Syllabus

CONCEPTUAL DATA MODELING

Database environment – Database system development lifecycle – Requirements collection –Database design - Entity-Relationship model – Enhanced-ER model – UML class diagrams.

RELATIONAL MODEL AND SQL

Relational model concepts - Integrity constraints - SQL Data manipulation – SQL Data definition – Views - SQL programming.

RELATIONAL DATABASE DESIGN AND NORMALIZATION

ER and EER-to-Relational mapping – Update anomalies – Functional dependencies – Inference rules – Minimal cover – Properties of relational decomposition – Normalization (upto BCNF).

TRANSACTIONS

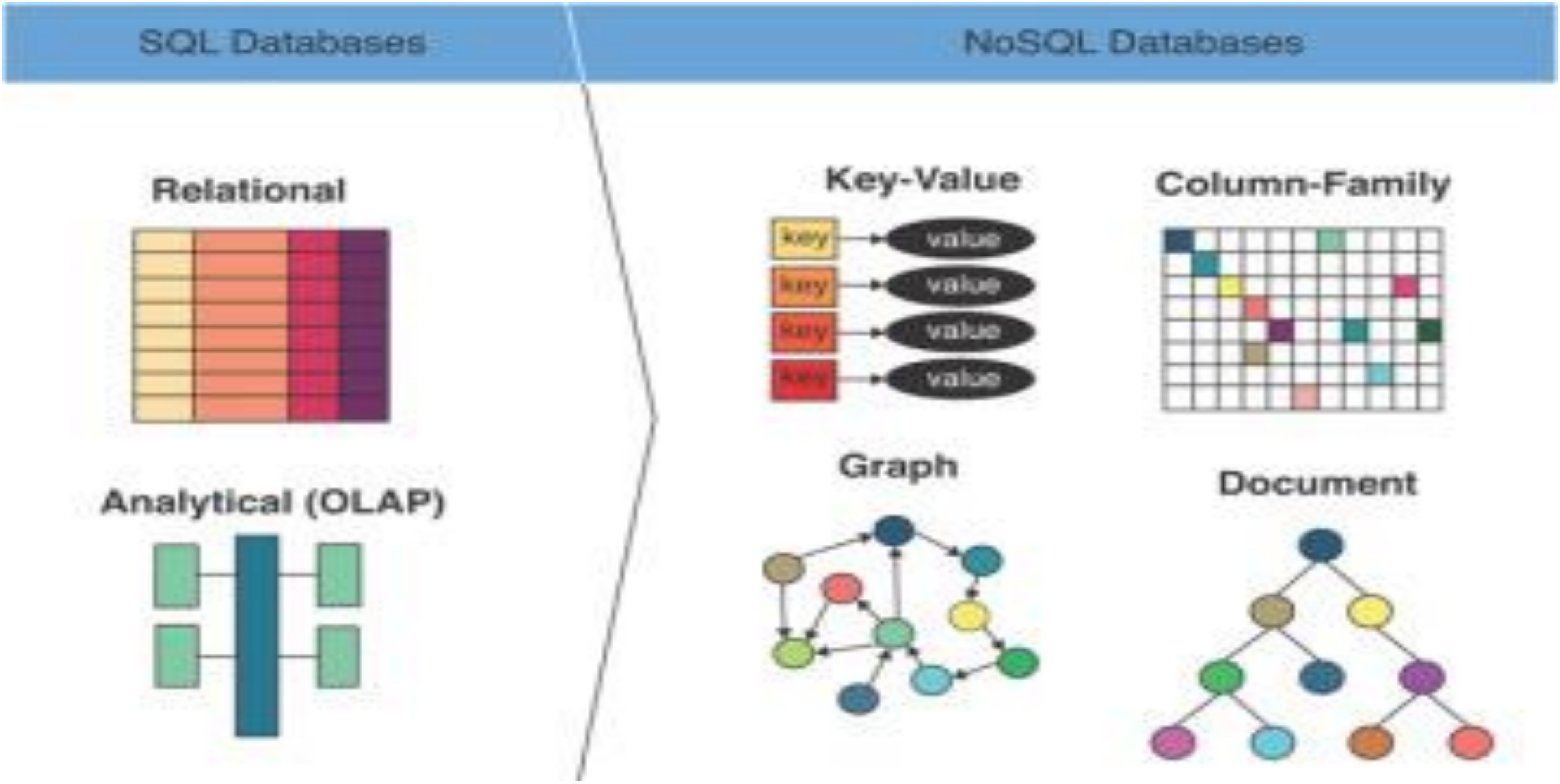
Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

NO-SQL DATABASES

No-SQL: CAP theorem – Document-based: MongoDB data model and CRUD operations; Column-based: Hbase data model and CRUD operations.

Unit V **NO-SQL DATABASES**

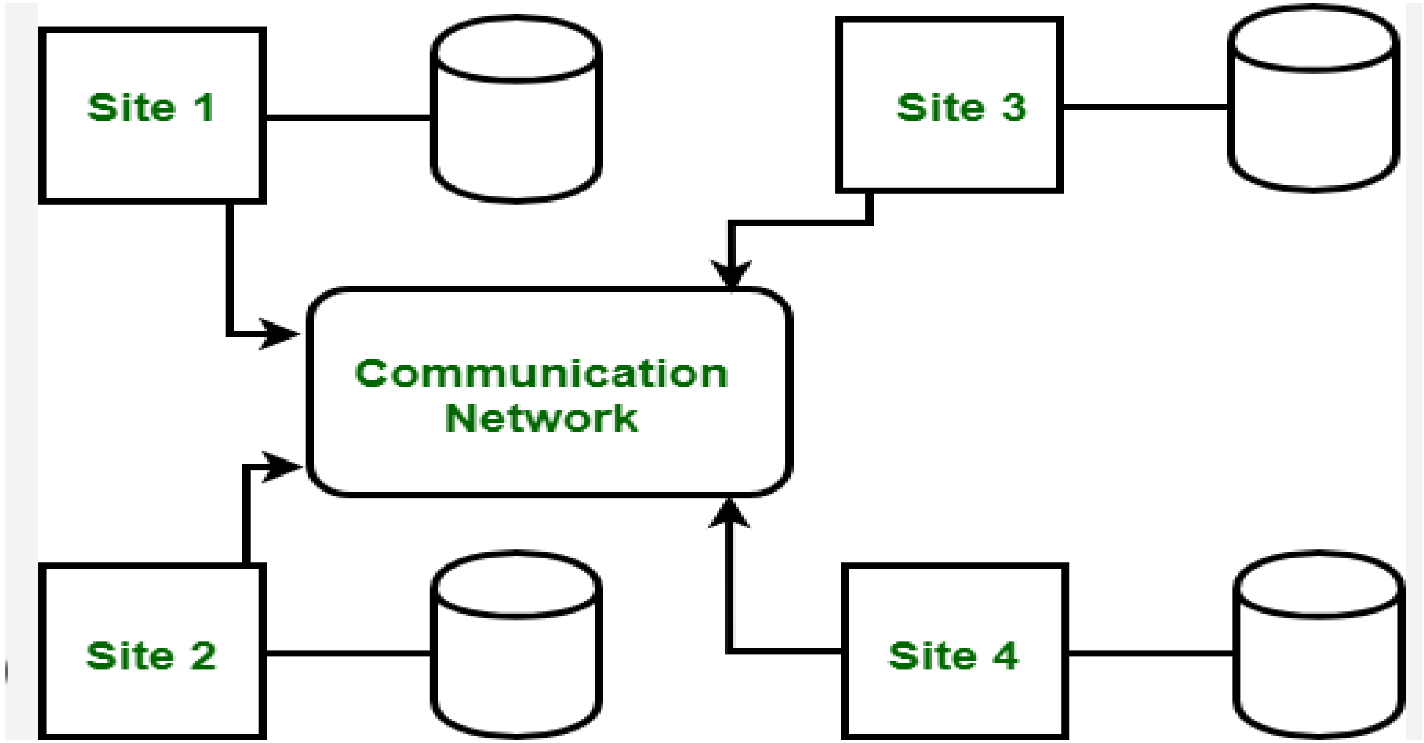
No-SQL: CAP theorem – Document-based: MongoDB data model and CRUD operations; Column-based: Hbase data model and CRUD operations.


SQL Databases
NoSQL Databases
Relational
Analytical (OLAP)
Key-Value
key
value
key
value
key
value
key
value
Column-Family
Graph
Document

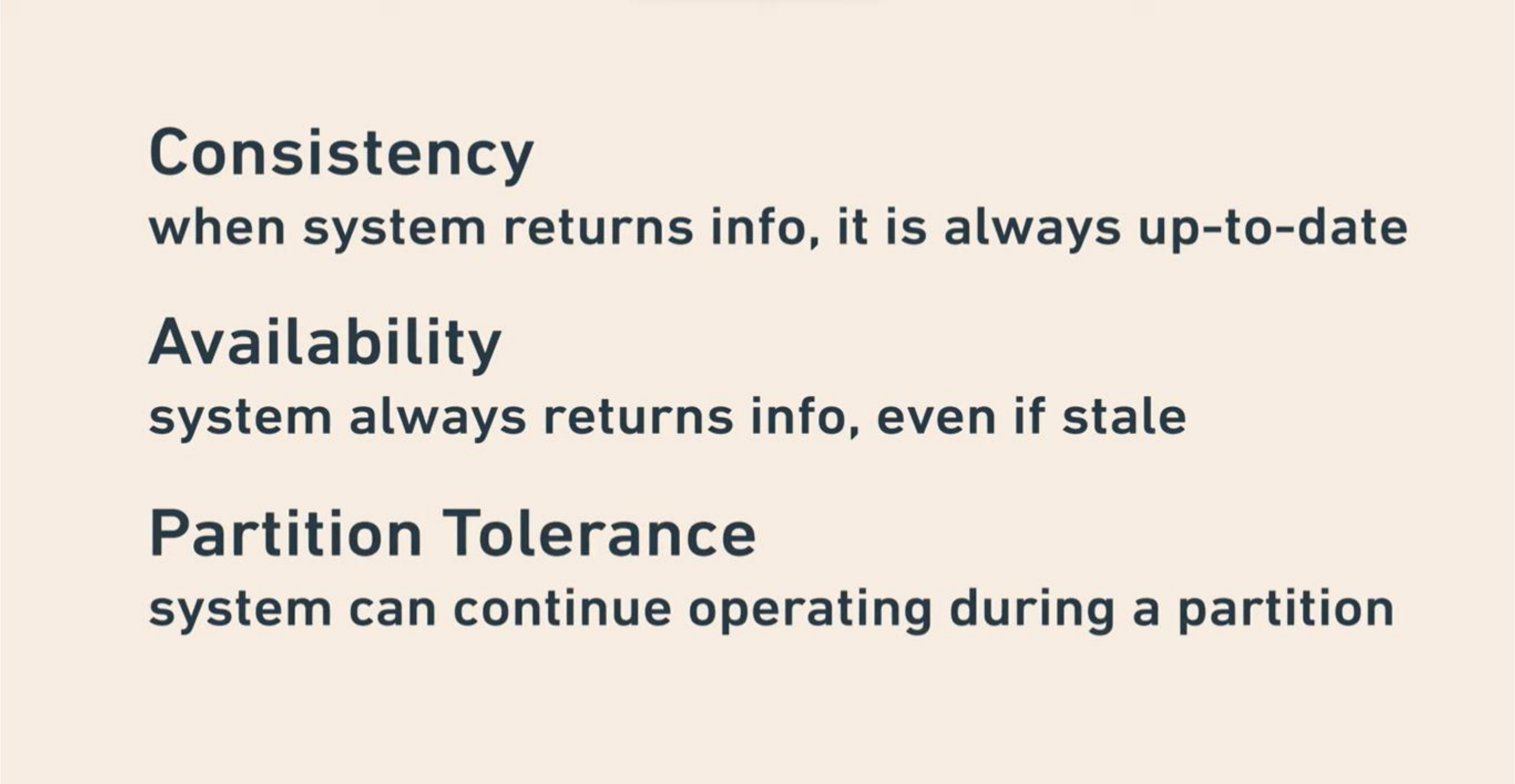
- NoSQL databases use a variety of data models for accessing and managing data.
- These types of databases are optimized specifically for applications that require large data volume, low latency, and flexible data models
- Types of NoSQL Databases
  - Document Databases
  - Wide-Column Databases

Distributed Database

- A distributed database is a database that runs and stores data across multiple computers
- Example: NOSQL



NOSQL CAP THEOREM



Consistency

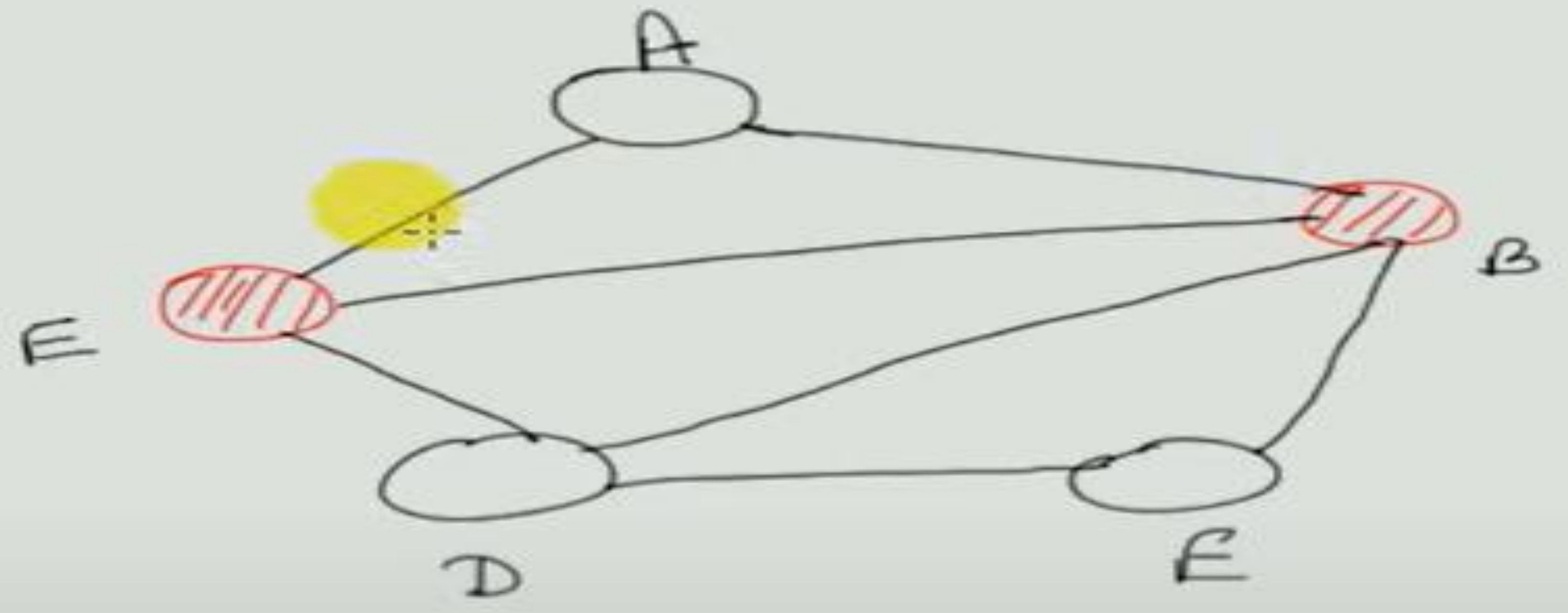
when system returns info, it is always up-to-date

