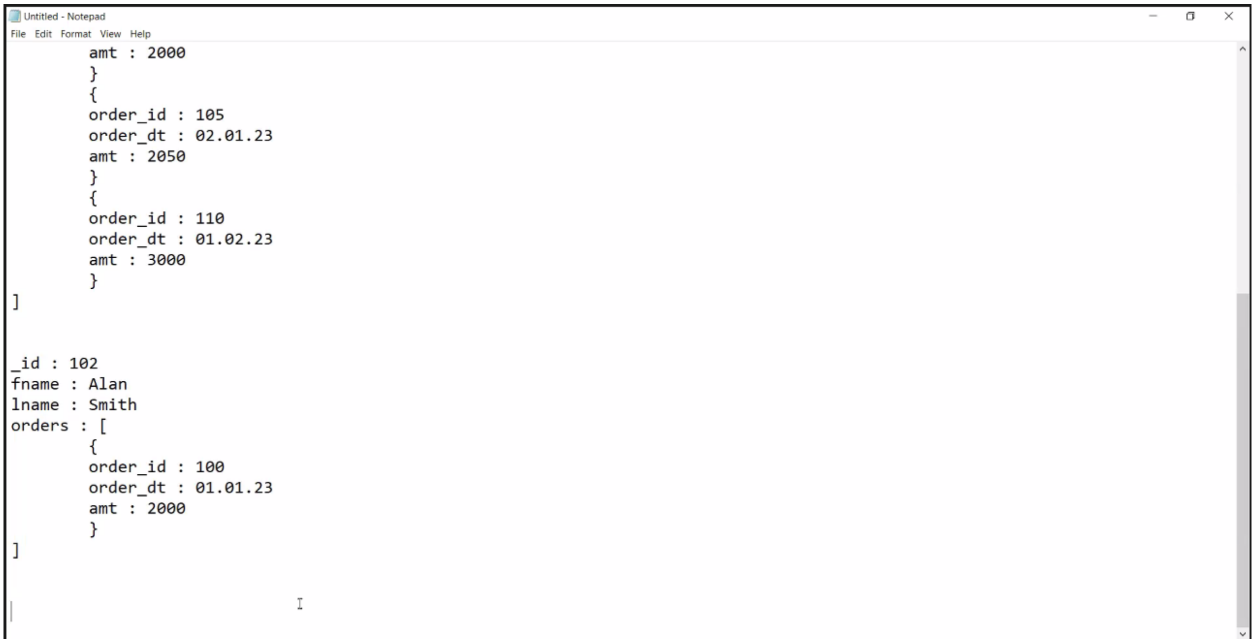
- Key features (advantages):
  - non-relational
  - don't require schema
  - data are replicated to multiple nodes (so, identical & fault-tolerant) and can be partitioned:
    - down nodes easily replaced
    - no single point of failure
  - horizontal scalable
  - cheap, easy to implement (open-source)
  - massive write performance
  - fast key-value access

→ Types of NOSQL

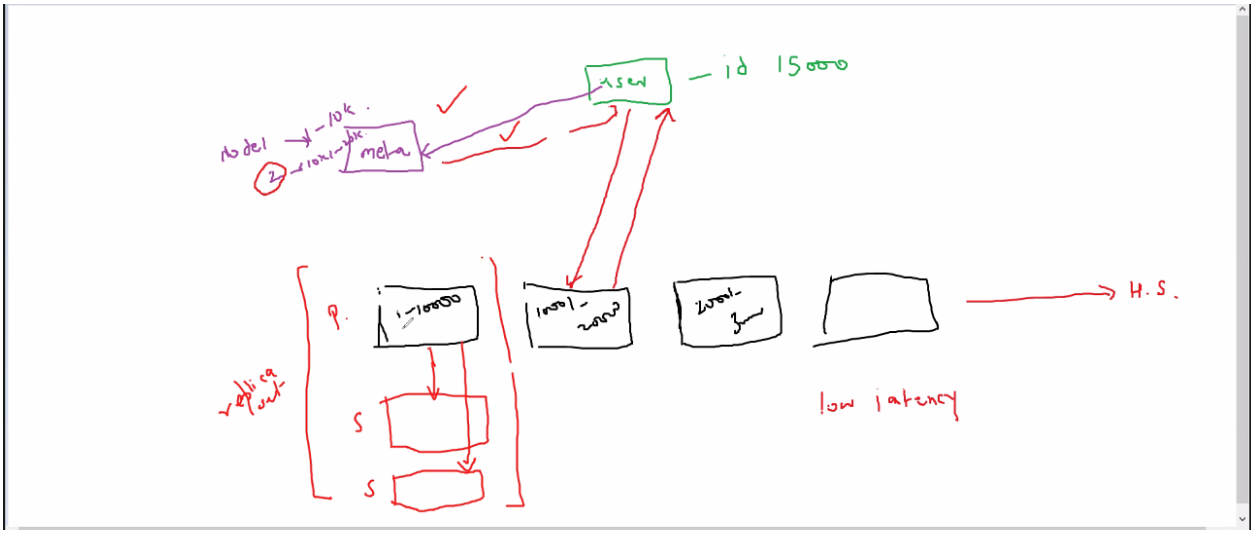
a.	Key-value pair	Amazon(DynamoDB)
b.	Columnar	Google’s, BigTable, Cassandra, HBase
c.	Document	MongoDB, CouchDB
d.	Graph-based	Neo4J