Availability

system always returns info, even if stale

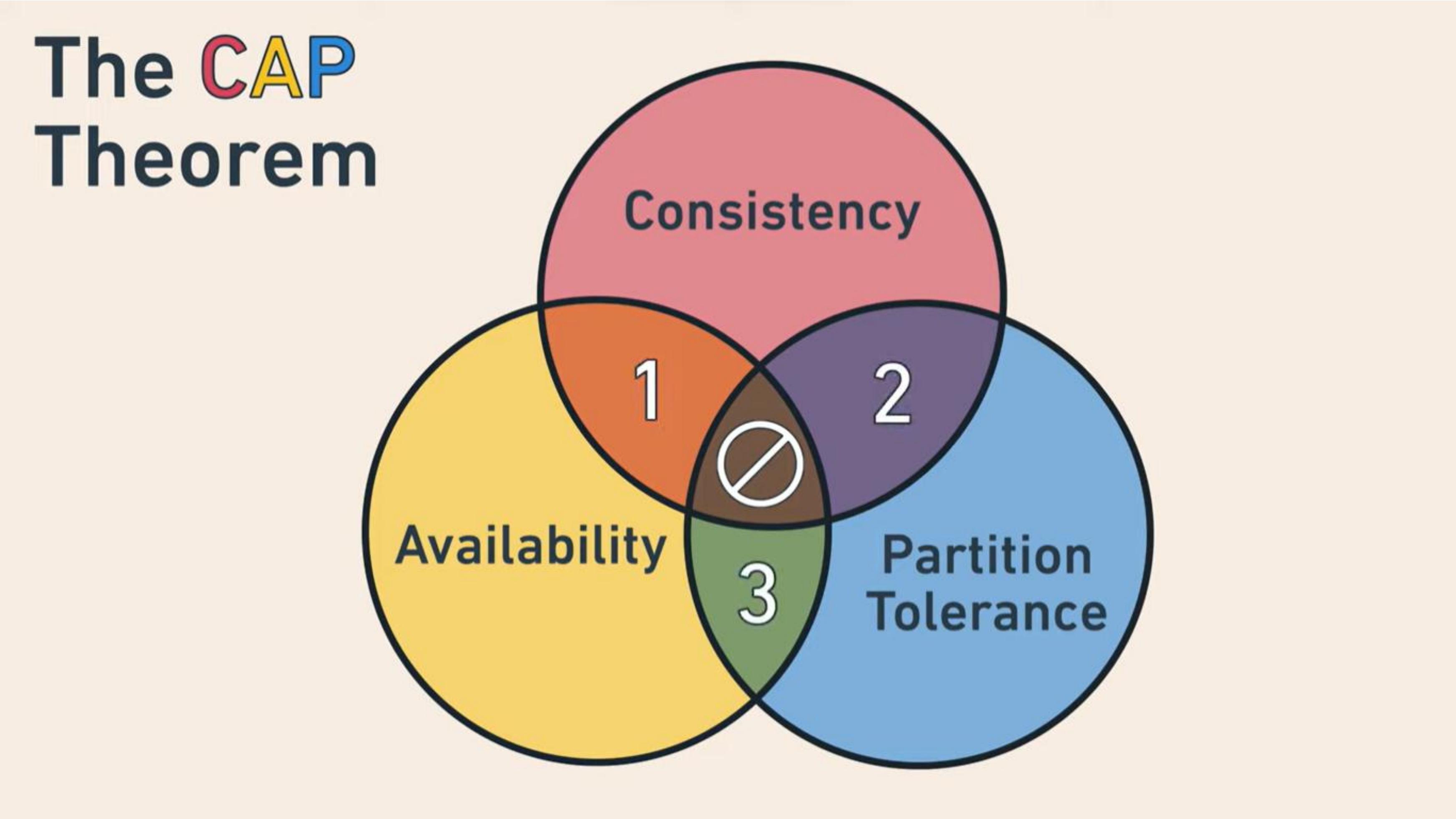
Partition Tolerance

system can continue operating during a partition

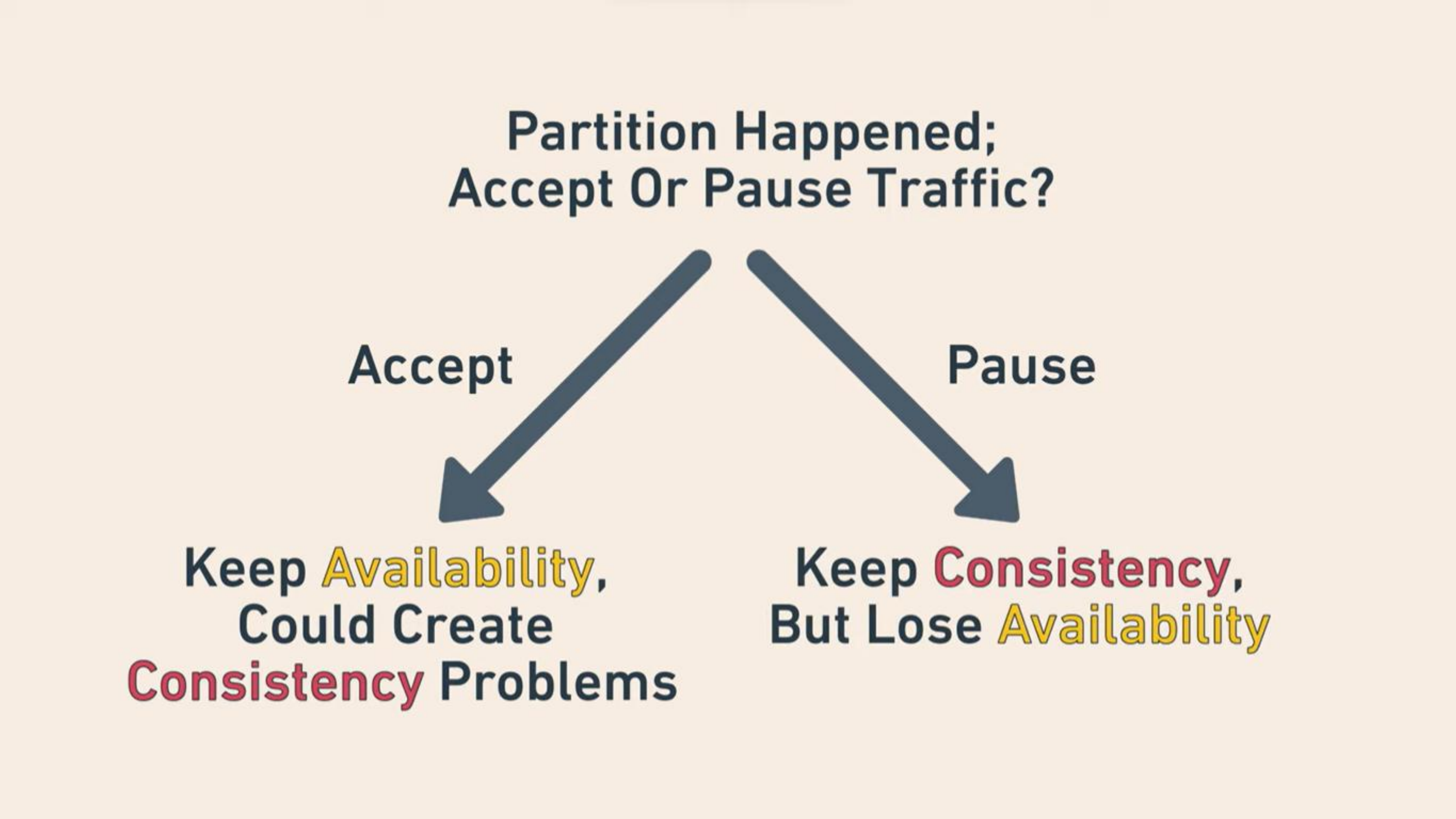


Replica Nodes : B, E

Fig: Distributed System



The CAP Theorem



Partition Happened;
Accept Or Pause Traffic?

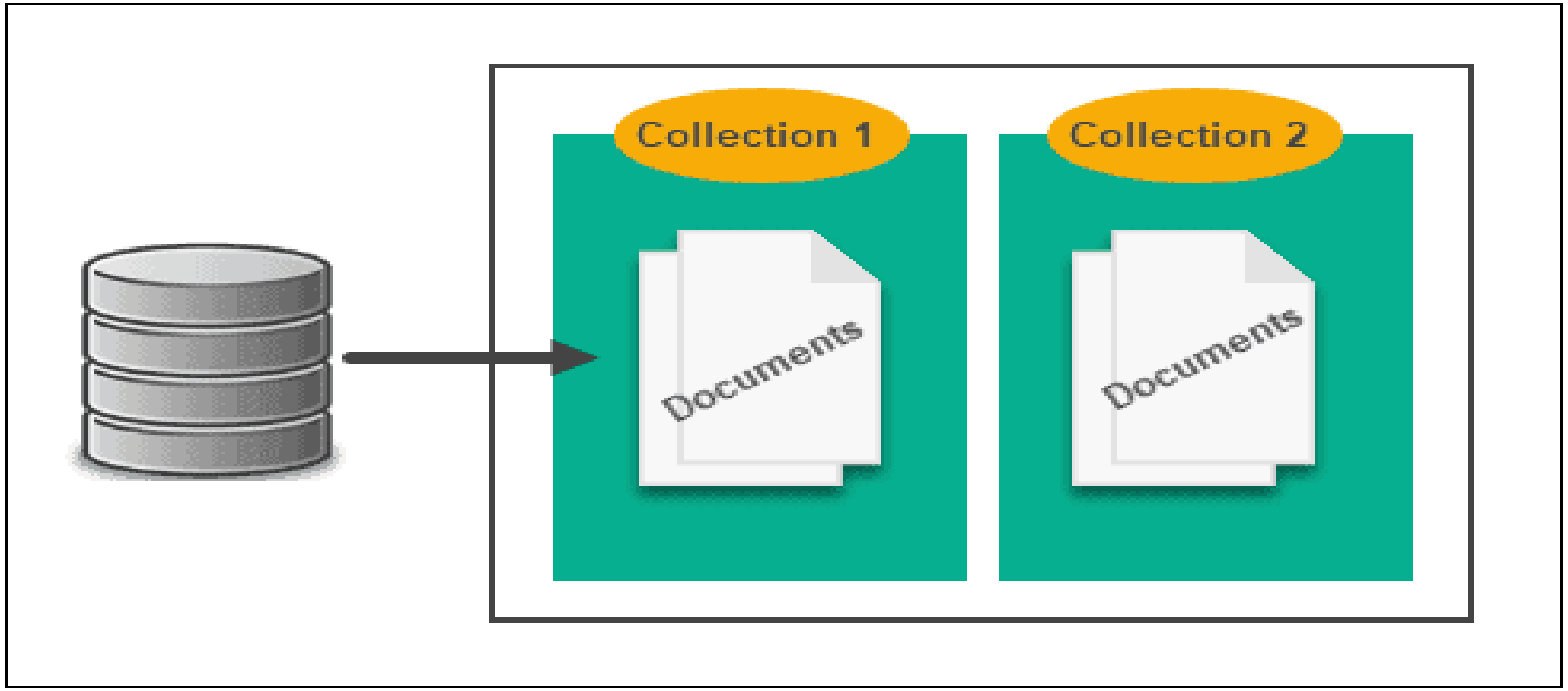
Accept

Pause

Keep Availability,
Could Create
Consistency Problems

Keep Consistency,
But Lose Availability

- **A document database** is a type of NoSQL database that consists of sets of key-value pairs stored into a document



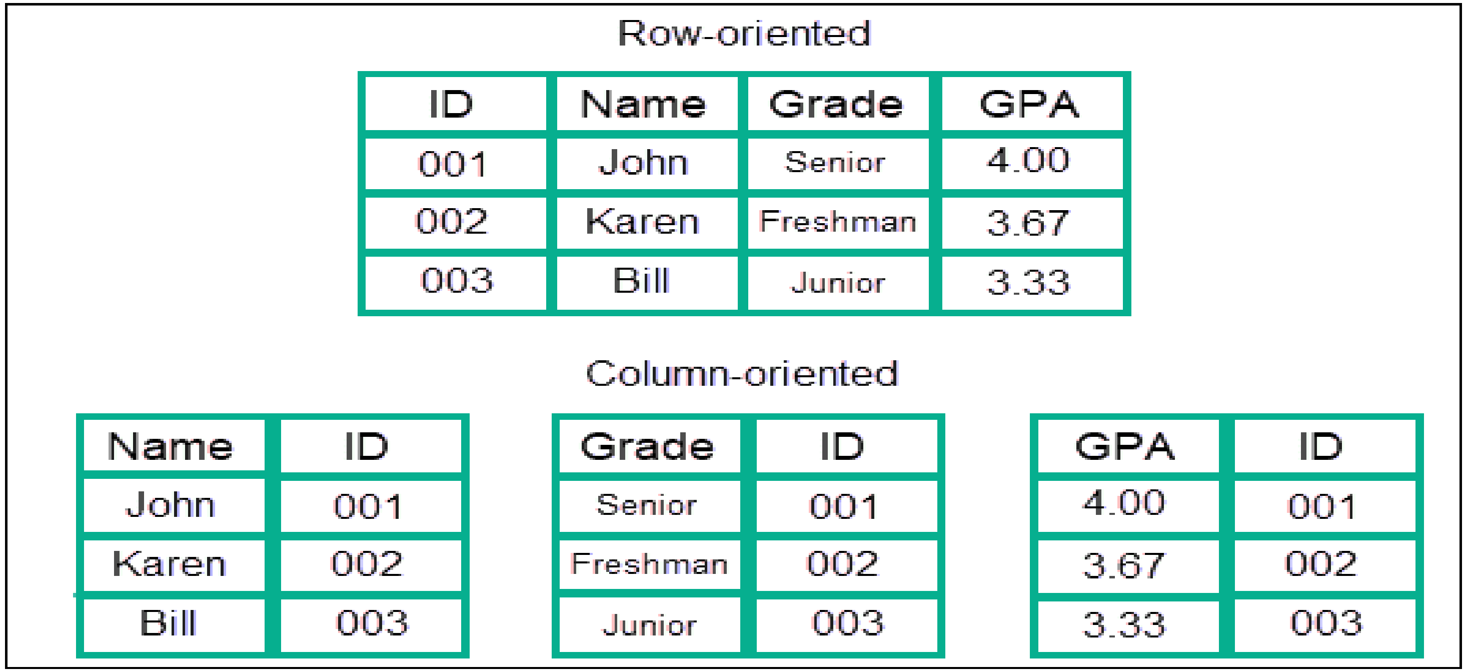
- Being a NoSQL database, you can easily store data without implementing a schema

- You can transfer the object model directly into a document using several different formats. The most commonly used are JSON, BSON, and XML.

- Examples of NoSQL document databases include MongoDB, CouchDB, Elasticsearch, and others.

Wide-Column Database

Wide-column stores are another type of NoSQL database. In them, data is stored and grouped into separately stored columns instead of rows. Such databases organize information into columns that function similarly to tables in relational databases.



Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

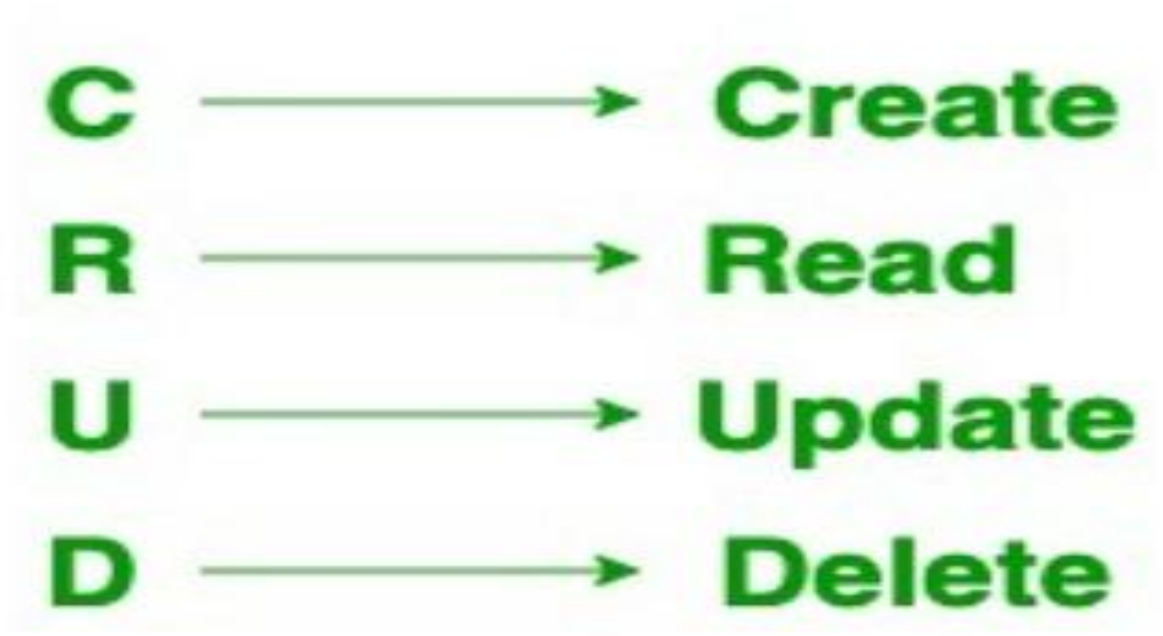
Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

- The most significant benefit of having column-oriented databases is that you can store large amounts of data within a single column

- Examples of popular wide-column databases include
  - HBase , Apache Cassandra, and CosmoDB.

MONGODB CRUD OPERATIONS

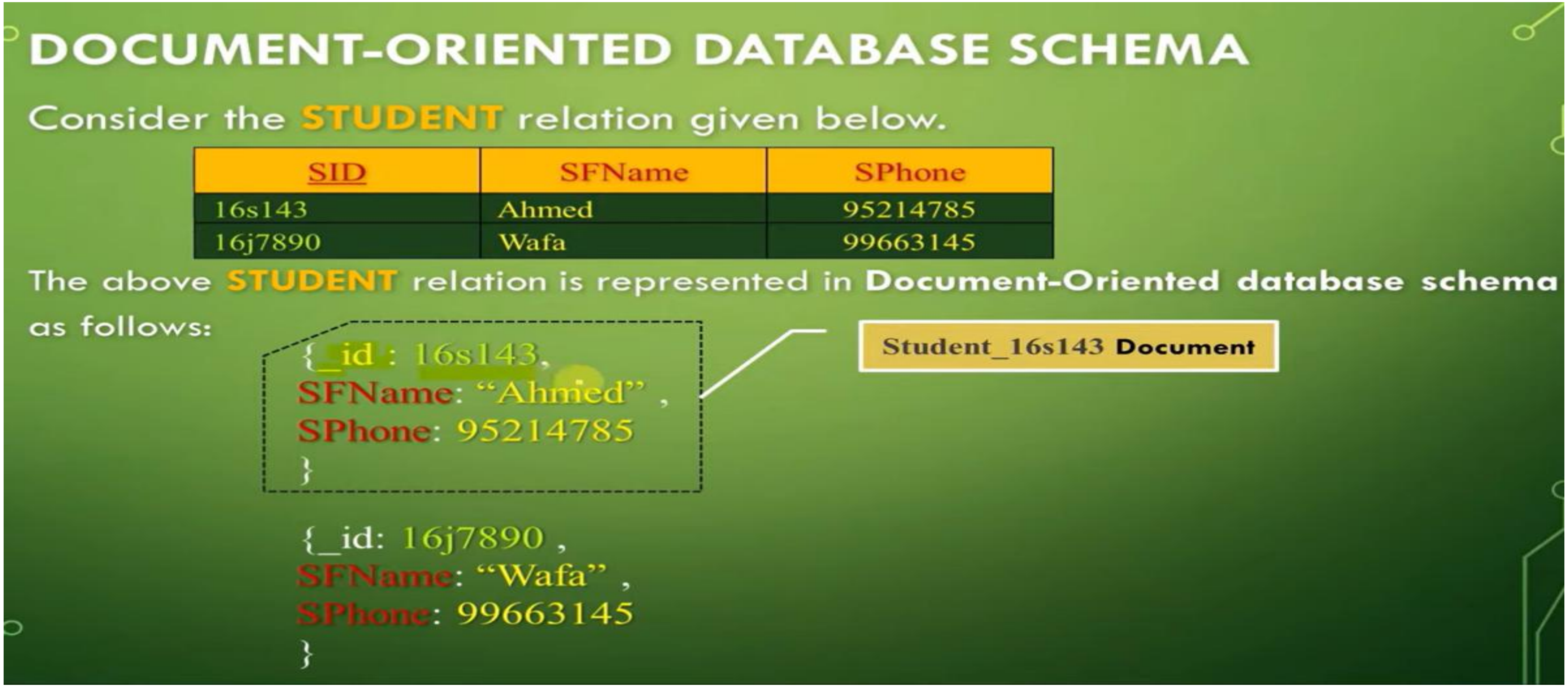


C → Create

R → Read

U → Update

D → Delete



DOCUMENT-ORIENTED DATABASE SCHEMA

Consider the **STUDENT** relation given below.

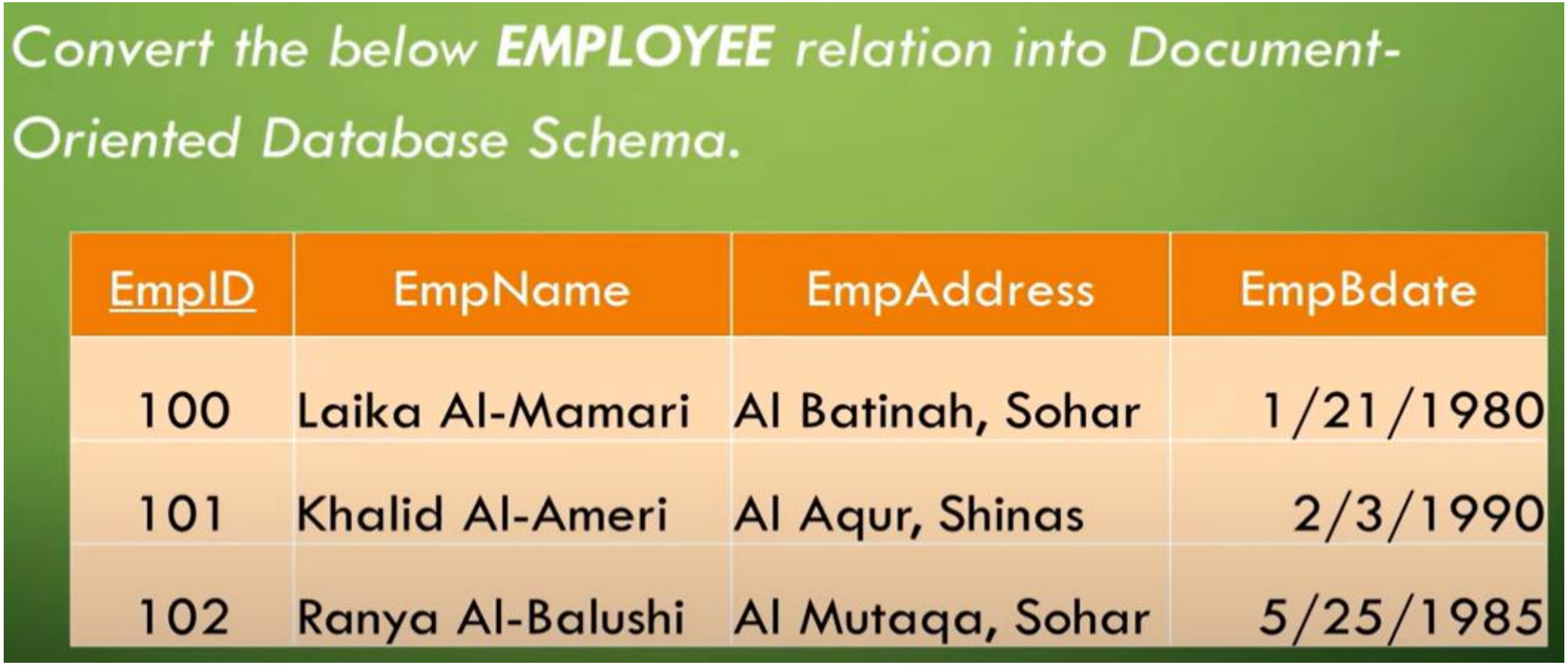
SID	SFName	SPhone
16s143	Ahmed	95214785
16j7890	Wafa	99663145

The above **STUDENT** relation is represented in **Document-Oriented database schema** as follows:

```
{_id : 16s143,
SFName: "Ahmed" ,
SPhone: 95214785
}
```

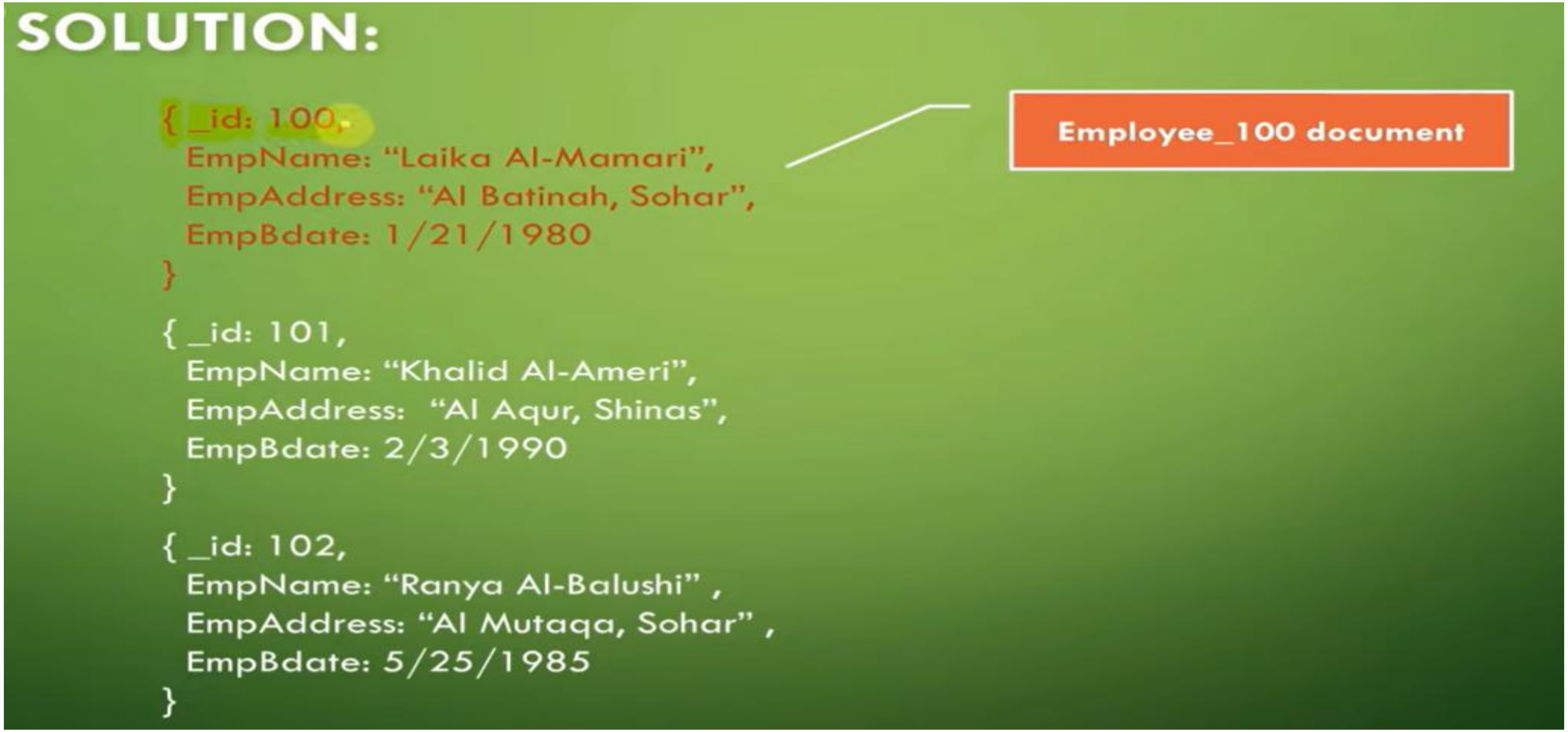
Student_16s143 **Document**

```
{_id: 16j7890 ,
SFName: "Wafa" ,
SPhone: 99663145
}
```



Convert the below ***EMPLOYEE*** *relation into Document-Oriented Database Schema.*

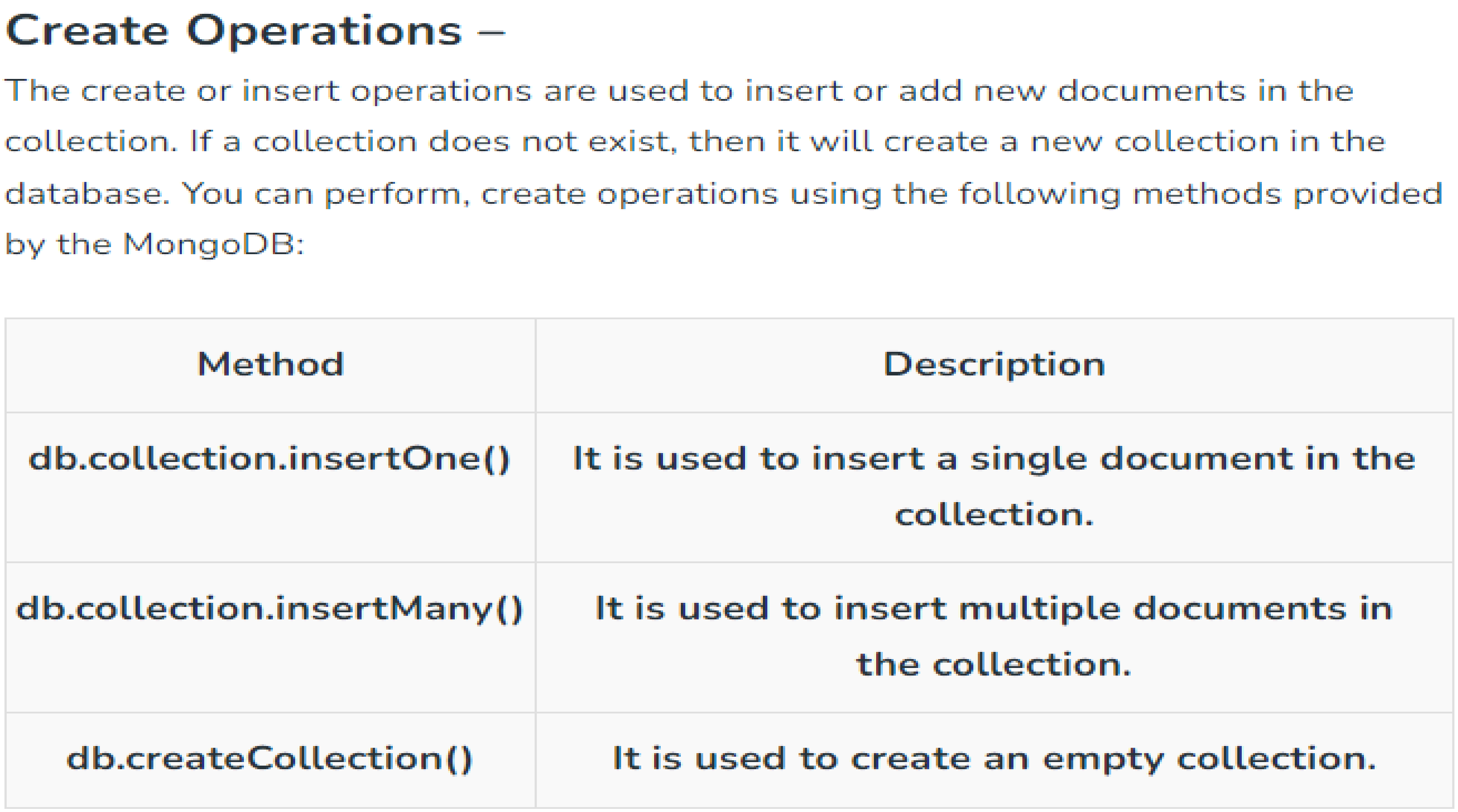
EmpID	EmpName	EmpAddress	EmpBdate
100	Laika Al-Mamari	Al Batinah, Sohar	1/21/1980
101	Khalid Al-Ameri	Al Aqur, Shinas	2/3/1990
102	Ranya Al-Balushi	Al Mutaqa, Sohar	5/25/1985



SOLUTION:

```
{ _id: 100,
  EmpName: “Laika Al-Mamari”,
  EmpAddress: “Al Batinah, Sohar”,
  EmpBdate: 1/21/1980
}

{ _id: 101,
  EmpName: “Khalid Al-Ameri”,
  EmpAddress:  “Al Aqur, Shinas”,
  EmpBdate: 2/3/1990
}

{ _id: 102,
  EmpName: “Ranya Al-Balushi” ,
  EmpAddress: “Al Mutaqa, Sohar” ,
  EmpBdate: 5/25/1985
}
```

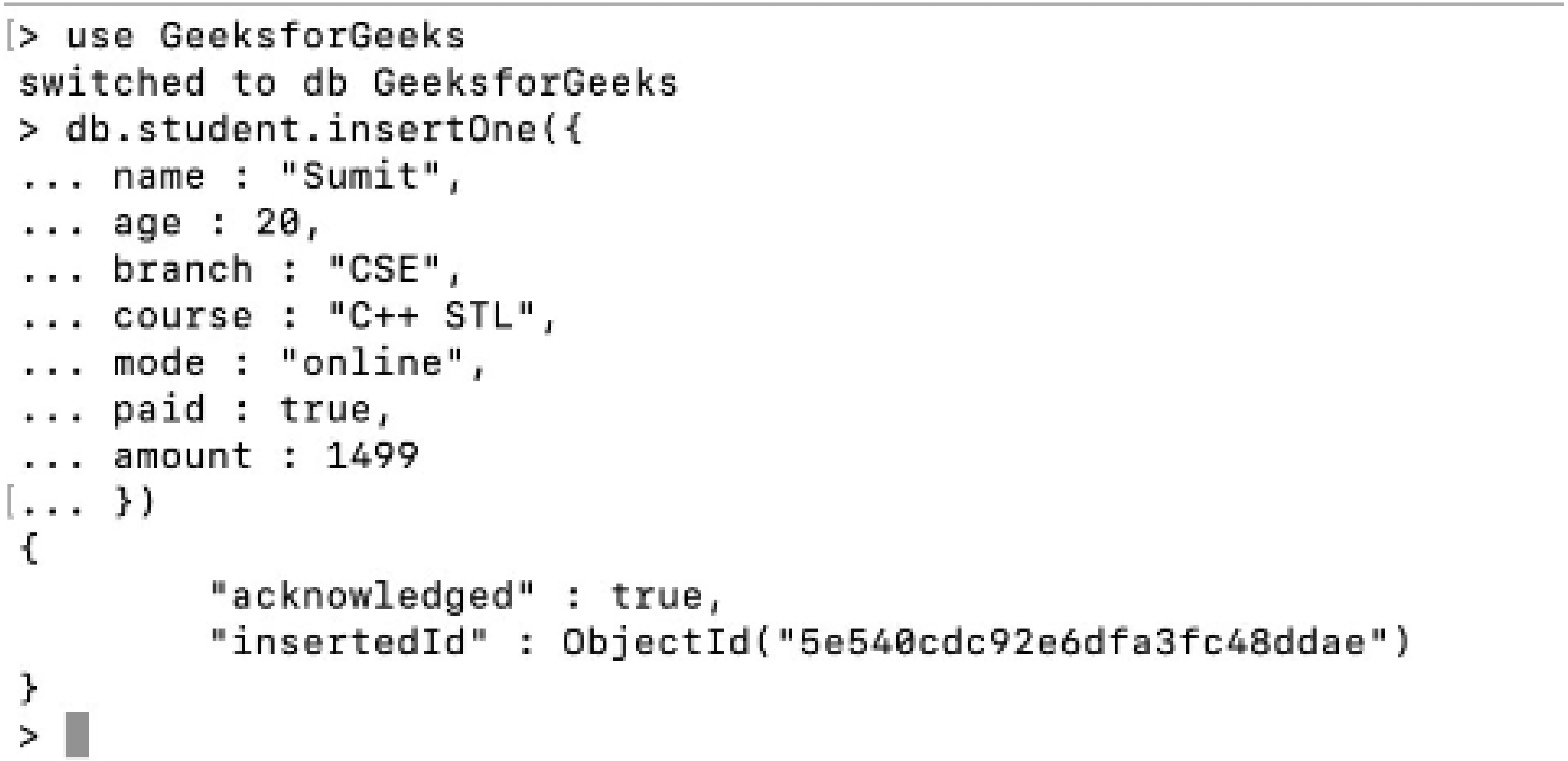
Employee_100 document



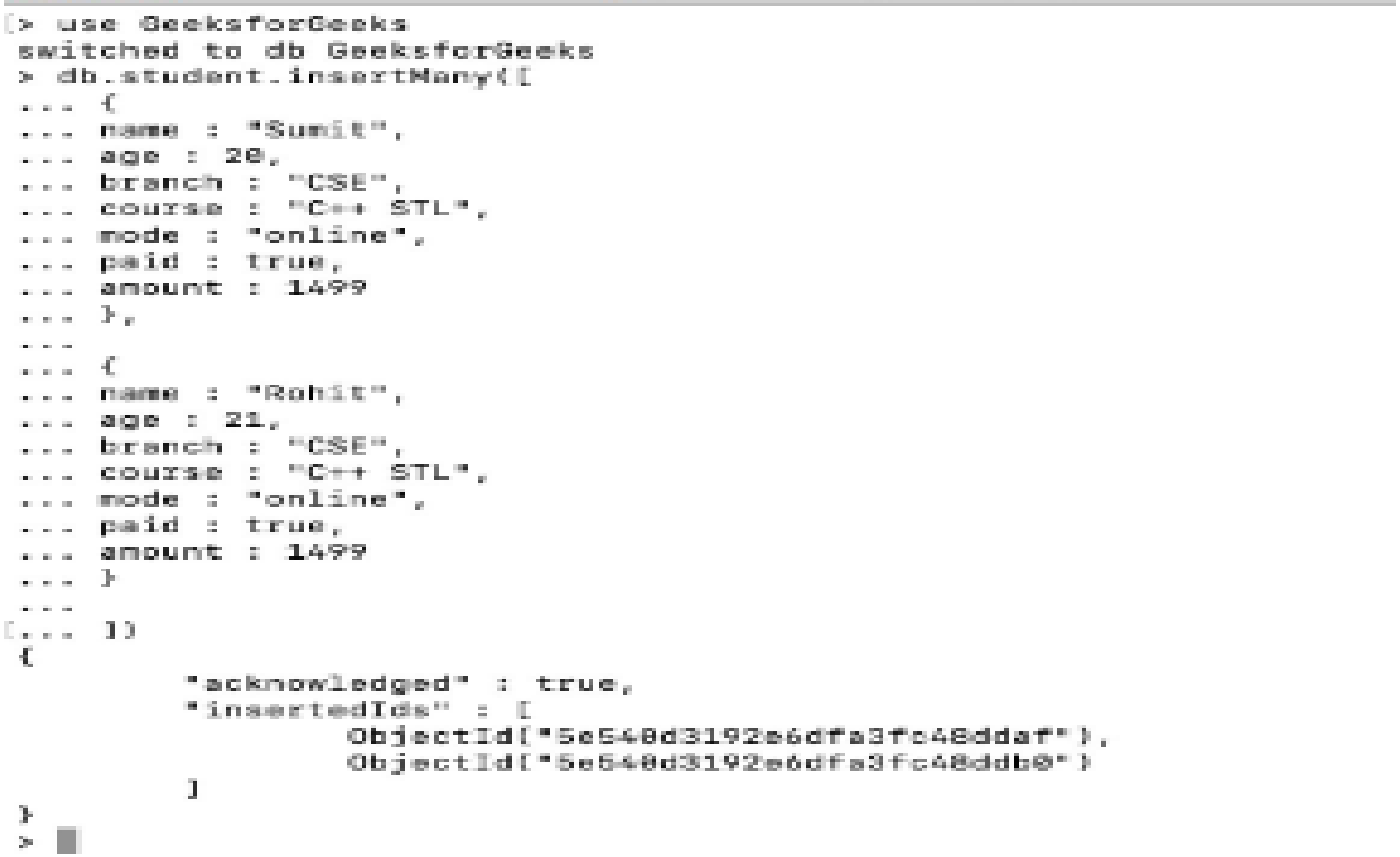
Create Operations –

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. You can perform, create operations using the following methods provided by the MongoDB:

Method	Description
db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.
db.createCollection()	It is used to create an empty collection.



```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertOne({
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5e540cdc92e6dfa3fc48ddae")
}
> 
```



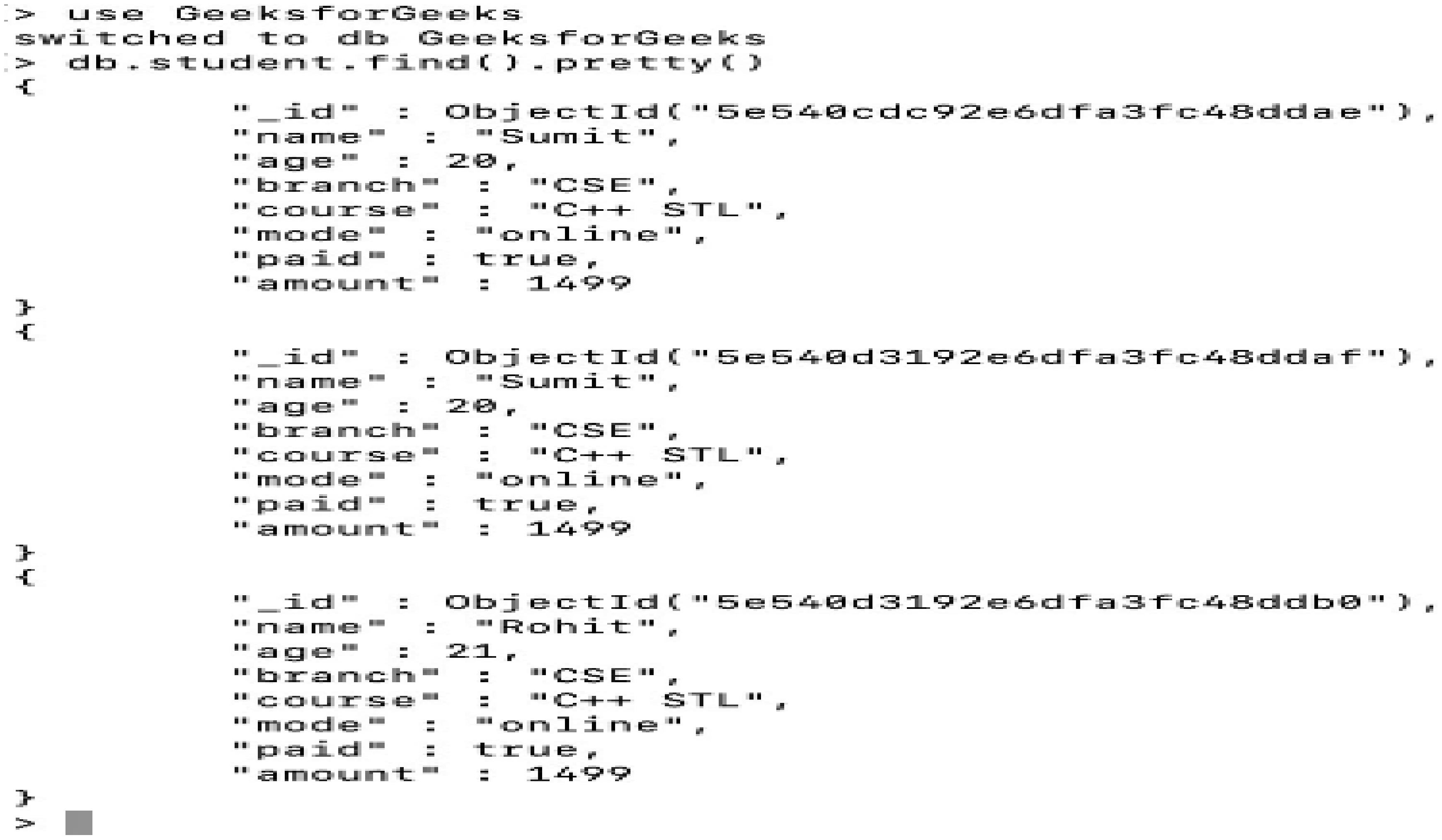
```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertMany([
... {
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... },
...
... {
... name : "Rohit",
... age : 21,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... }
...
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5e540d3192e6dfa3fc48ddaf"),
                ObjectId("5e540d3192e6dfa3fc48ddb0")
        ]
}
> 
```

- **Read Operations –**
- The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document.