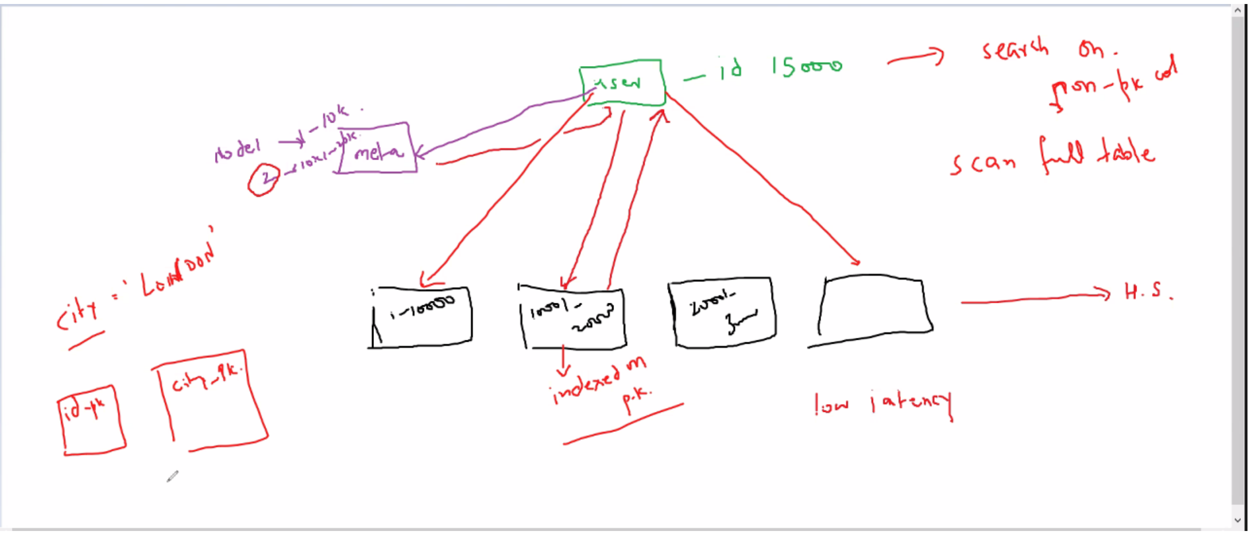
→

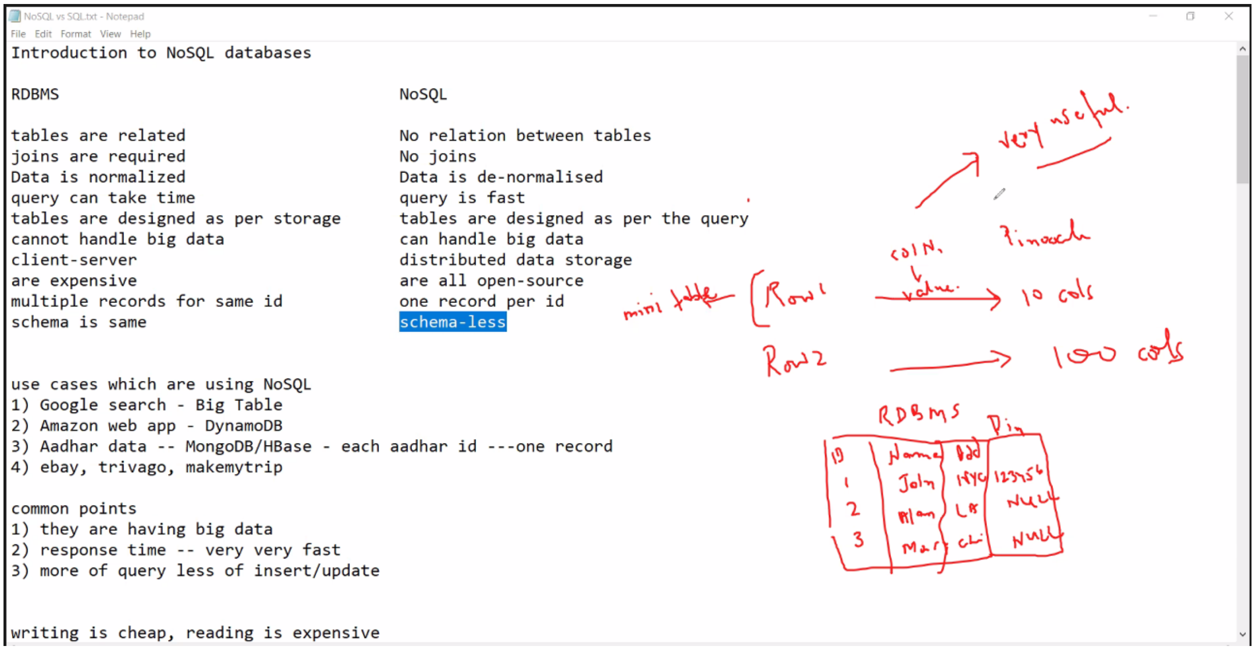
a. In MongoDB, table is called collection, row is called document
b. mongoDB follows node replication



```
        amt : 2000
        }
        {
        order_id : 105
        order_dt : 02.01.23
        amt : 2050
        }
        {
        order_id : 110
        order_dt : 01.02.23
        amt : 3000
        }
]


_id : 102
fname : Alan
lname : Smith
orders : [
        {
        order_id : 100
        order_dt : 01.01.23
        amt : 2000
        }
]
```


user
— id 15000
node1 → 1-10k.
2 — 10k1-20k.
meta
P.
1-10000
1001-2000
2001-3000
replica set
S
S
H.S.
low latency


user
— id 15000
→ search on.
non-pk col
scan full table
node1 → 1-10k.
2 — 10k1-20k.
meta
city = 'London'
1-10000
1001-2000
2001-3000
H.S.
id-pk
city-pk.
indexed on p.k.
low latency



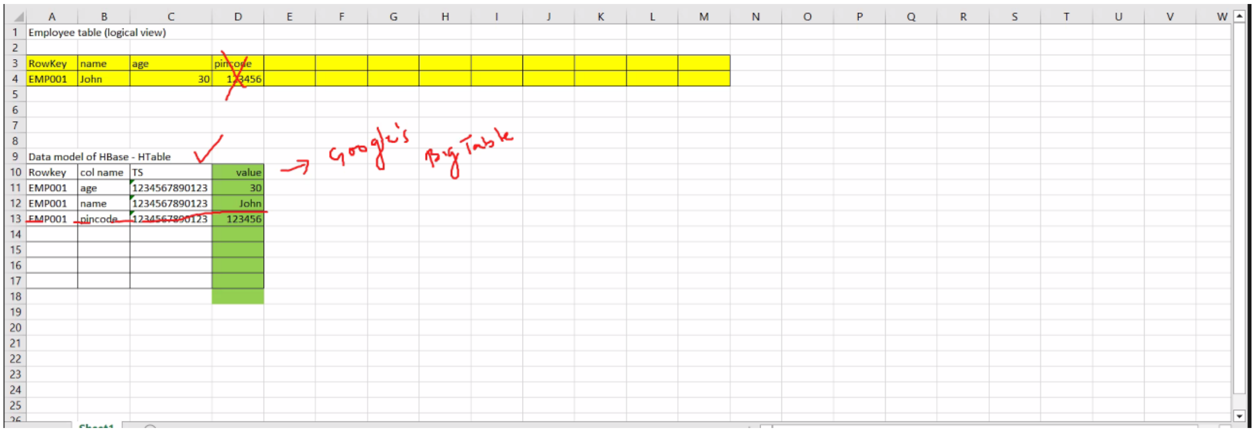
```
Introduction to NoSQL databases

RDBMS                                   NoSQL

tables are related                      No relation between tables
joins are required                      No joins
Data is normalized                      Data is de-normalised
query can take time                     query is fast
tables are designed as per storage      tables are designed as per the query
cannot handle big data                  can handle big data
client-server                           distributed data storage
are expensive                           are all open-source
multiple records for same id            one record per id
schema is same                          schema-less


use cases which are using NoSQL
1) Google search - Big Table
2) Amazon web app - DynamoDB
3) Aadhar data -- MongoDB/HBase - each aadhar id ---one record
4) ebay, trivago, makemytrip

common points
1) they are having big data
2) response time -- very very fast
3) more of query less of insert/update


writing is cheap, reading is expensive
```

ID	Name	Add	Pin
1	John	NYC	123456
2	Alan	LA	NULL
3	Mar	Chi	NULL

→ writing is cheap because user can wait for writing, but reading is expensive because user might leave as he has to wait while app loads due to slow reading of data from server

→ HBase

a. Is based on the idea of Google’s Big Table
b. Built on top of HDFS, which stores HTable [MongoDB & Cassandra have independent frameworks, not dependent on HDFS]
c. HTable is divided into regions, regions are maintained by Region servers
d. Regions are logically splits, but blocks are physical splits
e. HMaster: stores all meta-data
f. Has columnar structure
g. Data can grow bigger, while it has more rows
h. Has Four column structure:
    i. RowKey
    ii. Column Name
    iii. TimeStamp
    iv. Value
i. Indexed in RowKey (if row key is same, then indexed on col name)
j.



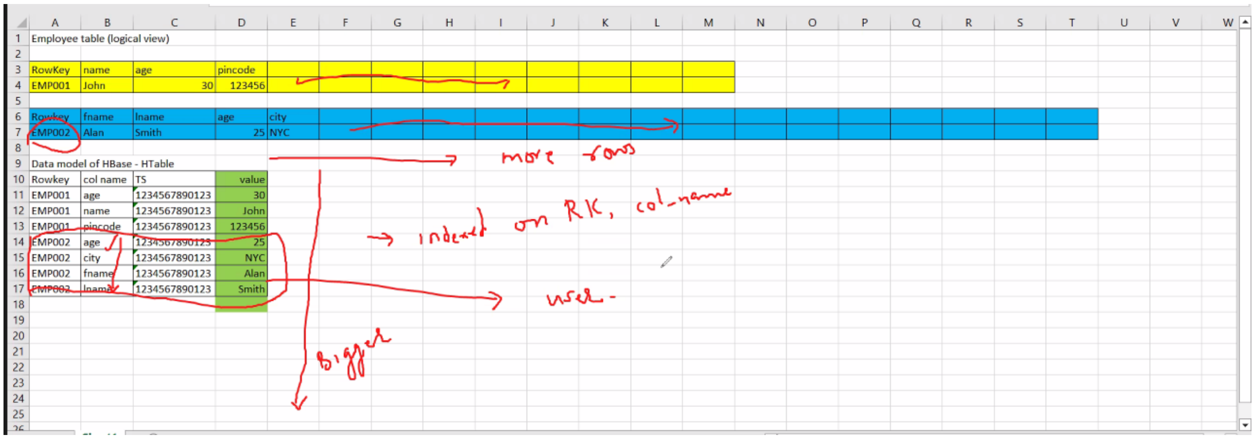
Employee table (logical view)

RowKey	name	age	pincode
EMP001	John	30	123456

Data model of HBase - HTable

Rowkey	col name	TS	value
EMP001	age	1234567890123	30
EMP001	name	1234567890123	John
EMP001	pincode	1234567890123	123456