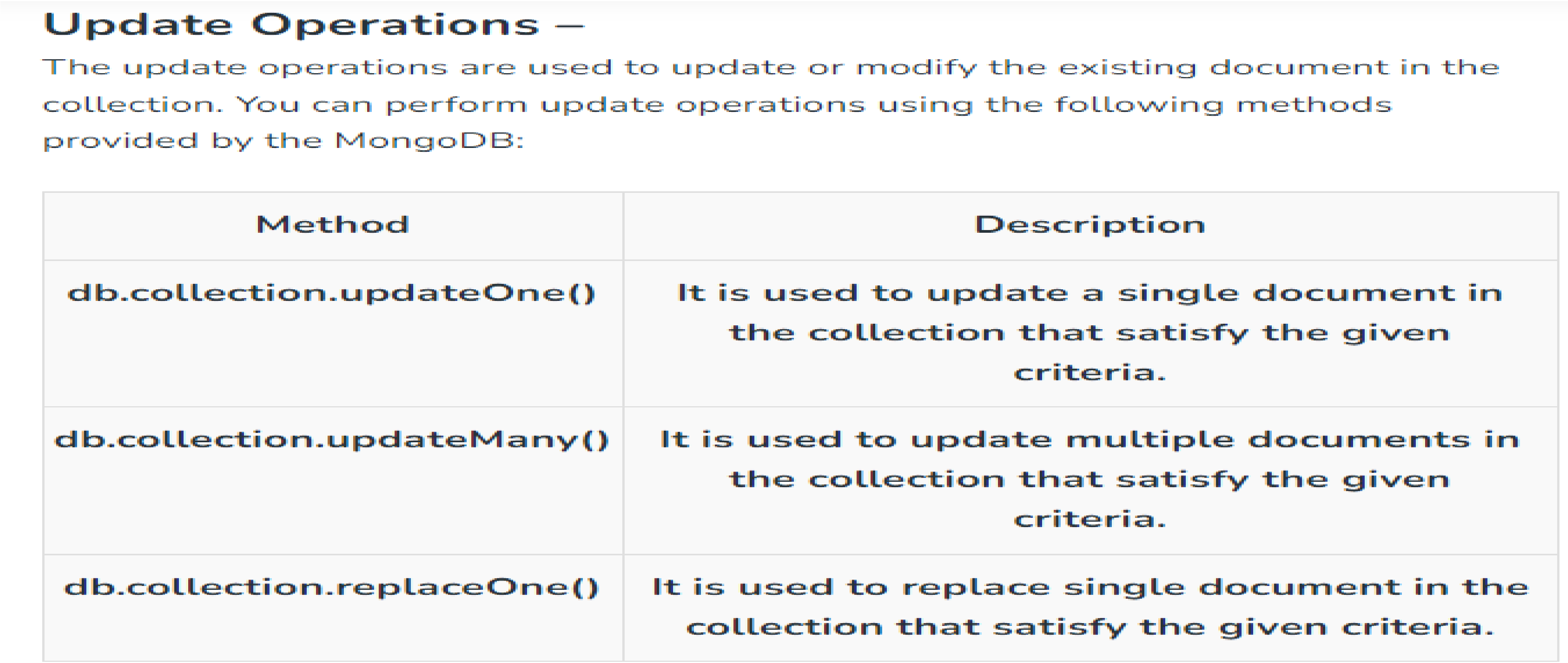
db.collection.find()	It is used to retrieve documents from the collection.

.pretty() : this method is used to decorate the result such that it is easy to read.

Example : In this example, we are retrieving the details of students from the student collection using db.collection.find() method.



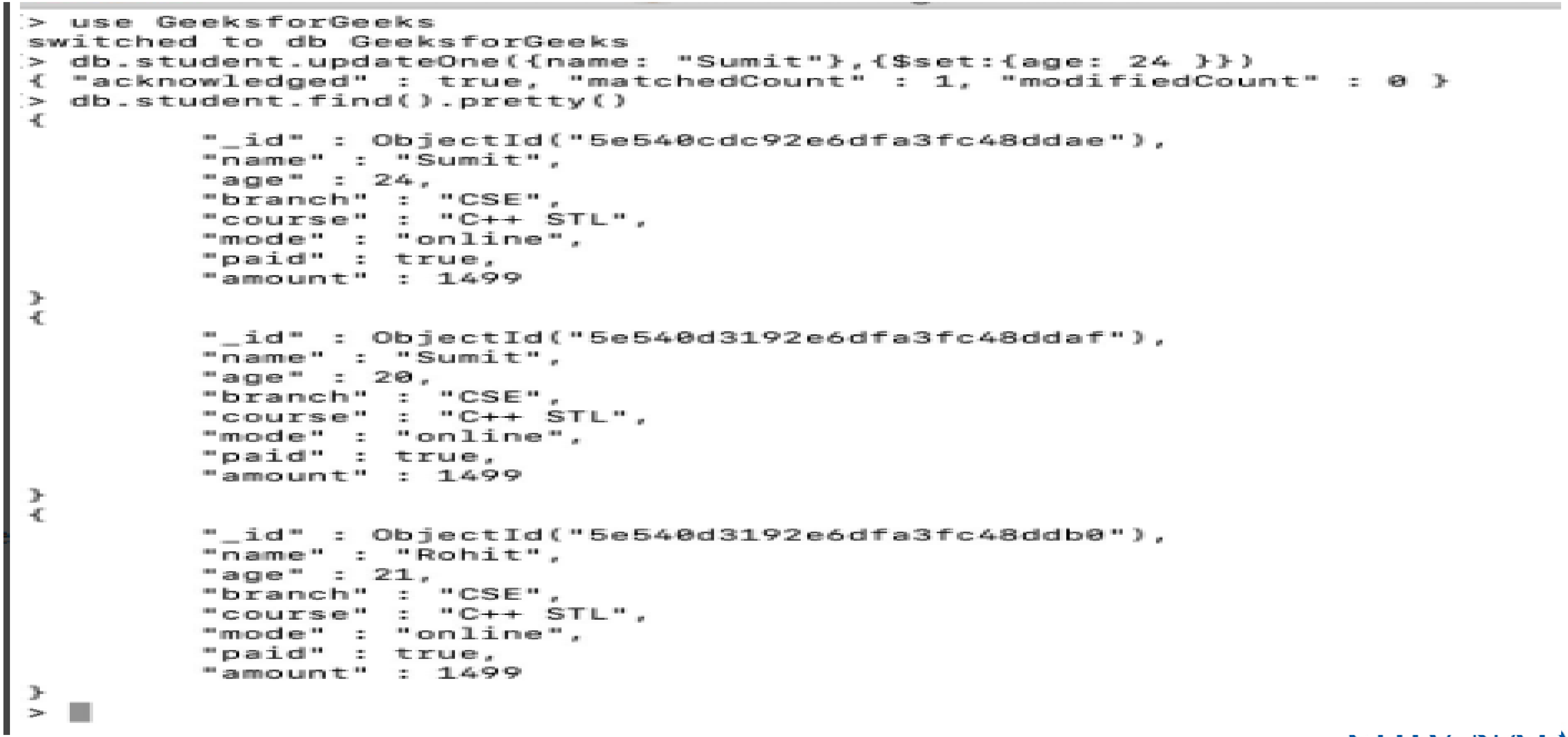
```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
>
```



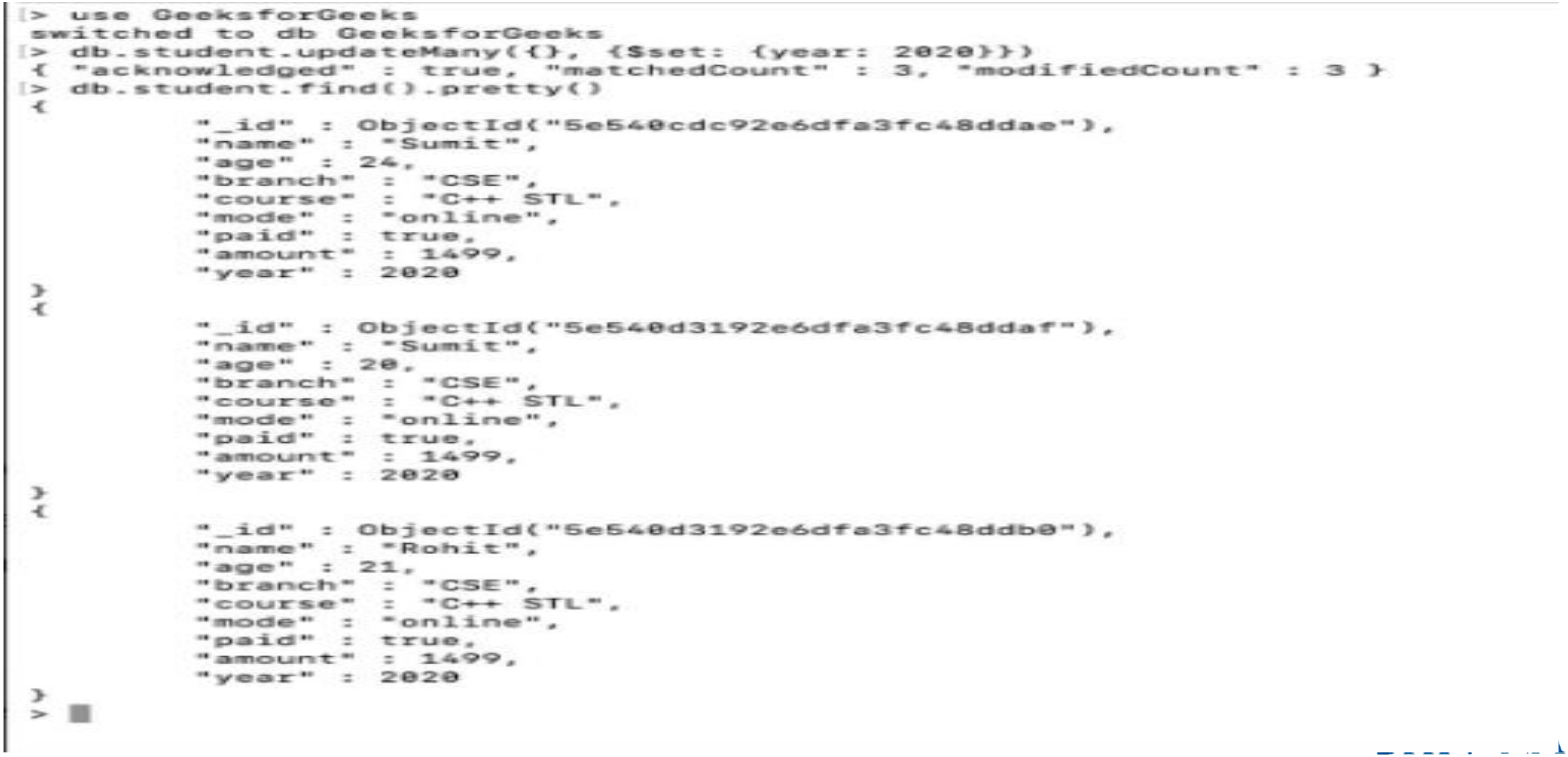
Update Operations –

The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:

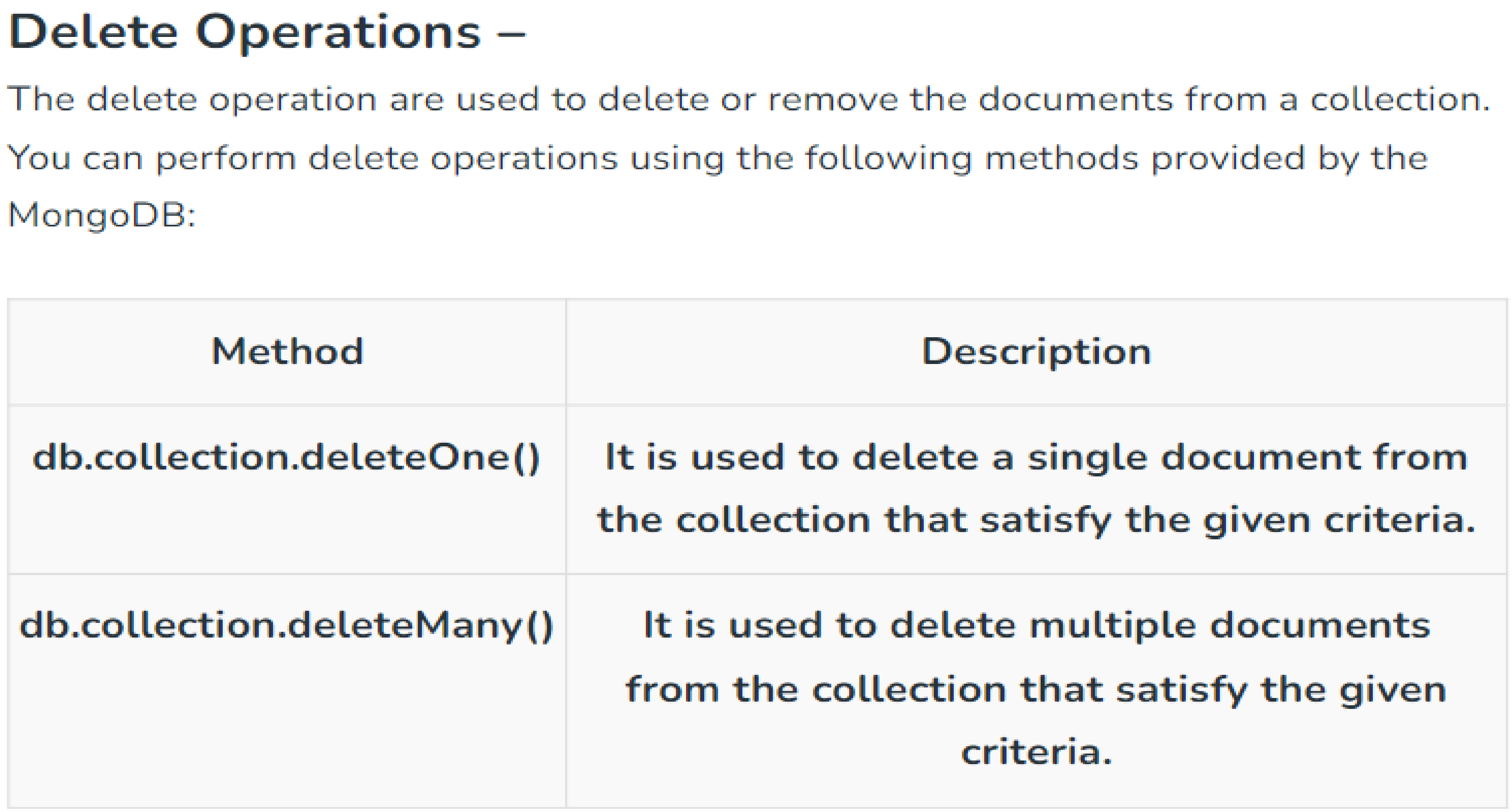
Method	Description
db.collection.updateOne()	It is used to update a single document in the collection that satisfy the given criteria.
db.collection.updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.
db.collection.replaceOne()	It is used to replace single document in the collection that satisfy the given criteria.



```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateOne({name: "Sumit"},{$set:{age: 24 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
>
```



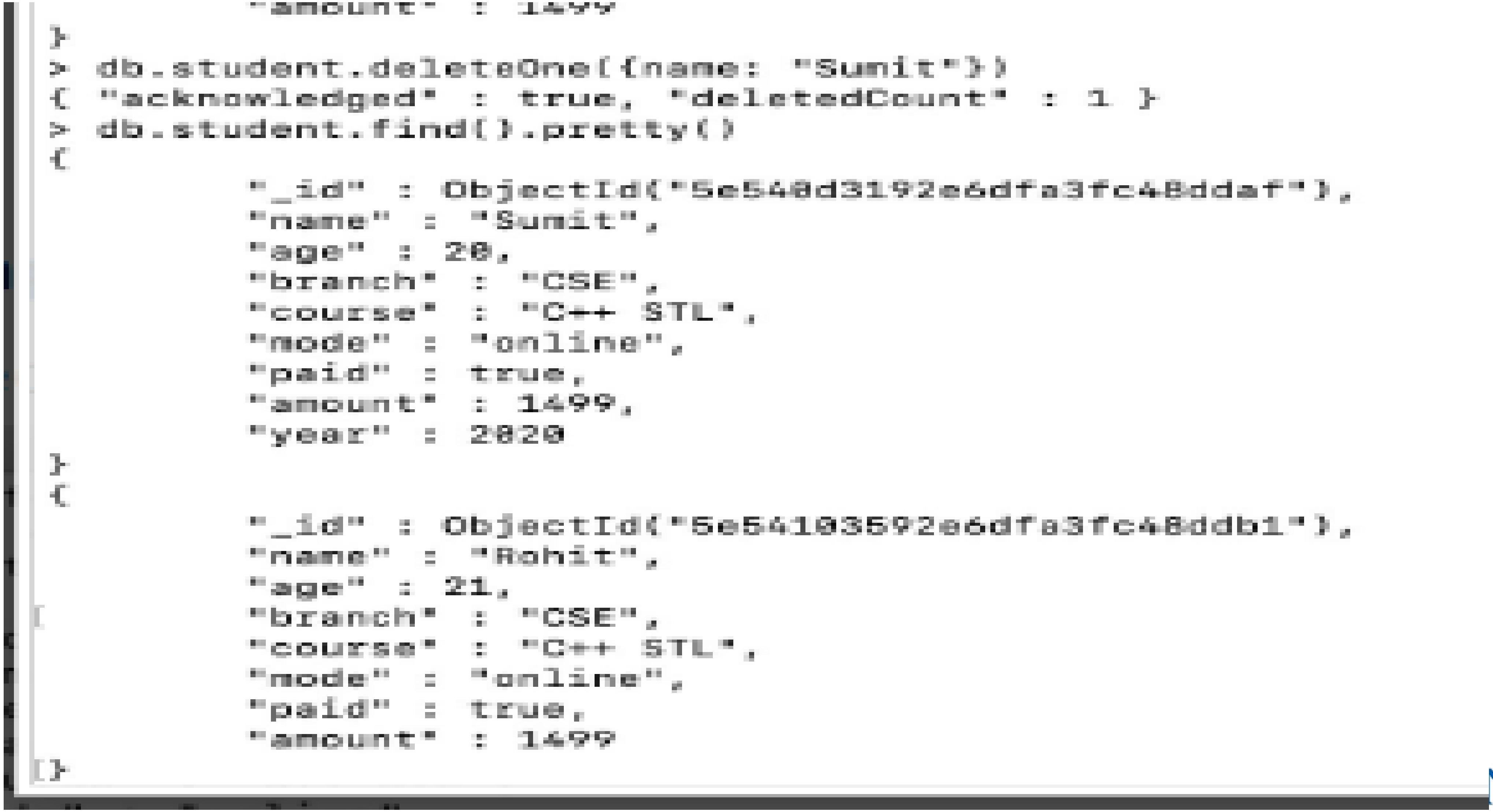
```
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateMany({}, {$set: {year: 2020}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
        "name" : "Sumit",
        "age" : 24,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
> 
```