→ Google's BigTable



Employee table (logical view)

RowKey	name	age	pincode
EMP001	John	30	123456

Rowkey	fname	lname	age	city
EMP002	Alan	Smith	25	NYC

Data model of HBase - HTable

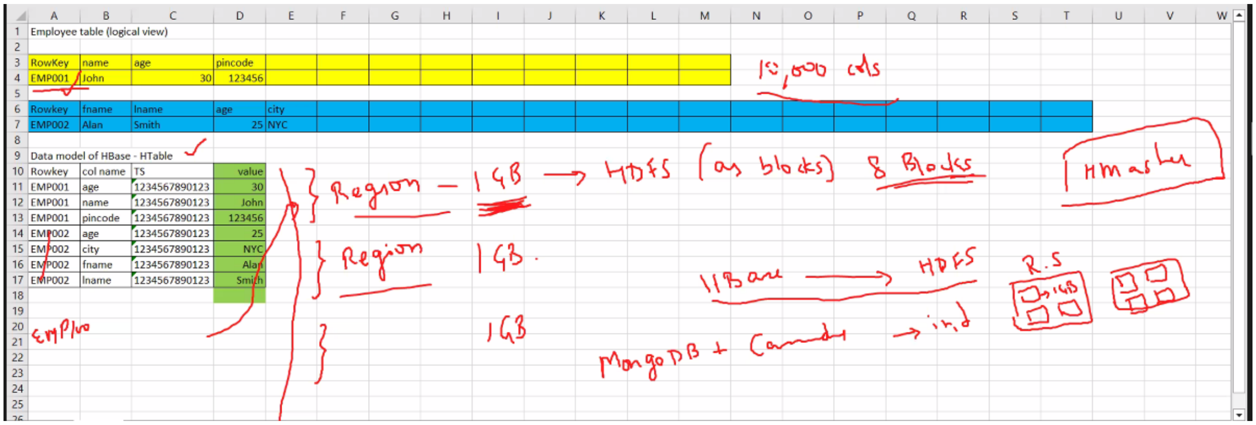
Rowkey	col name	TS	value
EMP001	age	1234567890123	30
EMP001	name	1234567890123	John
EMP001	pincode	1234567890123	123456
EMP002	age	1234567890123	25
EMP002	city	1234567890123	NYC
EMP002	fname	1234567890123	Alan
EMP002	lname	1234567890123	Smith

more rows

→ indexed on RK, col_name

user-

Bigger



Employee table (logical view)

RowKey	name	age	pincode
EMP001	John	30	123456

Rowkey	fname	lname	age	city
EMP002	Alan	Smith	25	NYC

Data model of HBase - HTable

Rowkey	col name	TS	value
EMP001	age	1234567890123	30
EMP001	name	1234567890123	John
EMP001	pincode	1234567890123	123456
EMP002	age	1234567890123	25
EMP002	city	1234567890123	NYC
EMP002	fname	1234567890123	Alan
EMP002	lname	1234567890123	Smith

10,000 cols

} Region — 1GB → HDFS (as blocks) 8 Blocks

HMaster

} Region 1GB.