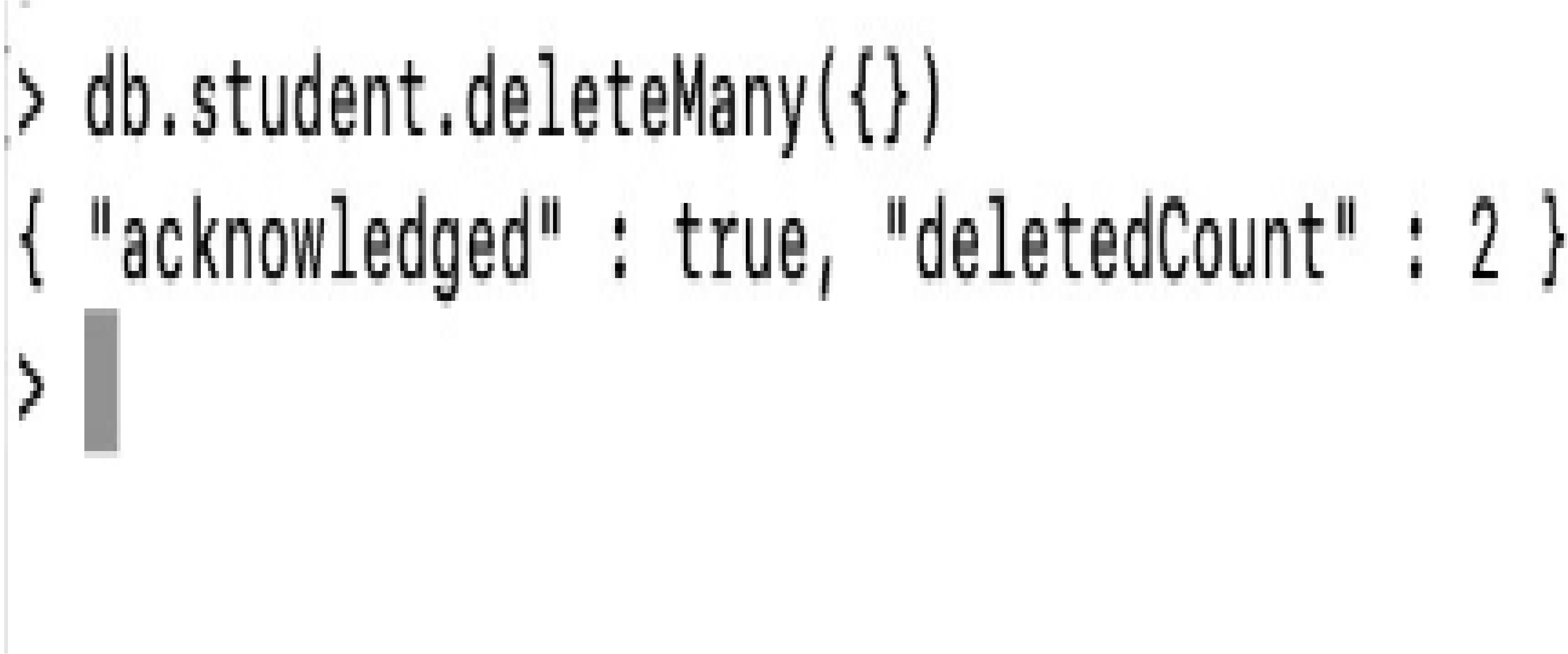
Delete Operations –

The delete operation are used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the MongoDB:

Method	Description
db.collection.deleteOne()	**It is used to delete a single document from the collection that satisfy the given criteria.**
db.collection.deleteMany()	**It is used to delete multiple documents from the collection that satisfy the given criteria.**



```
        "amount" : 1499
}
> db.student.deleteOne({name: "Sumit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.student.find().pretty()
{
        "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
        "name" : "Sumit",
        "age" : 20,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499,
        "year" : 2020
}
{
        "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
        "name" : "Rohit",
        "age" : 21,
        "branch" : "CSE",
        "course" : "C++ STL",
        "mode" : "online",
        "paid" : true,
        "amount" : 1499
}
```

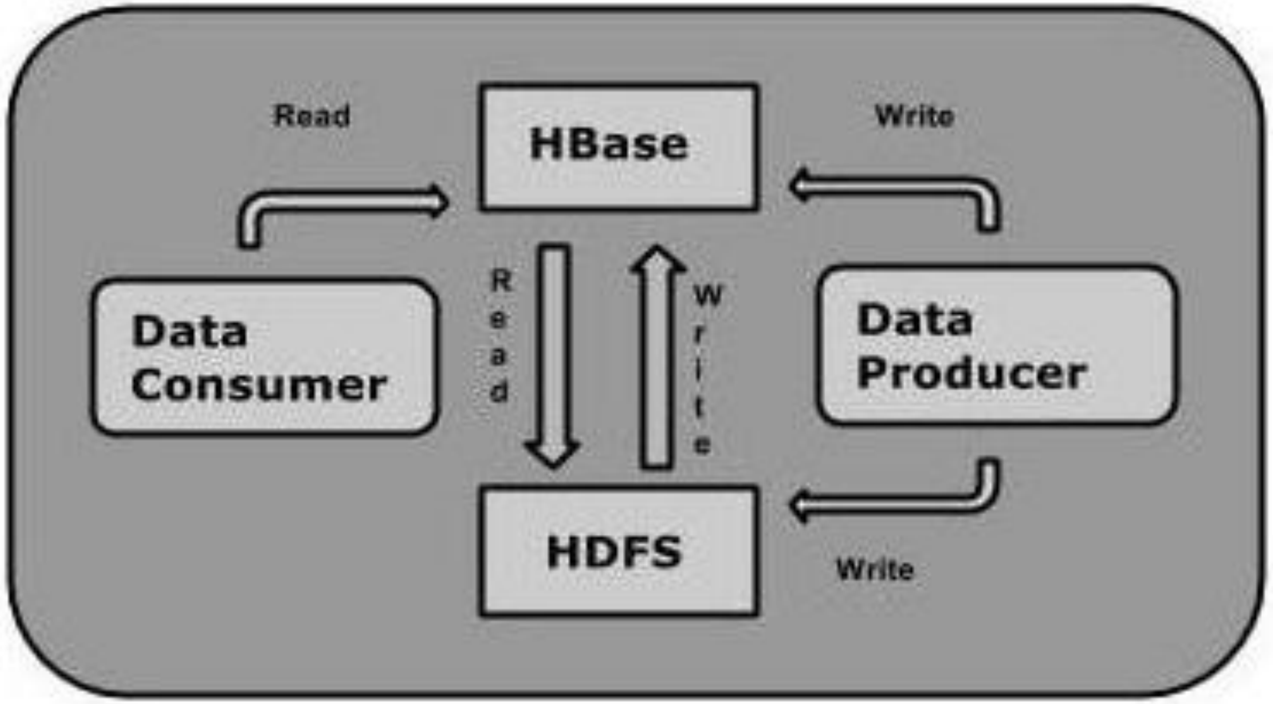


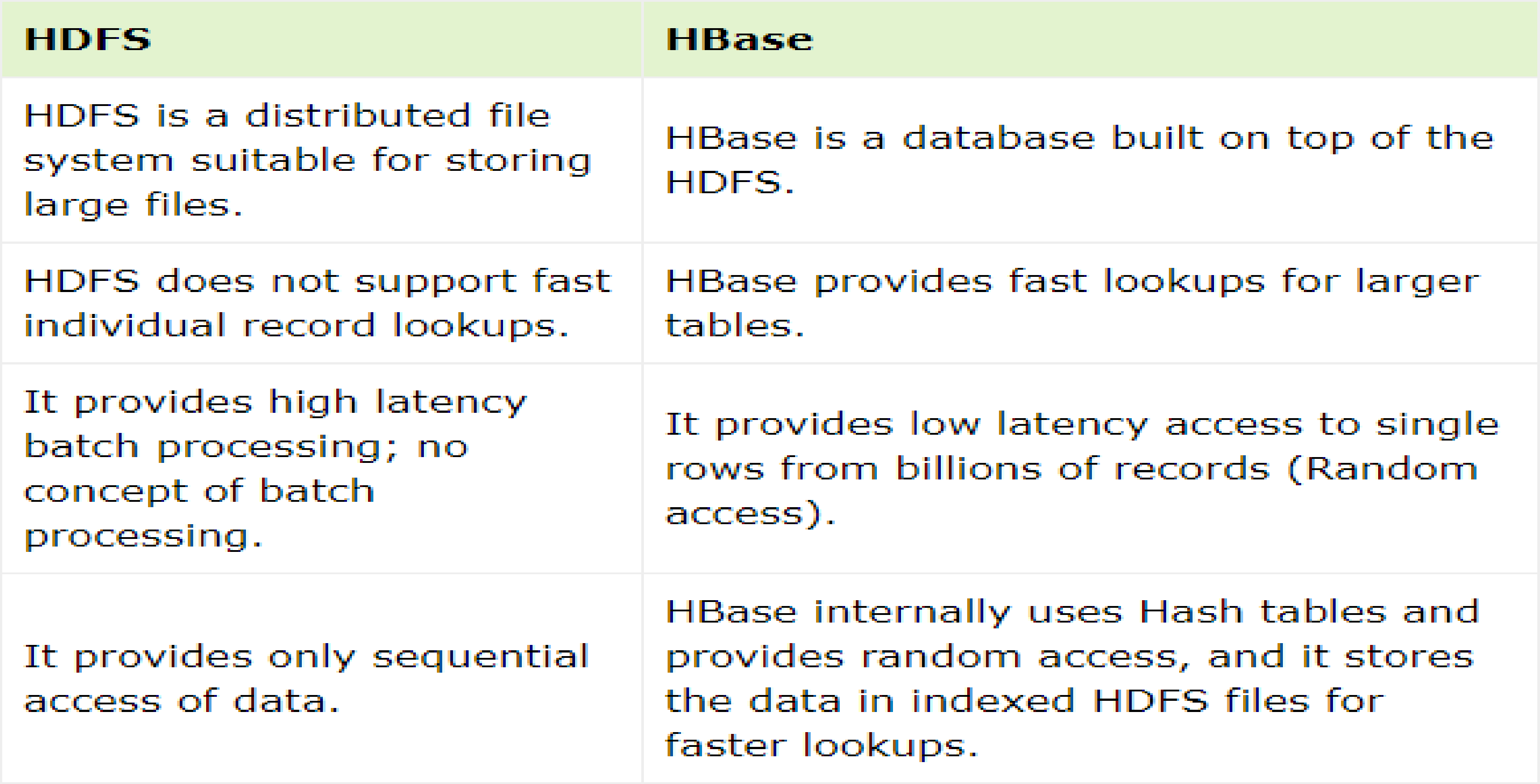
```
> db.student.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
>
```

HBASE

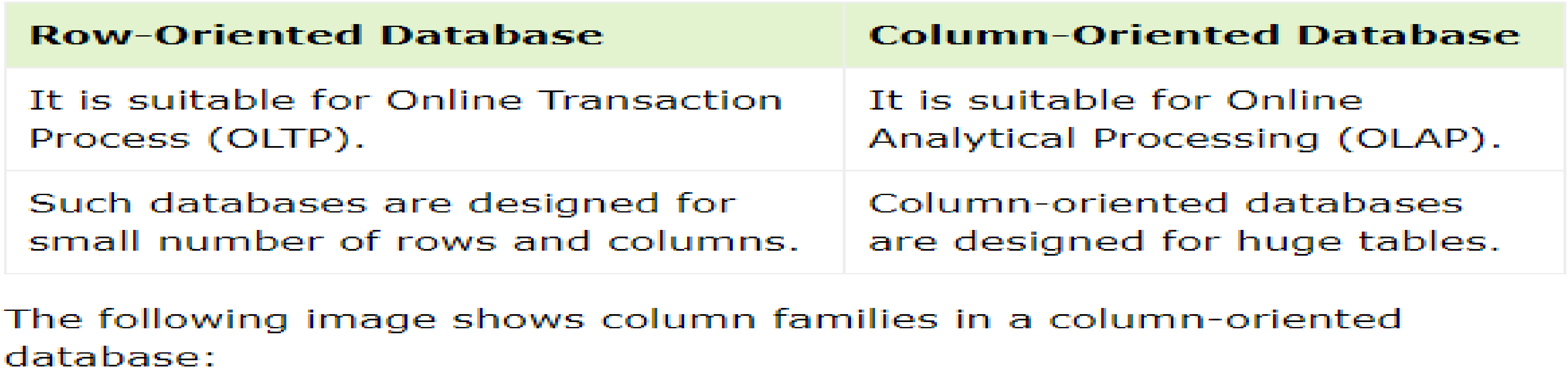
- HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

- it leverages the fault tolerance provided by the Hadoop File System (HDFS).


Read
HBase
Write
Data Consumer
Read
Write
Data Producer
HDFS
Write

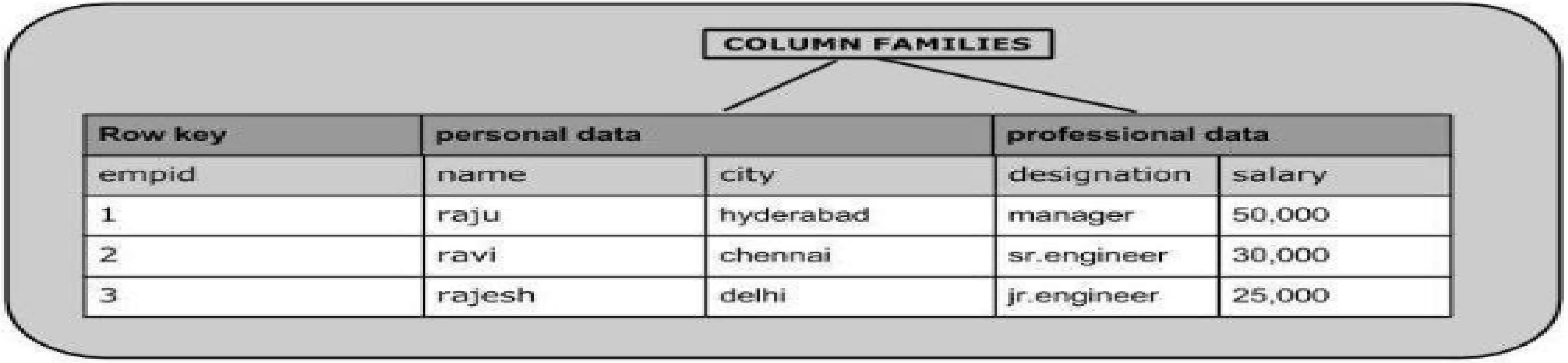


HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

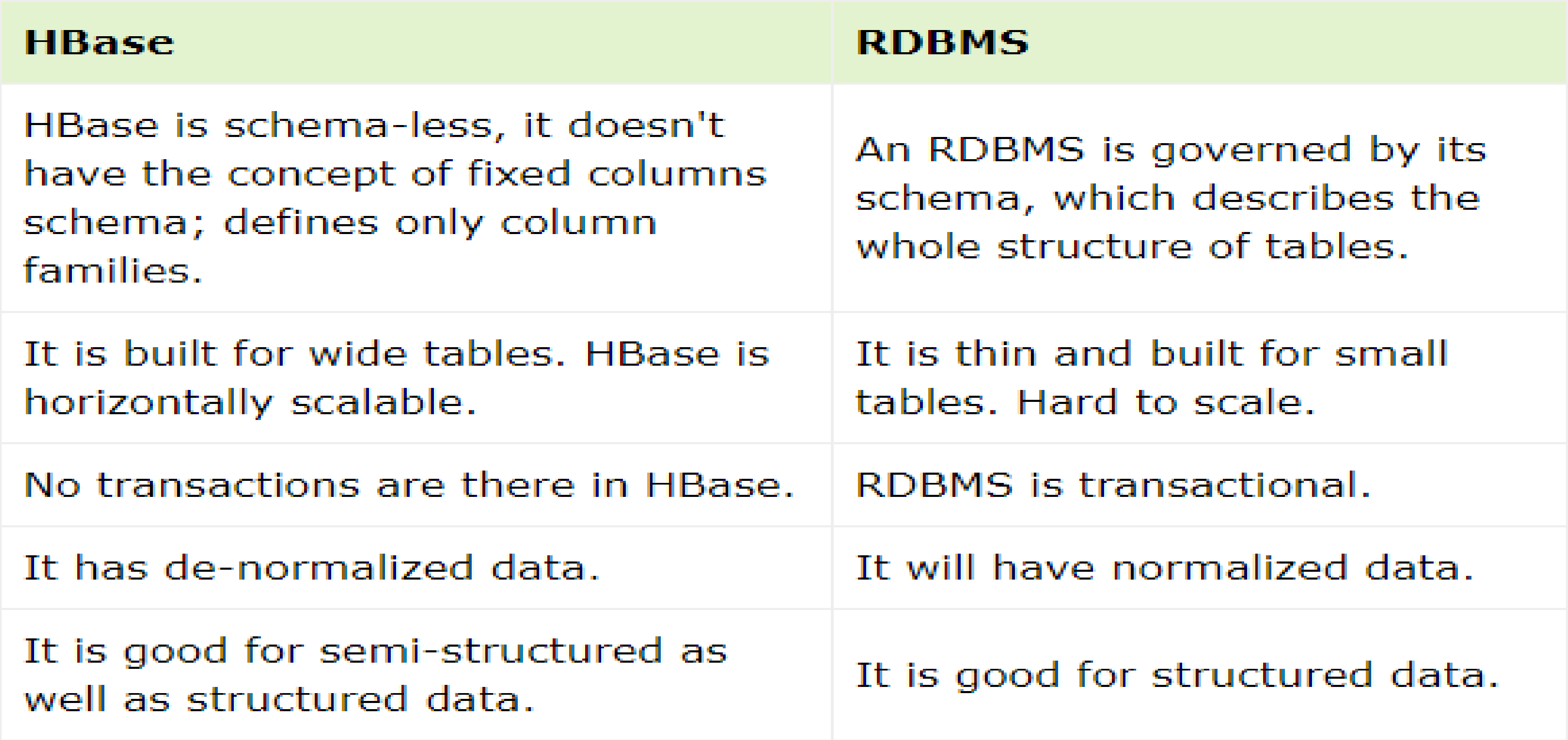


Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:

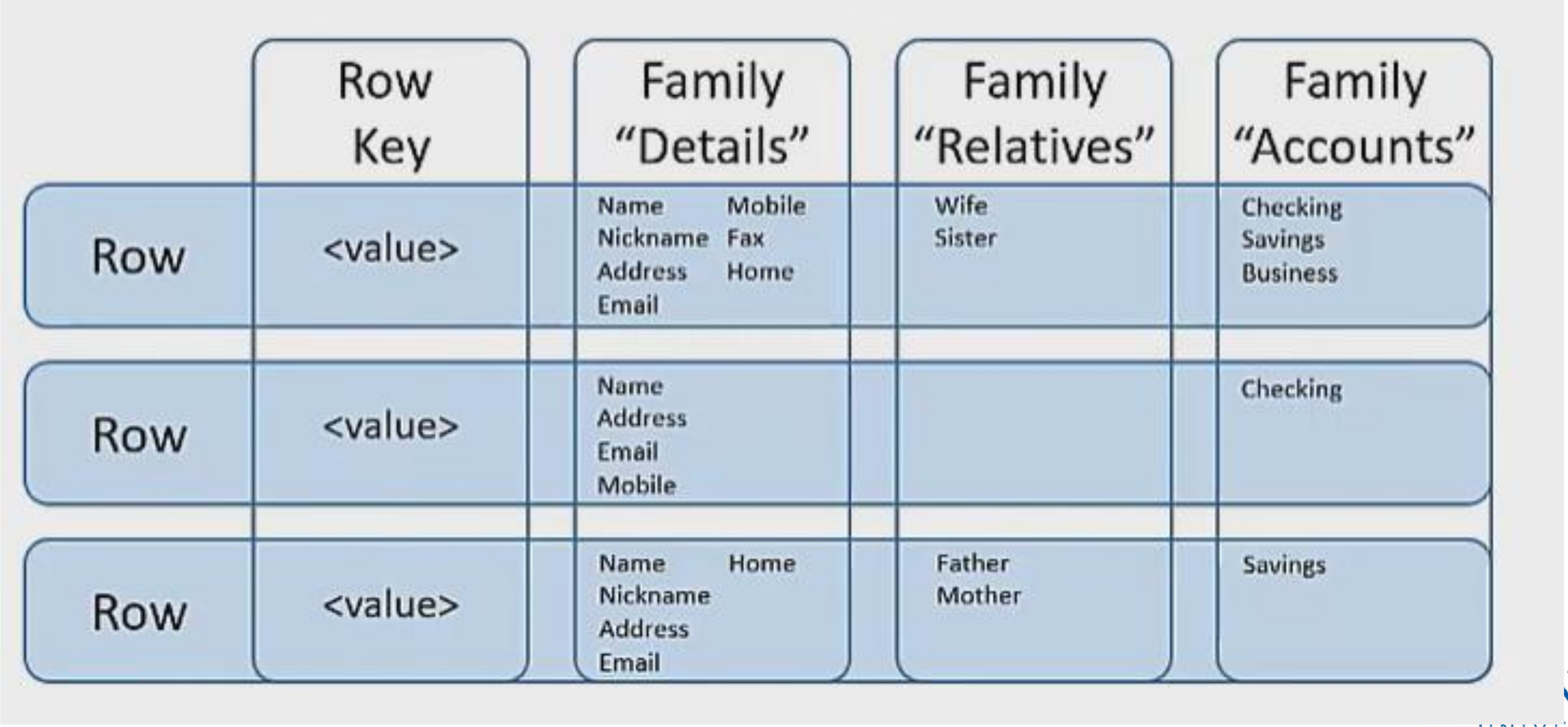


Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

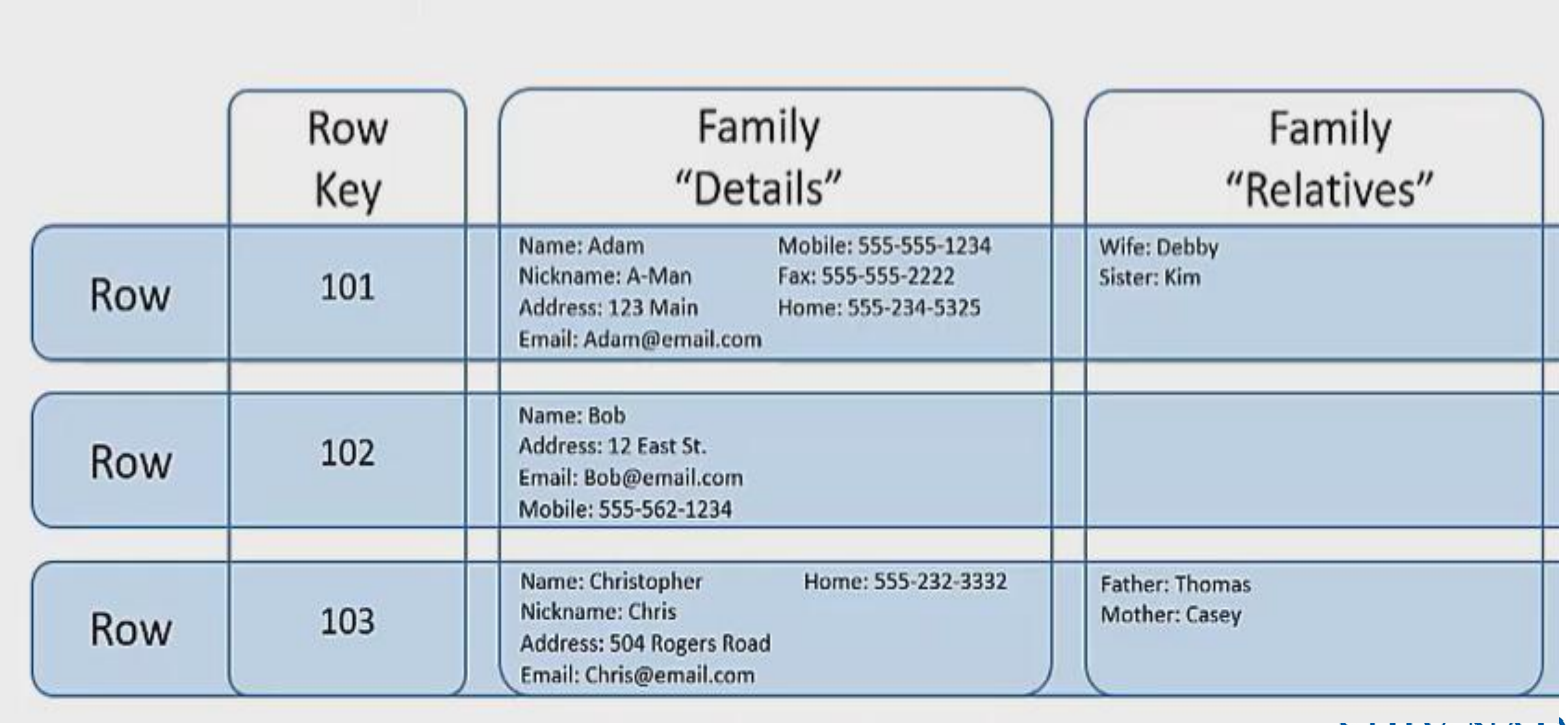


HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

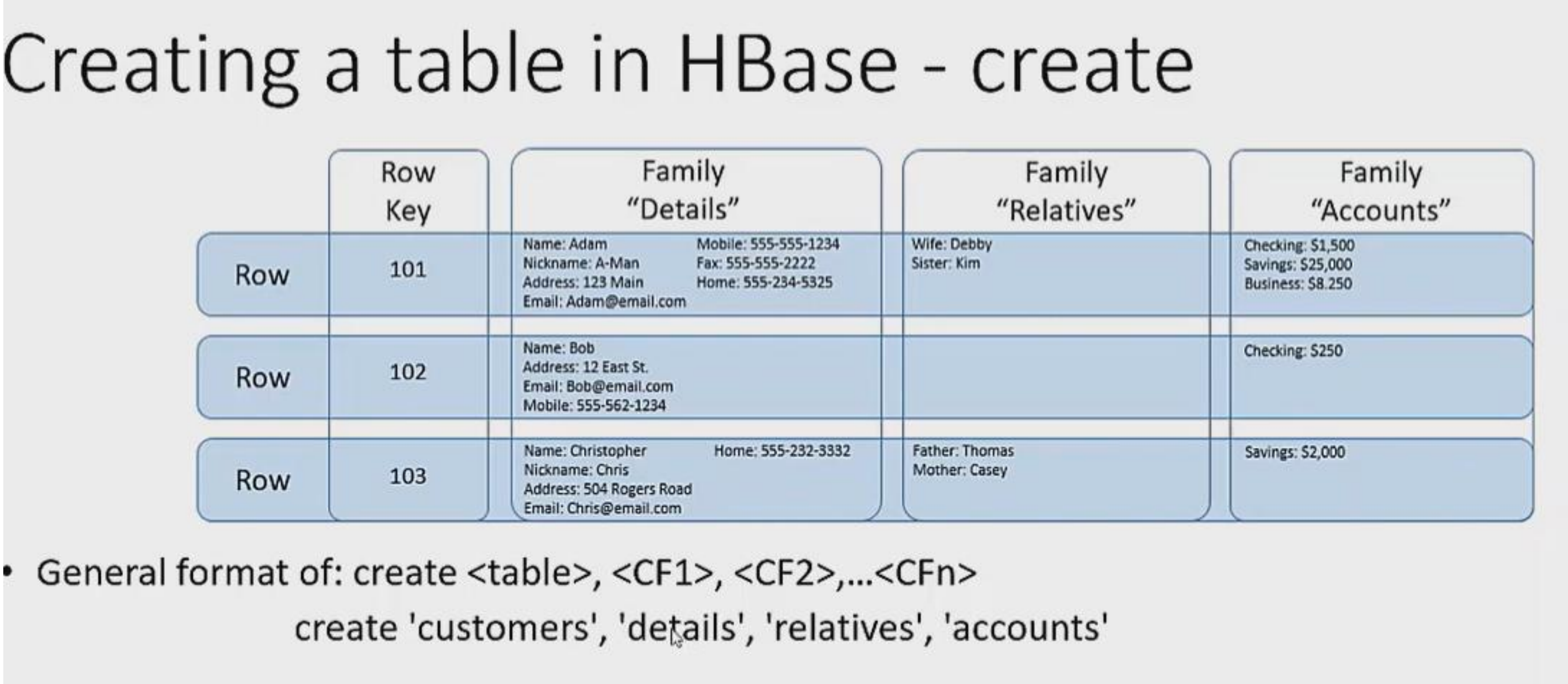
- Structure of HBASE table



	Row Key	Family “Details”	Family “Relatives”	Family “Accounts”
Row	<value>	Name Nickname Address Email Mobile Fax Home	Wife Sister	Checking Savings Business
Row	<value>	Name Address Email Mobile		Checking
Row	<value>	Name Nickname Address Email Home	Father Mother	Savings



	Row Key	Family “Details”	Family “Relatives”
Row	101	Name: Adam Nickname: A-Man Address: 123 Main Email: Adam@email.com Mobile: 555-555-1234 Fax: 555-555-2222 Home: 555-234-5325	Wife: Debby Sister: Kim
Row	102	Name: Bob Address: 12 East St. Email: Bob@email.com Mobile: 555-562-1234	
Row	103	Name: Christopher Nickname: Chris Address: 504 Rogers Road Email: Chris@email.com Home: 555-232-3332	Father: Thomas Mother: Casey

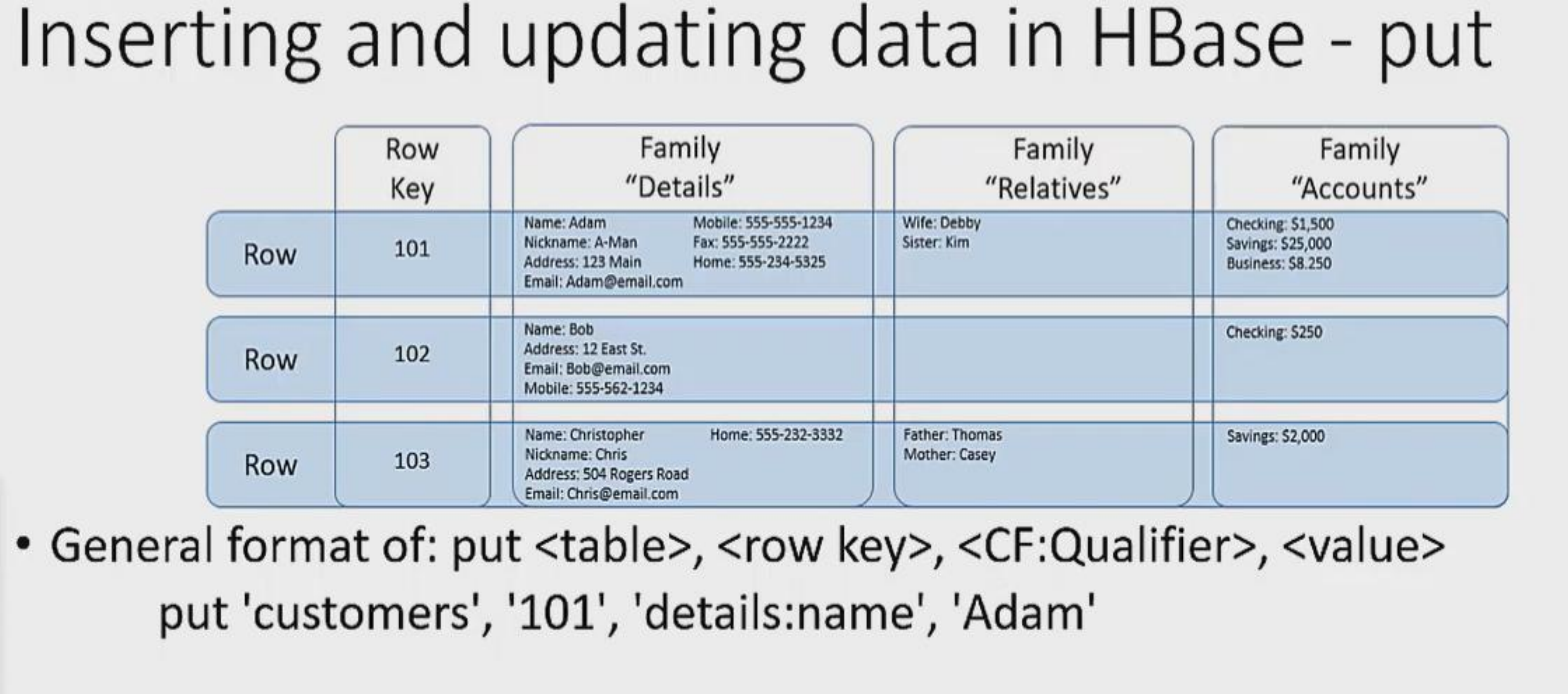


Creating a table in HBase - create

	Row Key	Family "Details"	Family "Relatives"	Family "Accounts"
Row	101	Name: Adam Nickname: A-Man Address: 123 Main Email: Adam@email.com Mobile: 555-555-1234 Fax: 555-555-2222 Home: 555-234-5325	Wife: Debby Sister: Kim	Checking: $1,500 Savings: $25,000 Business: $8.250
Row	102	Name: Bob Address: 12 East St. Email: Bob@email.com Mobile: 555-562-1234		Checking: $250
Row	103	Name: Christopher Nickname: Chris Address: 504 Rogers Road Email: Chris@email.com Home: 555-232-3332	Father: Thomas Mother: Casey	Savings: $2,000

- General format of: create <table>, <CF1>, <CF2>,…<CFn>

  create 'customers', 'details', 'relatives', 'accounts'

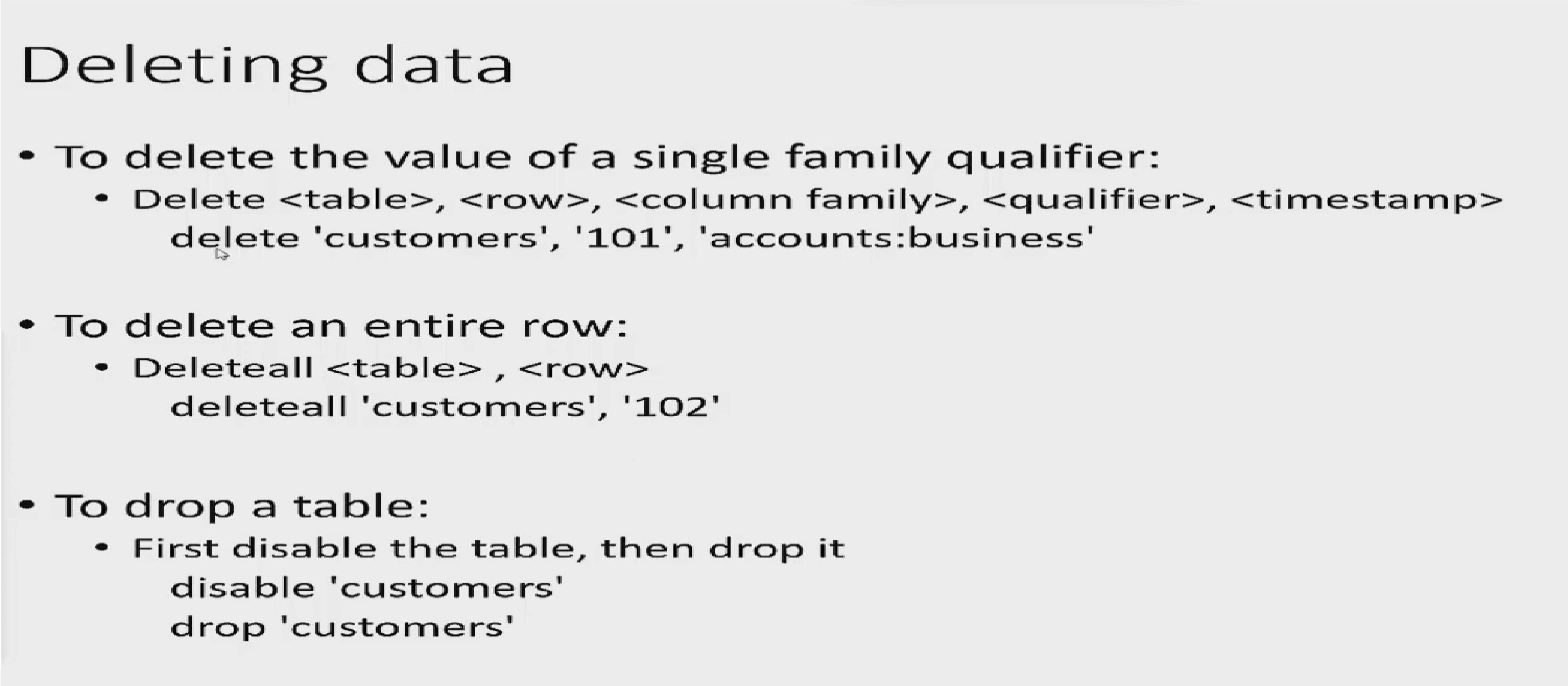


Inserting and updating data in HBase - put

	Row Key	Family "Details"	Family "Relatives"	Family "Accounts"
Row	101	Name: Adam Nickname: A-Man Address: 123 Main Email: Adam@email.com Mobile: 555-555-1234 Fax: 555-555-2222 Home: 555-234-5325	Wife: Debby Sister: Kim	Checking: $1,500 Savings: $25,000 Business: $8.250
Row	102	Name: Bob Address: 12 East St. Email: Bob@email.com Mobile: 555-562-1234		Checking: $250
Row	103	Name: Christopher Nickname: Chris Address: 504 Rogers Road Email: Chris@email.com Home: 555-232-3332	Father: Thomas Mother: Casey	Savings: $2,000

- General format of: put <table>, <row key>, <CF:Qualifier>, <value>

```
put 'customers', '101', 'details:name', 'Adam'
```



Deleting data

- To delete the value of a single family qualifier:
  - Delete <table>, <row>, <column family>, <qualifier>, <timestamp>

```
delete 'customers', '101', 'accounts:business'
```