} 1GB

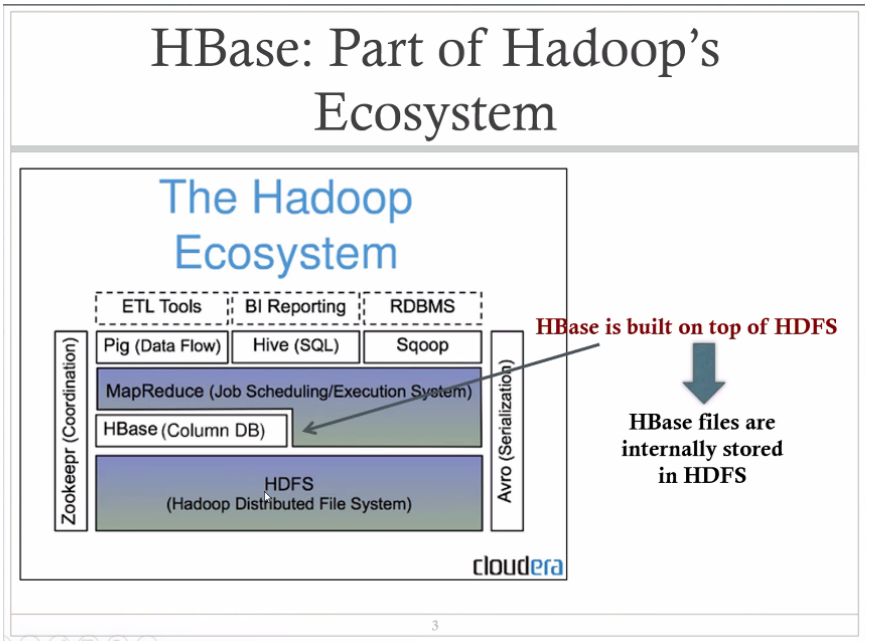
EMP/10

HBase → HDFS

R.S

1GB

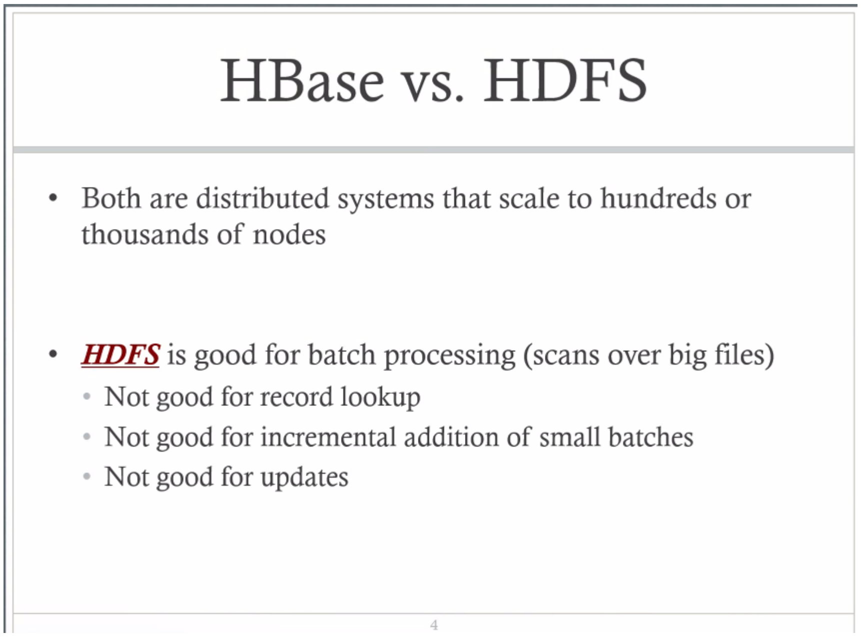
MongoDB + Cassandra → ind



HBase: Part of Hadoop's Ecosystem

HBase is built on top of HDFS

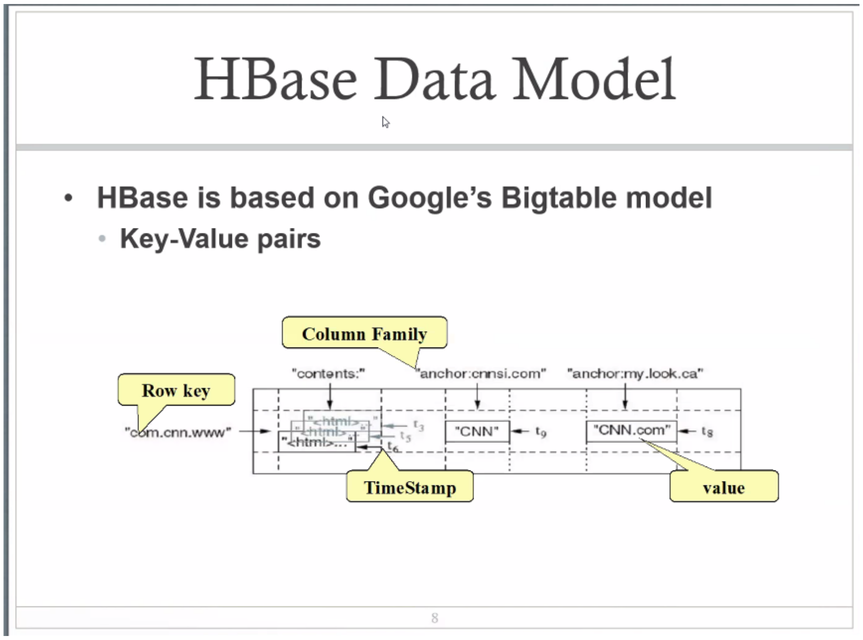
HBase files are internally stored in HDFS



HBase vs. HDFS

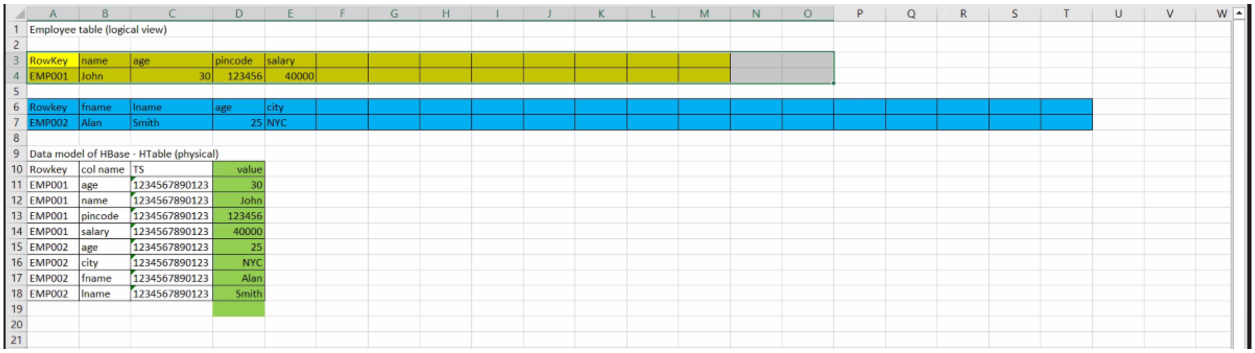
- Both are distributed systems that scale to hundreds or thousands of nodes

- ***HDFS*** is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates



HBase Data Model

- HBase is based on Google's Bigtable model
  - Key-Value pairs



Employee table (logical view)

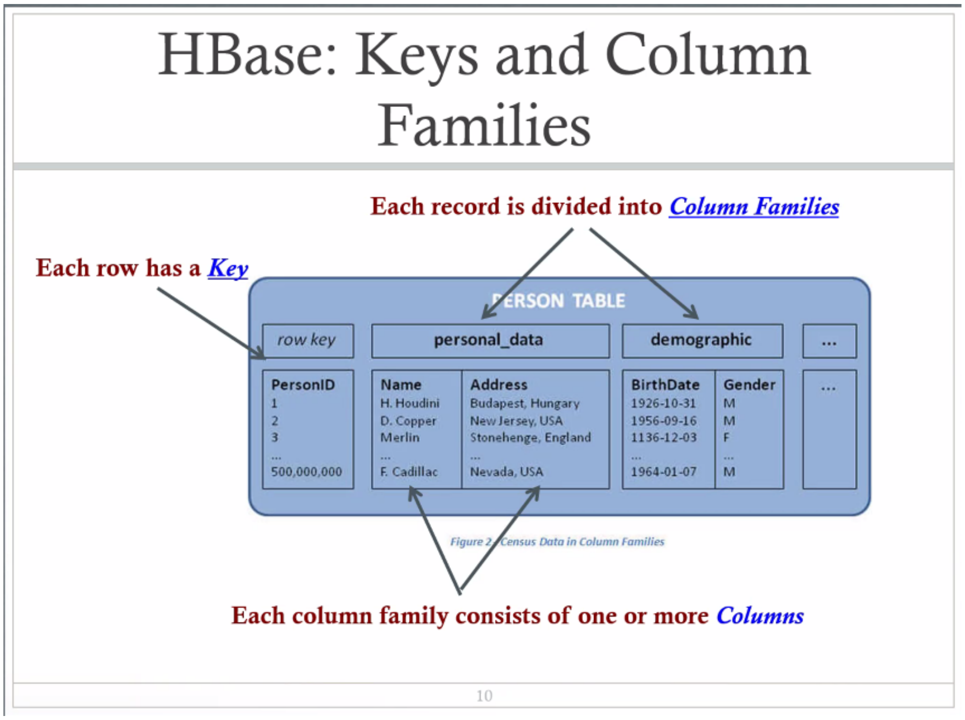
RowKey	name	age	pincode	salary
EMP001	John	30	123456	40000

Rowkey	fname	lname	age	city
EMP002	Alan	Smith	25	NYC

Data model of HBase - HTable (physical)

Rowkey	col name	TS	value
EMP001	age	1234567890123	30
EMP001	name	1234567890123	John
EMP001	pincode	1234567890123	123456
EMP001	salary	1234567890123	40000
EMP002	age	1234567890123	25
EMP002	city	1234567890123	NYC
EMP002	fname	1234567890123	Alan
EMP002	lname	1234567890123	Smith

→ can have record with same row key, and column name, but it'll have different time stamp, where latest timestamp will be prioritized
→ indexing is on row key, but it row key is also same, indexing is on column name, but if column name is also same, indexing will take place on descending order of TimeStamp



HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*

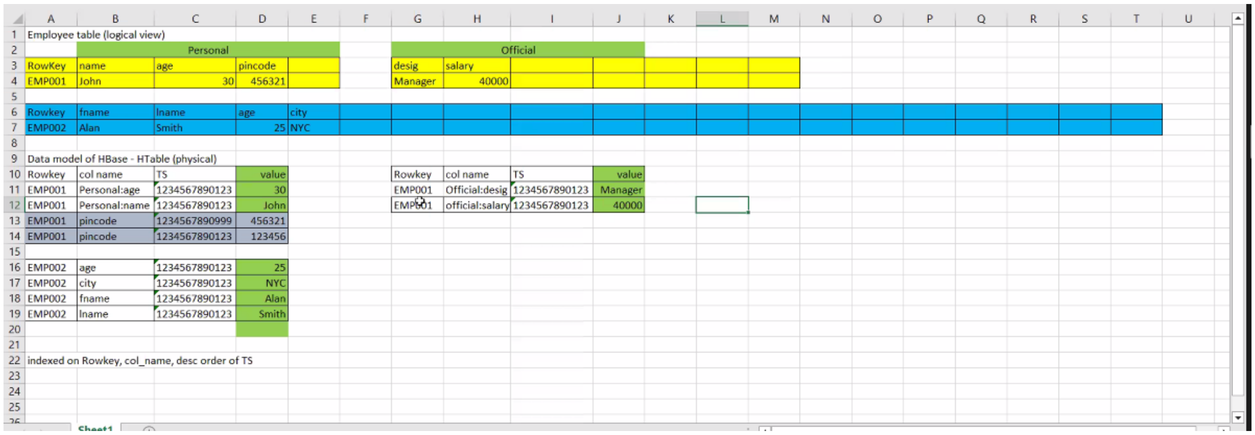
PERSON TABLE

row key	personal_data		demographic		...
PersonID	Name	Address	BirthDate	Gender	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2. Census Data in Column Families

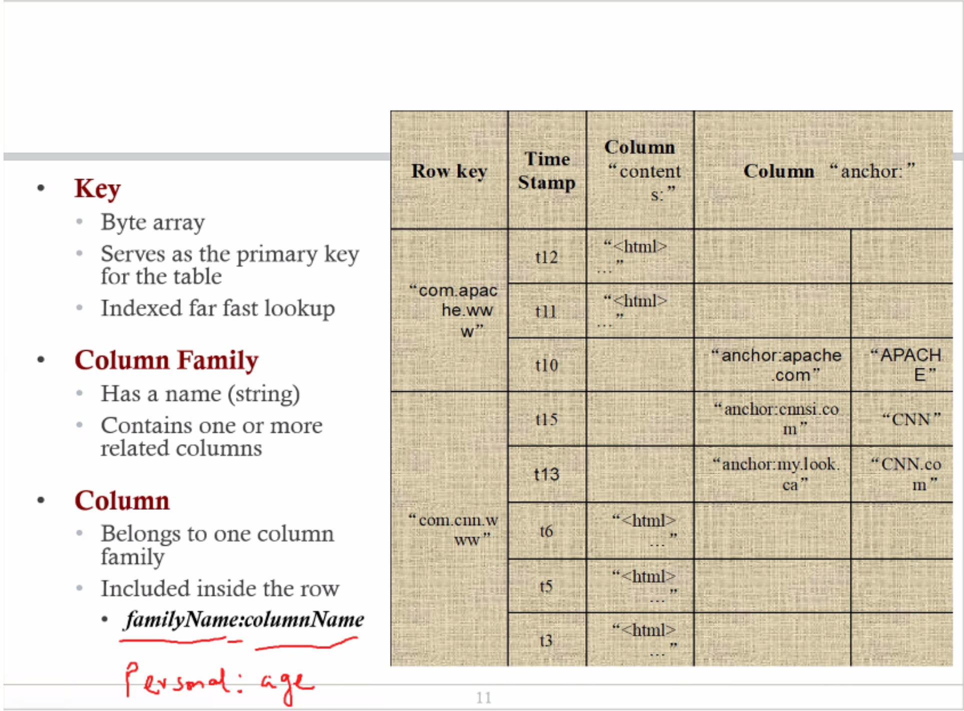
Each column family consists of one or more *Columns*