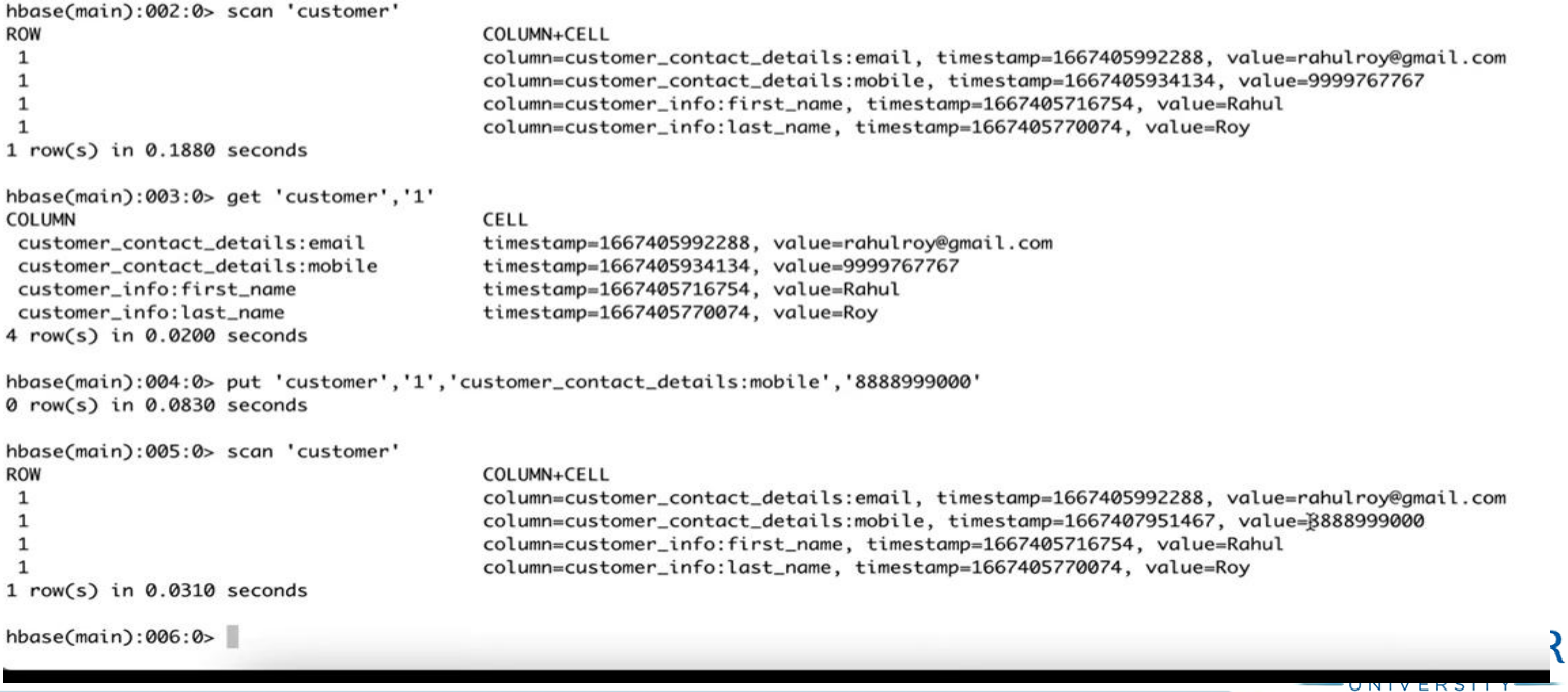
- To delete an entire row:
  - Deleteall <table> , <row>

```
deleteall 'customers', '102'
```

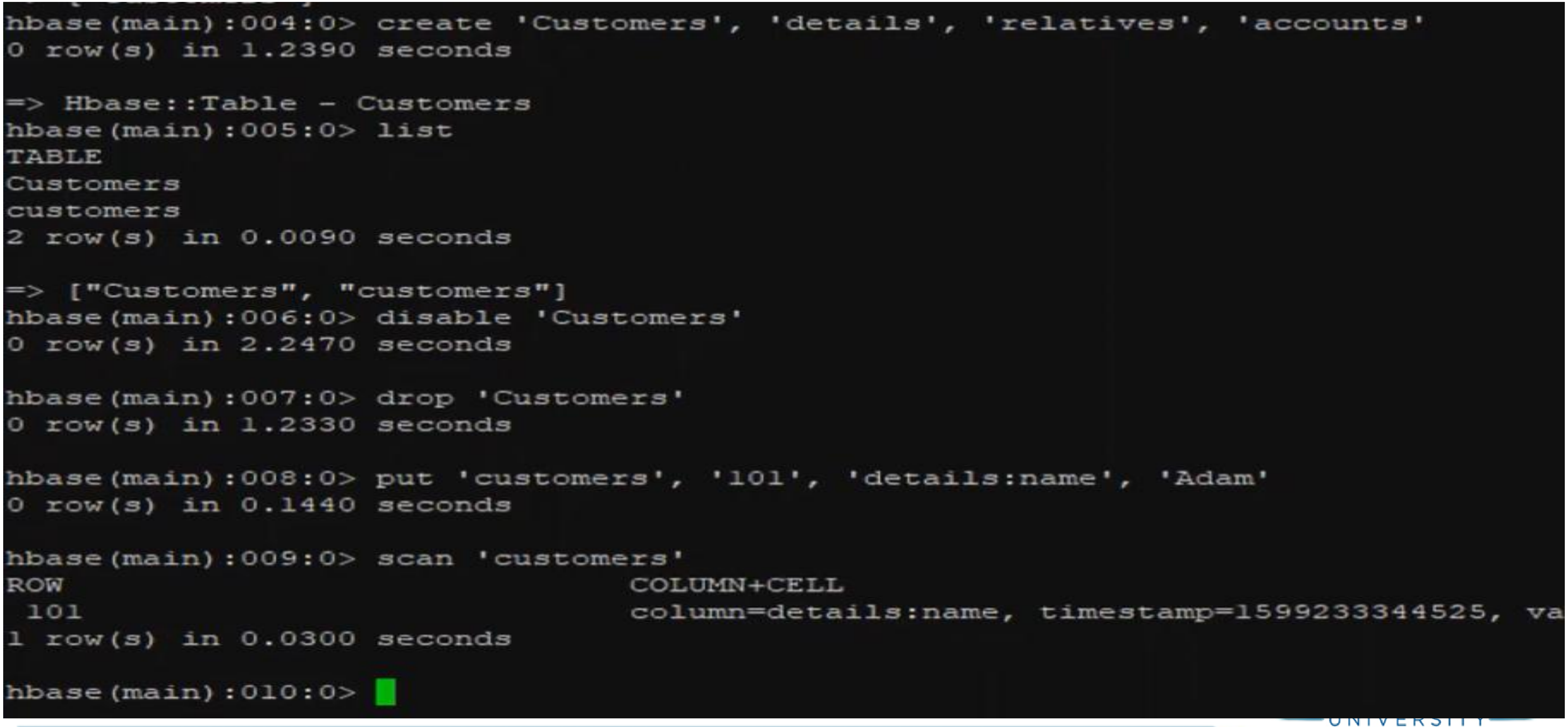
- To drop a table:
  - First disable the table, then drop it

```
disable 'customers'
drop 'customers'
```

- Update table

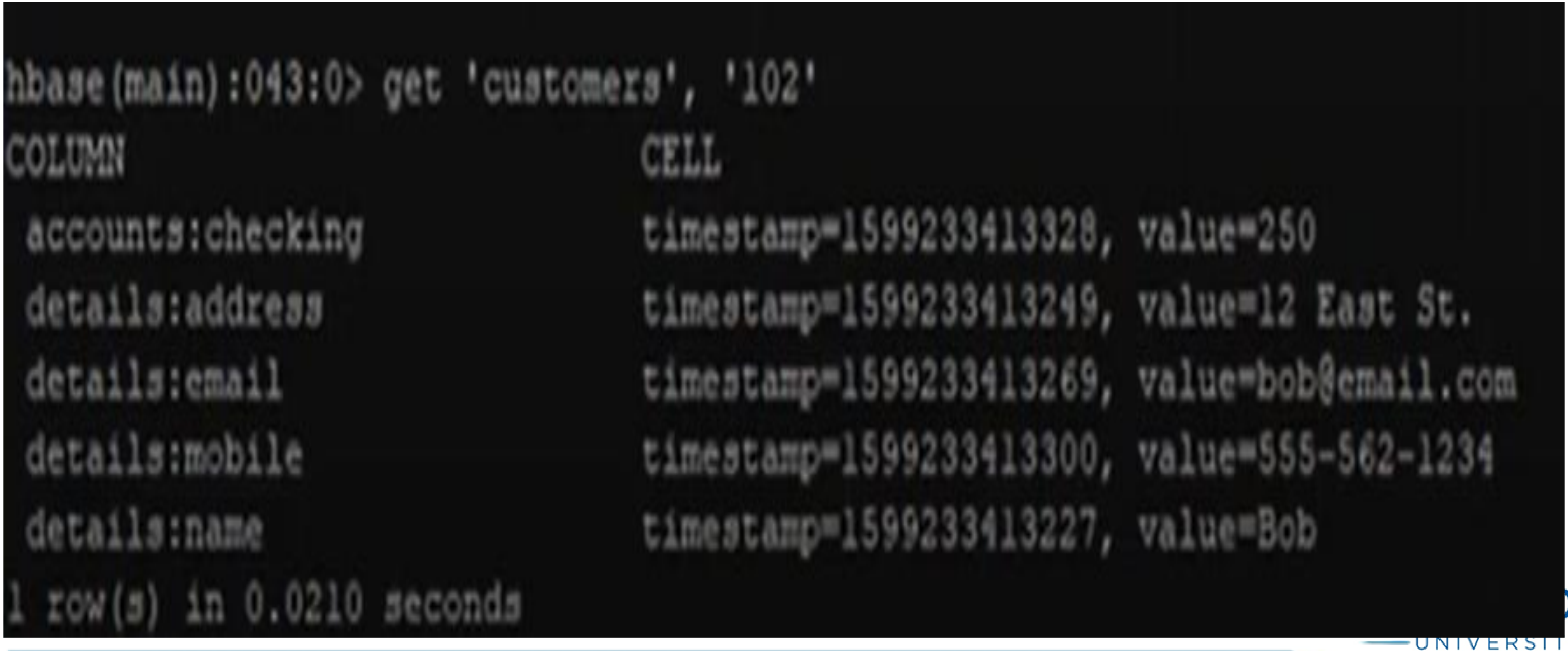


```
hbase(main):002:0> scan 'customer'
ROW                                       COLUMN+CELL
 1                                        column=customer_contact_details:email, timestamp=1667405992288, value=rahulroy@gmail.com
 1                                        column=customer_contact_details:mobile, timestamp=1667405934134, value=9999767767
 1                                        column=customer_info:first_name, timestamp=1667405716754, value=Rahul
 1                                        column=customer_info:last_name, timestamp=1667405770074, value=Roy
1 row(s) in 0.1880 seconds

hbase(main):003:0> get 'customer','1'
COLUMN                                    CELL
 customer_contact_details:email           timestamp=1667405992288, value=rahulroy@gmail.com
 customer_contact_details:mobile          timestamp=1667405934134, value=9999767767
 customer_info:first_name                 timestamp=1667405716754, value=Rahul
 customer_info:last_name                  timestamp=1667405770074, value=Roy
4 row(s) in 0.0200 seconds

hbase(main):004:0> put 'customer','1','customer_contact_details:mobile','8888999000'
0 row(s) in 0.0830 seconds

hbase(main):005:0> scan 'customer'
ROW                                       COLUMN+CELL
 1                                        column=customer_contact_details:email, timestamp=1667405992288, value=rahulroy@gmail.com
 1                                        column=customer_contact_details:mobile, timestamp=1667407951467, value=8888999000
 1                                        column=customer_info:first_name, timestamp=1667405716754, value=Rahul
 1                                        column=customer_info:last_name, timestamp=1667405770074, value=Roy
1 row(s) in 0.0310 seconds

hbase(main):006:0>
```

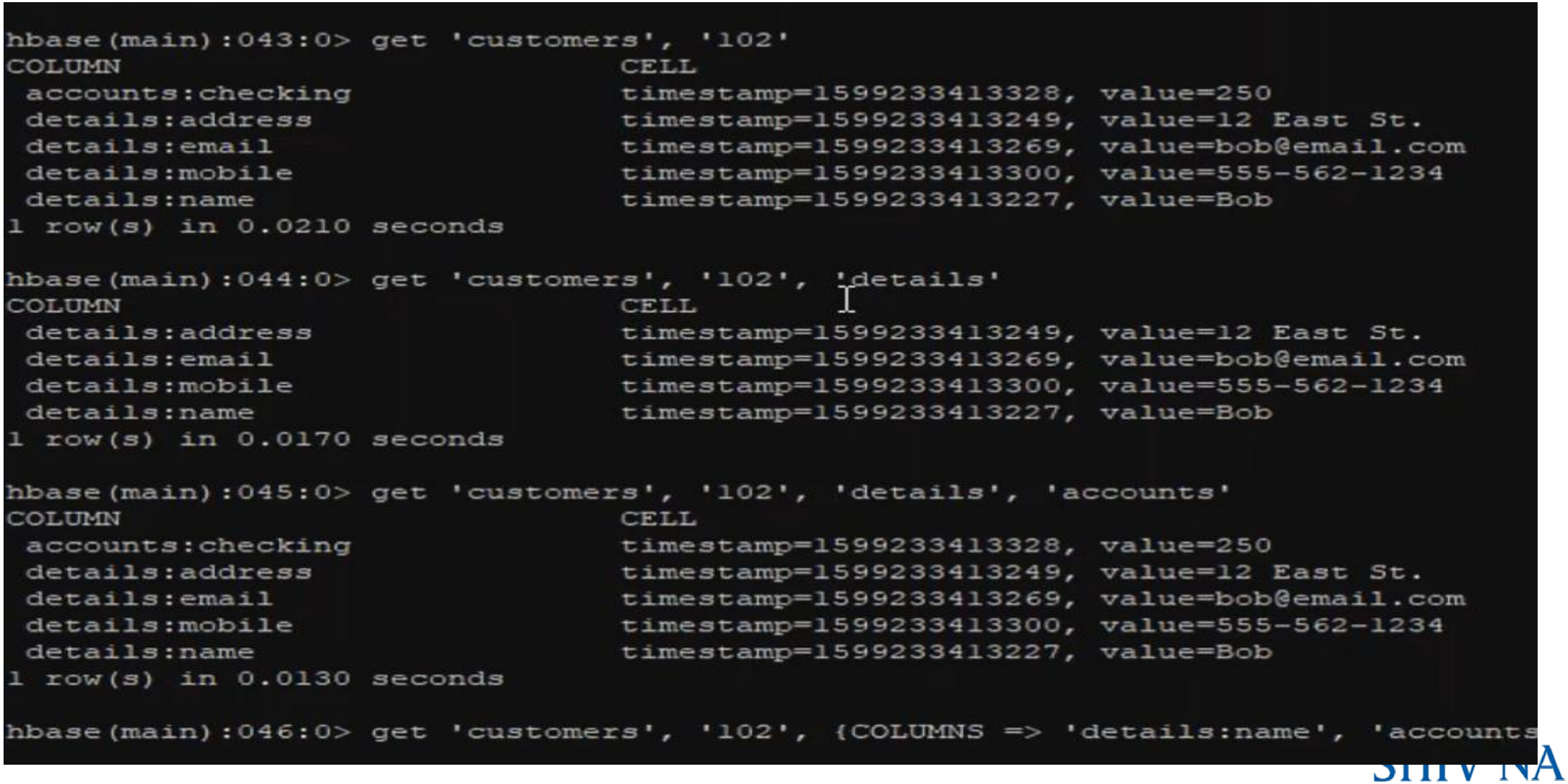


```
hbase(main):004:0> create 'Customers', 'details', 'relatives', 'accounts'
0 row(s) in 1.2390 seconds

=> Hbase::Table - Customers
hbase(main):005:0> list
TABLE
Customers
customers
2 row(s) in 0.0090 seconds

=> ["Customers", "customers"]
hbase(main):006:0> disable 'Customers'
0 row(s) in 2.2470 seconds

hbase(main):007:0> drop 'Customers'
0 row(s) in 1.2330 seconds

hbase(main):008:0> put 'customers', '101', 'details:name', 'Adam'
0 row(s) in 0.1440 seconds

hbase(main):009:0> scan 'customers'
ROW                              COLUMN+CELL
 101                             column=details:name, timestamp=159923344525, va
1 row(s) in 0.0300 seconds

hbase(main):010:0> 
```

- Get- used to display specific row of table
- Syntax: get ‘table_name’,’row_id’

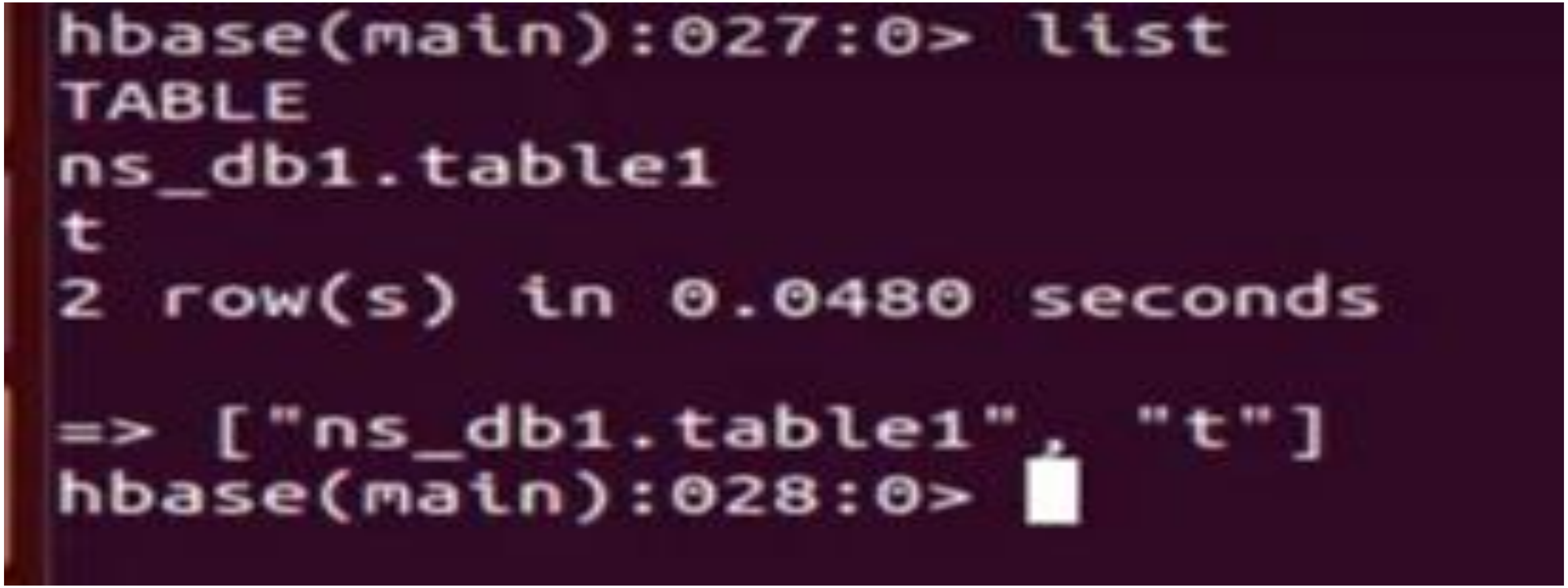


```
hbase(main):043:0> get 'customers', '102'
COLUMN                         CELL
 accounts:checking             timestamp=1599233413328, value=250
 details:address               timestamp=1599233413249, value=12 East St.
 details:email                 timestamp=1599233413269, value=bob@email.com
 details:mobile                timestamp=1599233413300, value=555-562-1234
 details:name                  timestamp=1599233413227, value=Bob
1 row(s) in 0.0210 seconds
```