In case of column family, column is referred as <column family>:<column name>
E.g. Personal:age , Personal:pincode, Official:salary, Official:Designation



	A	B	C	D	E	F	G	H	I	J
1	Employee table (logical view)									
2		Personal					Official			
3	RowKey	name	age	pincode			desig	salary		
4	EMP001	John	30	456321			Manager	40000		
5										
6	Rowkey	fname	lname	age	city					
7	EMP002	Alan	Smith	25	NYC					
8										
9	Data model of HBase - HTable (physical)									
10	Rowkey	col name	TS	value			Rowkey	col name	TS	value
11	EMP001	Personal:age	1234567890123	30			EMP001	Official:desig	1234567890123	Manager
12	EMP001	Personal:name	1234567890123	John			EMP001	official:salary	1234567890123	40000
13	EMP001	pincode	1234567890999	456321						
14	EMP001	pincode	1234567890123	123456						
15										
16	EMP002	age	1234567890123	25						
17	EMP002	city	1234567890123	NYC						
18	EMP002	fname	1234567890123	Alan						
19	EMP002	lname	1234567890123	Smith						
20										
21										
22	indexed on Rowkey, col_name, desc order of TS									
23										
24										
25										

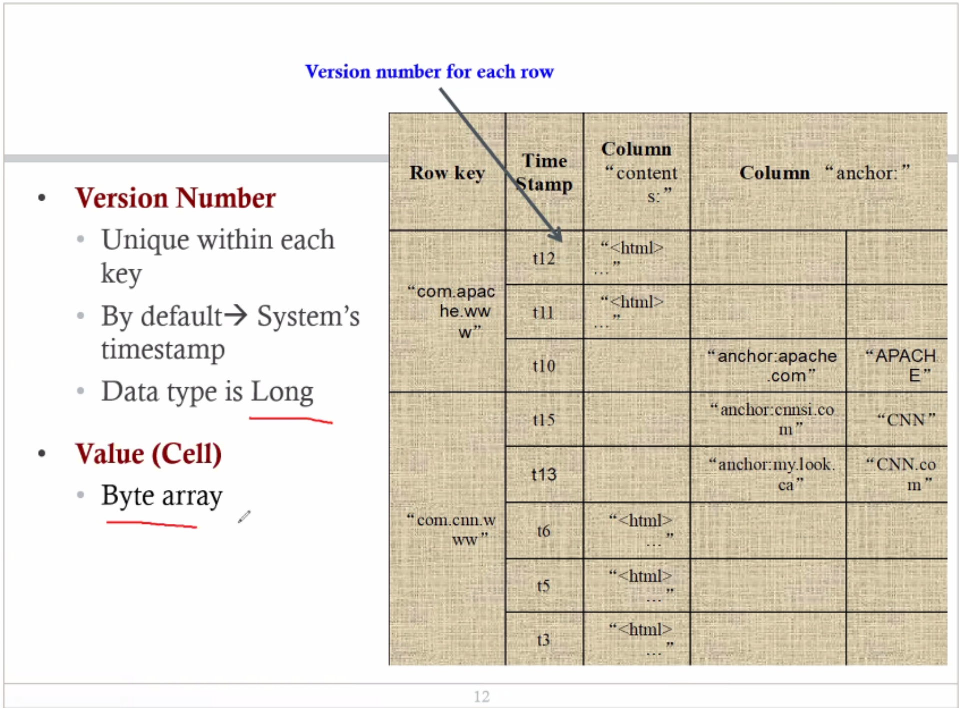
→ Has only one dataType “ByteArray” for both Key & value



- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - *familyName:columnName*

Personal: age

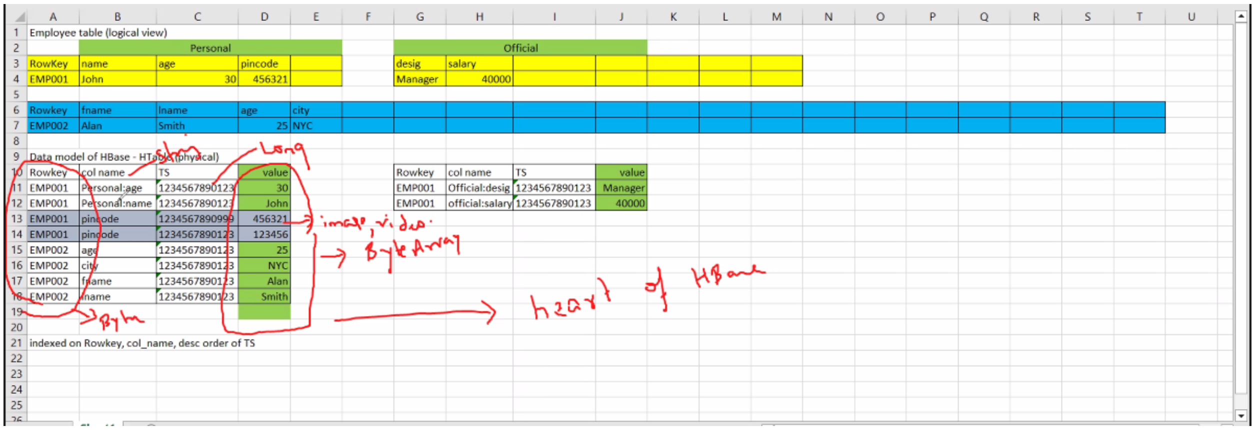
Row key	Time Stamp	Column “contents:”	Column “anchor:”	
“com.apache.www”	t12	“<html> …”		
	t11	“<html> …”		
	t10		“anchor:apache.com”	“APACHE”
“com.cnn.www”	t15		“anchor:cnnsi.com”	“CNN”
	t13		“anchor:my.look.ca”	“CNN.com”
	t6	“<html> …”		
	t5	“<html> …”		
	t3	“<html> …”		



Version number for each row

- **Version Number**
  - Unique within each key
  - By default→ System’s timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array

Row key	Time Stamp	Column “contents:”	Column “anchor:”	
“com.apache.www”	t12	“<html> …”		
	t11	“<html> …”		
	t10		“anchor:apache.com”	“APACHE”
“com.cnn.www”	t15		“anchor:cnnsi.com”	“CNN”
	t13		“anchor:my.look.ca”	“CNN.com”
	t6	“<html> …”		
	t5	“<html> …”		
	t3	“<html> …”		



Employee table (logical view)

	Personal					Official	
RowKey	name	age	pincode			desig	salary
EMP001	John	30	456321			Manager	40000

Rowkey	fname	lname	age	city
EMP002	Alan	Smith	25	NYC

Data model of HBase - HTable (physical)

Rowkey	col name	TS	value
EMP001	Personal:age	1234567890123	30
EMP001	Personal:name	1234567890123	John
EMP001	pincode	1234567890999	456321
EMP001	pincode	1234567890123	123456
EMP002	age	1234567890123	25
EMP002	city	1234567890123	NYC
EMP002	fname	1234567890123	Alan
EMP002	lname	1234567890123	Smith

Rowkey	col name	TS	value
EMP001	Official:desig	1234567890123	Manager
EMP001	official:salary	1234567890123	40000

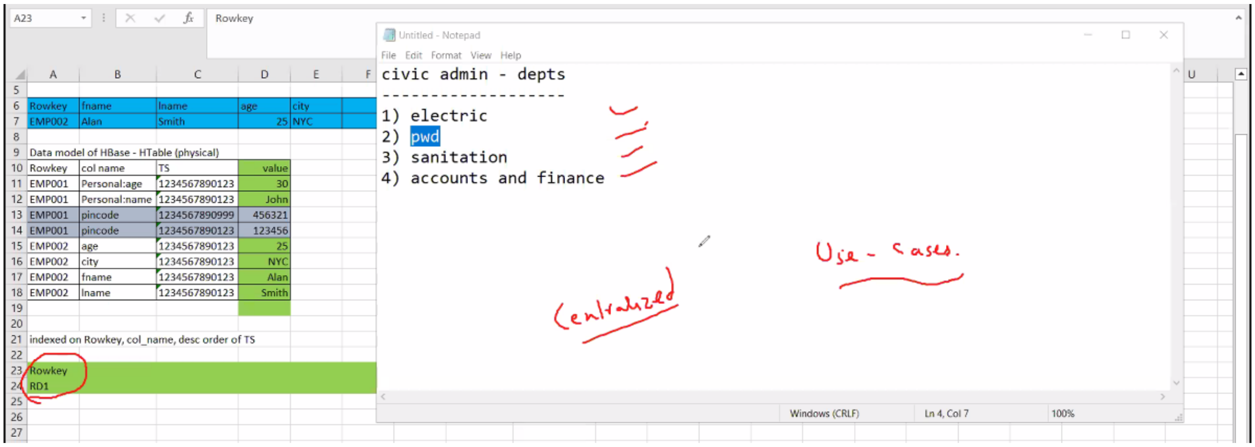
indexed on Rowkey, col_name, desc order of TS

→ Use Cases of HBase

a. Civic administration:

   i. Has departments viz. Electric, PWD, Sanitation, Accounts

   ii. Suppose , we have some roads in a city associated with different departments, and if one department is going to dig up some road to do some maintenance,it can make entry in table to communicate with other departments, so that road is not digged multiple times, and it could save some budget and time



Rowkey	fname	lname	age	city
EMP002	Alan	Smith	25	NYC

Data model of HBase - HTable (physical)

Rowkey	col name	TS	value
EMP001	Personal:age	1234567890123	30
EMP001	Personal:name	1234567890123	John
EMP001	pincode	1234567890999	456321
EMP001	pincode	1234567890123	123456
EMP002	age	1234567890123	25
EMP002	city	1234567890123	NYC
EMP002	fname	1234567890123	Alan
EMP002	lname	1234567890123	Smith

indexed on Rowkey, col_name, desc order of TS

Rowkey
RD1

```
civic admin - depts
-------------------
1) electric
2) pwd
3) sanitation
4) accounts and finance
```

b.

→