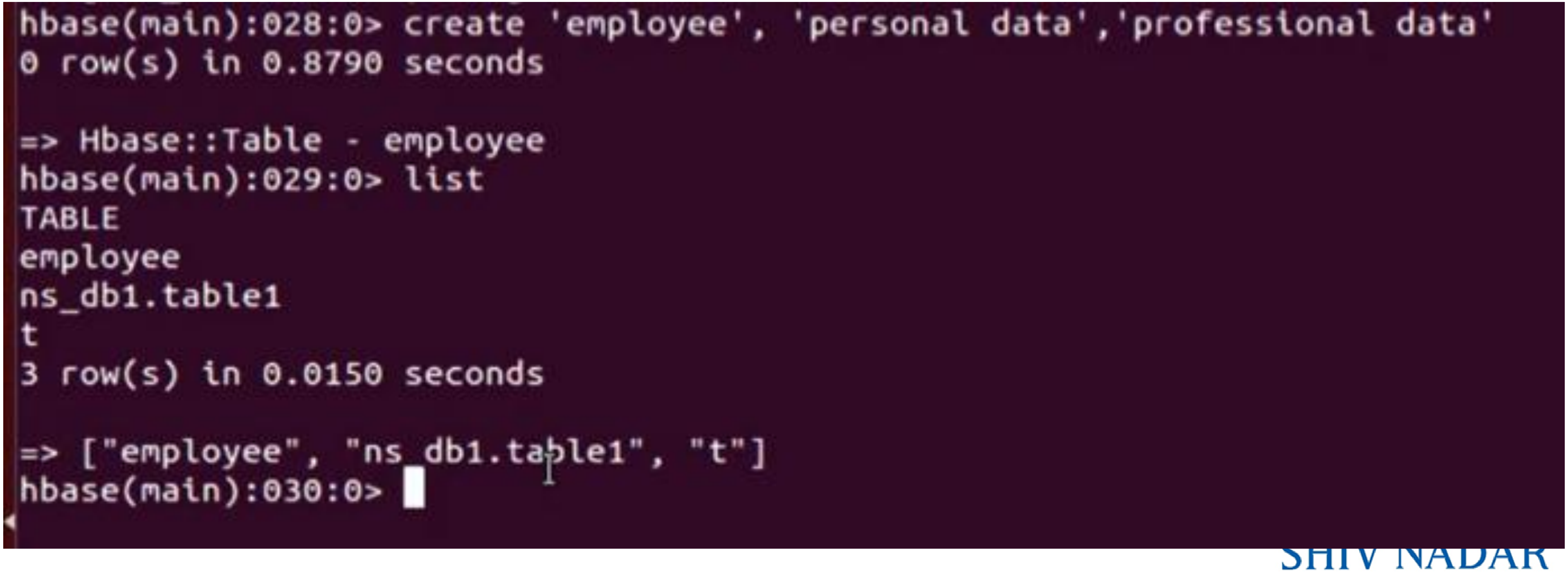
```
hbase(main):043:0> get 'customers', '102'
COLUMN                          CELL
 accounts:checking              timestamp=1599233413328, value=250
 details:address                timestamp=1599233413249, value=12 East St.
 details:email                  timestamp=1599233413269, value=bob@email.com
 details:mobile                 timestamp=1599233413300, value=555-562-1234
 details:name                   timestamp=1599233413227, value=Bob
1 row(s) in 0.0210 seconds

hbase(main):044:0> get 'customers', '102', 'details'
COLUMN                          CELL
 details:address                timestamp=1599233413249, value=12 East St.
 details:email                  timestamp=1599233413269, value=bob@email.com
 details:mobile                 timestamp=1599233413300, value=555-562-1234
 details:name                   timestamp=1599233413227, value=Bob
1 row(s) in 0.0170 seconds

hbase(main):045:0> get 'customers', '102', 'details', 'accounts'
COLUMN                          CELL
 accounts:checking              timestamp=1599233413328, value=250
 details:address                timestamp=1599233413249, value=12 East St.
 details:email                  timestamp=1599233413269, value=bob@email.com
 details:mobile                 timestamp=1599233413300, value=555-562-1234
 details:name                   timestamp=1599233413227, value=Bob
1 row(s) in 0.0130 seconds

hbase(main):046:0> get 'customers', '102', {COLUMNS => 'details:name', 'accounts
```

- List- display all list of table available



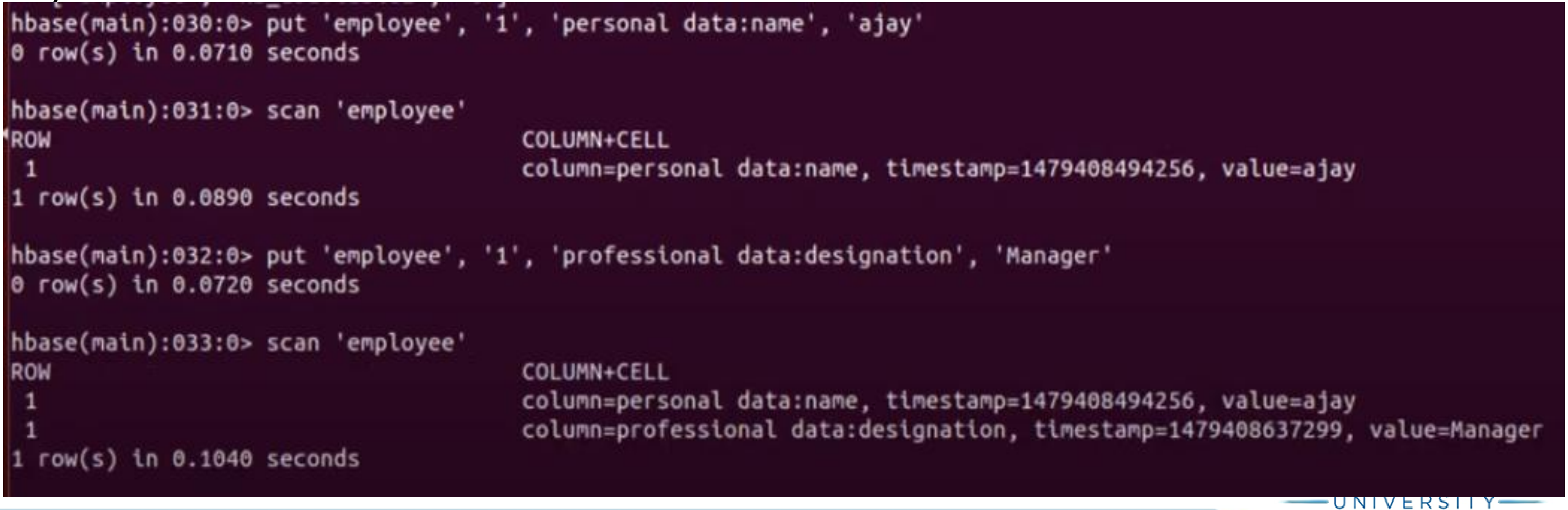
```
hbase(main):027:0> list
TABLE
ns_db1.table1
t
2 row(s) in 0.0480 seconds

=> ["ns_db1.table1", "t"]
hbase(main):028:0> 
```

- Create – used to create a new table
- Syntax: create ‘table_name’, ‘colum_name’,…’column_name’

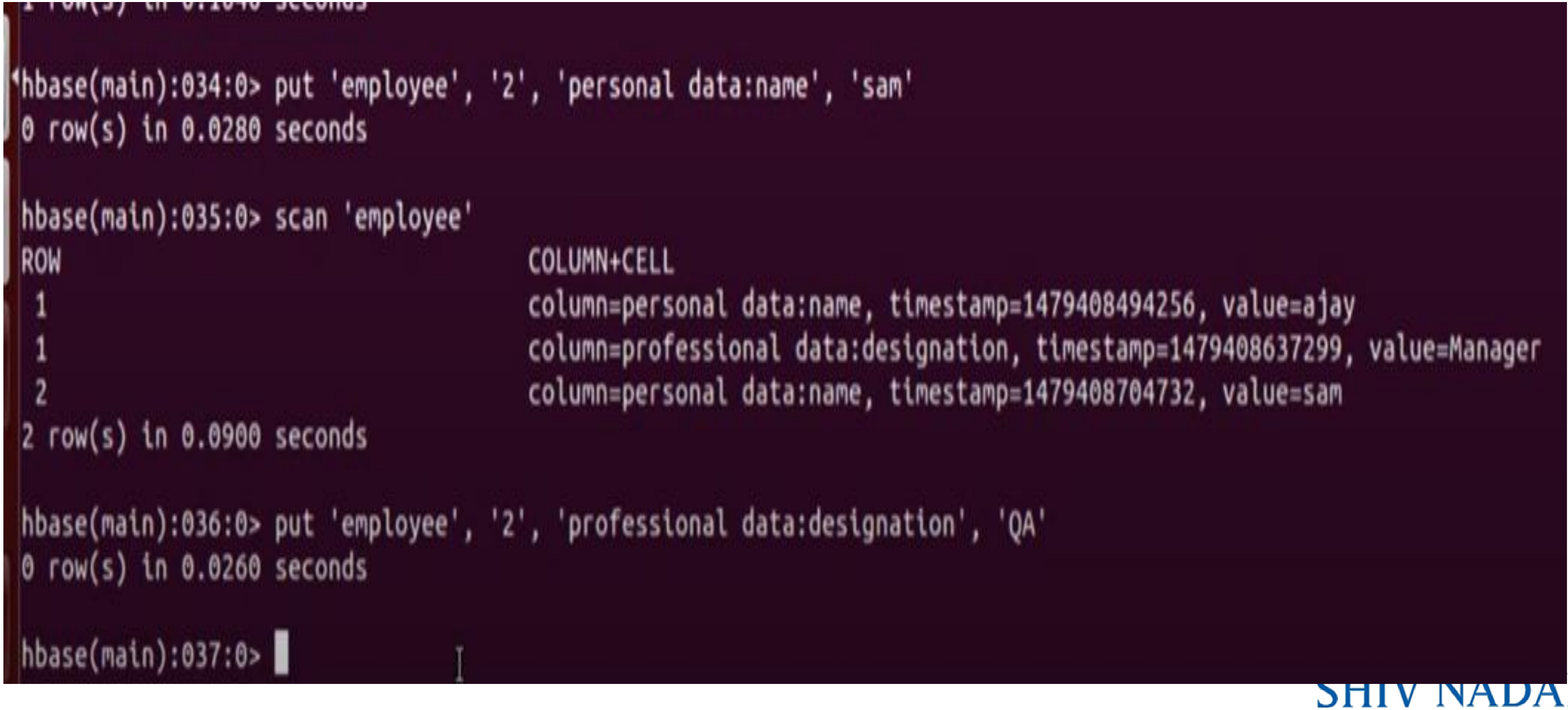


```
hbase(main):028:0> create 'employee', 'personal data','professional data'
0 row(s) in 0.8790 seconds

=> Hbase::Table - employee
hbase(main):029:0> list
TABLE
employee
ns_db1.table1
t
3 row(s) in 0.0150 seconds

=> ["employee", "ns_db1.table1", "t"]
hbase(main):030:0> 
```

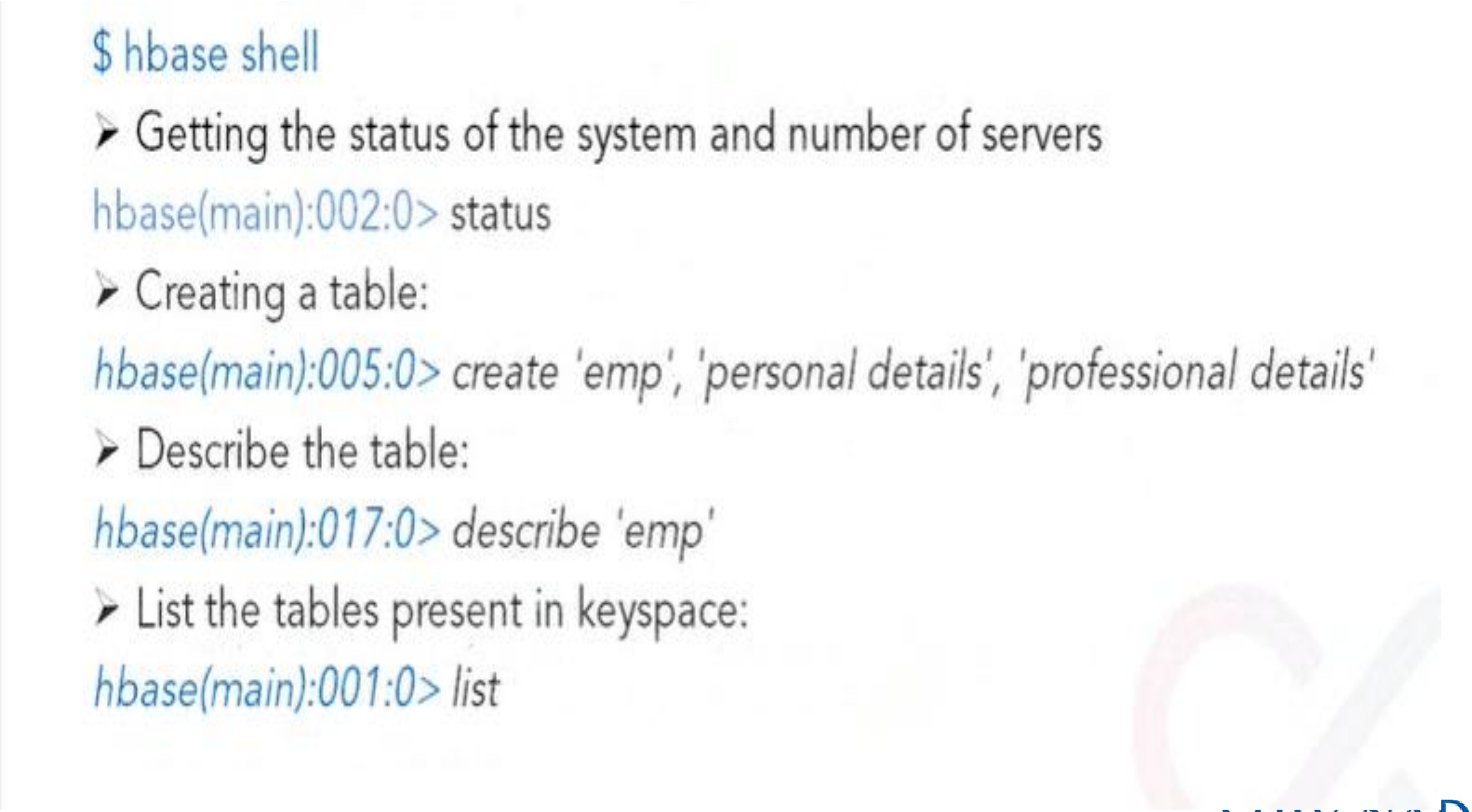
- Put- Used to insert a record into the tables
- Syntax: put ‘table_name’, ‘id’,’column_name:attribute’,’value’
- Scan- used to display table data
- Syntax: scan ‘table_name’



```
hbase(main):030:0> put 'employee', '1', 'personal data:name', 'ajay'
0 row(s) in 0.0710 seconds

hbase(main):031:0> scan 'employee'
ROW                                   COLUMN+CELL
 1                                    column=personal data:name, timestamp=1479408494256, value=ajay
1 row(s) in 0.0890 seconds

hbase(main):032:0> put 'employee', '1', 'professional data:designation', 'Manager'
0 row(s) in 0.0720 seconds

hbase(main):033:0> scan 'employee'
ROW                                   COLUMN+CELL
 1                                    column=personal data:name, timestamp=1479408494256, value=ajay
 1                                    column=professional data:designation, timestamp=1479408637299, value=Manager
1 row(s) in 0.1040 seconds
```



```
1 row(s) in 0.1040 seconds

hbase(main):034:0> put 'employee', '2', 'personal data:name', 'sam'
0 row(s) in 0.0280 seconds

hbase(main):035:0> scan 'employee'
ROW                                COLUMN+CELL
 1                                 column=personal data:name, timestamp=1479408494256, value=ajay
 1                                 column=professional data:designation, timestamp=1479408637299, value=Manager
 2                                 column=personal data:name, timestamp=1479408704732, value=sam
2 row(s) in 0.0900 seconds

hbase(main):036:0> put 'employee', '2', 'professional data:designation', 'QA'
0 row(s) in 0.0260 seconds

hbase(main):037:0> 
```



```
$ hbase shell
```

- Getting the status of the system and number of servers

```
hbase(main):002:0> status
```

- Creating a table:

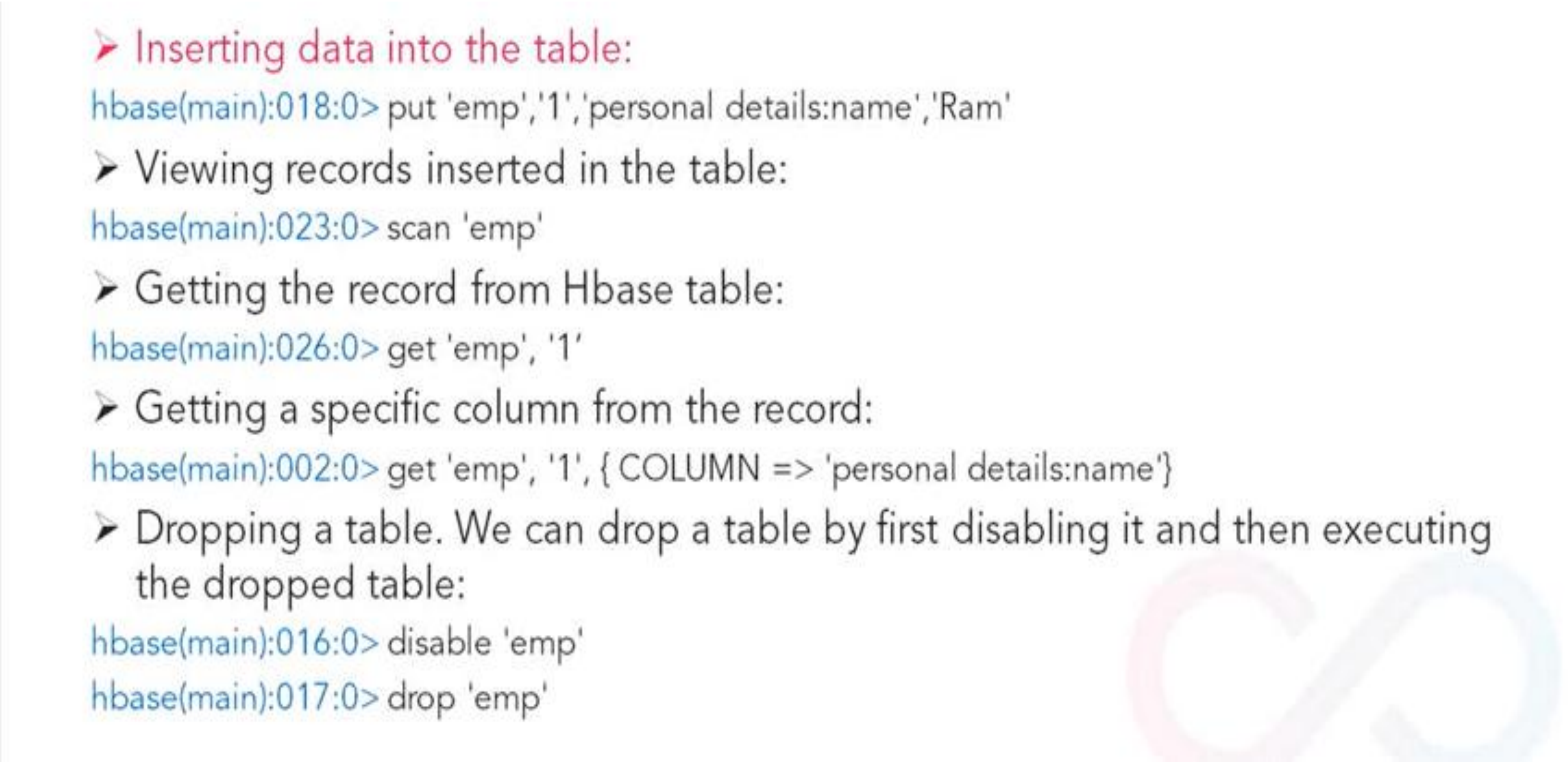
```
hbase(main):005:0> create 'emp', 'personal details', 'professional details'
```

- Describe the table:

```
hbase(main):017:0> describe 'emp'
```

- List the tables present in keyspace:

```
hbase(main):001:0> list
```



- Inserting data into the table:

```
hbase(main):018:0> put 'emp','1','personal details:name','Ram'
```

- Viewing records inserted in the table:

```
hbase(main):023:0> scan 'emp'
```

- Getting the record from Hbase table:

```
hbase(main):026:0> get 'emp', '1'
```

- Getting a specific column from the record:

```
hbase(main):002:0> get 'emp', '1', { COLUMN => 'personal details:name'}
```

- Dropping a table. We can drop a table by first disabling it and then executing the dropped table:

```
hbase(main):016:0> disable 'emp'
hbase(main):017:0> drop 'emp'
```