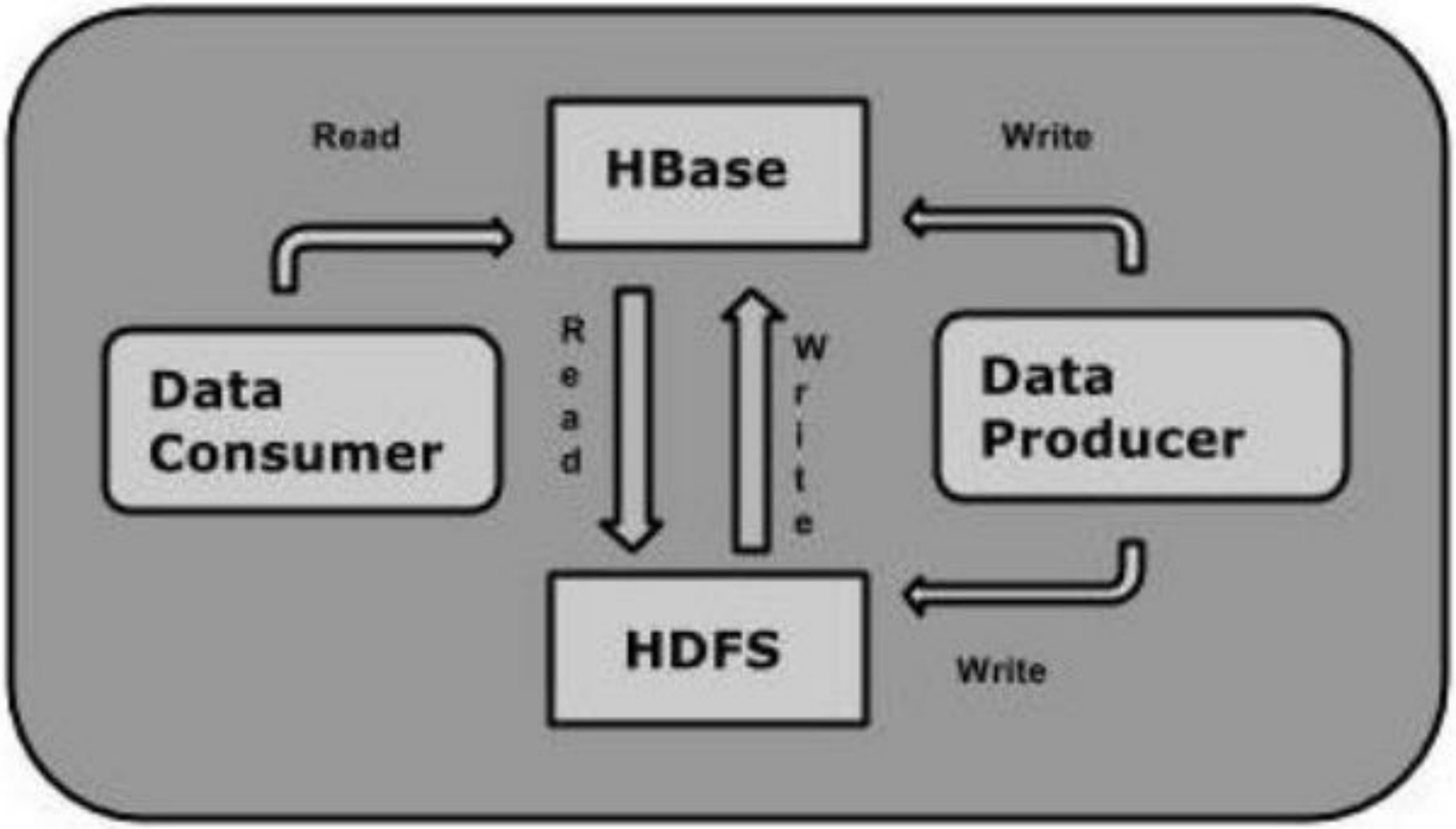
Unit 4 – Part 2

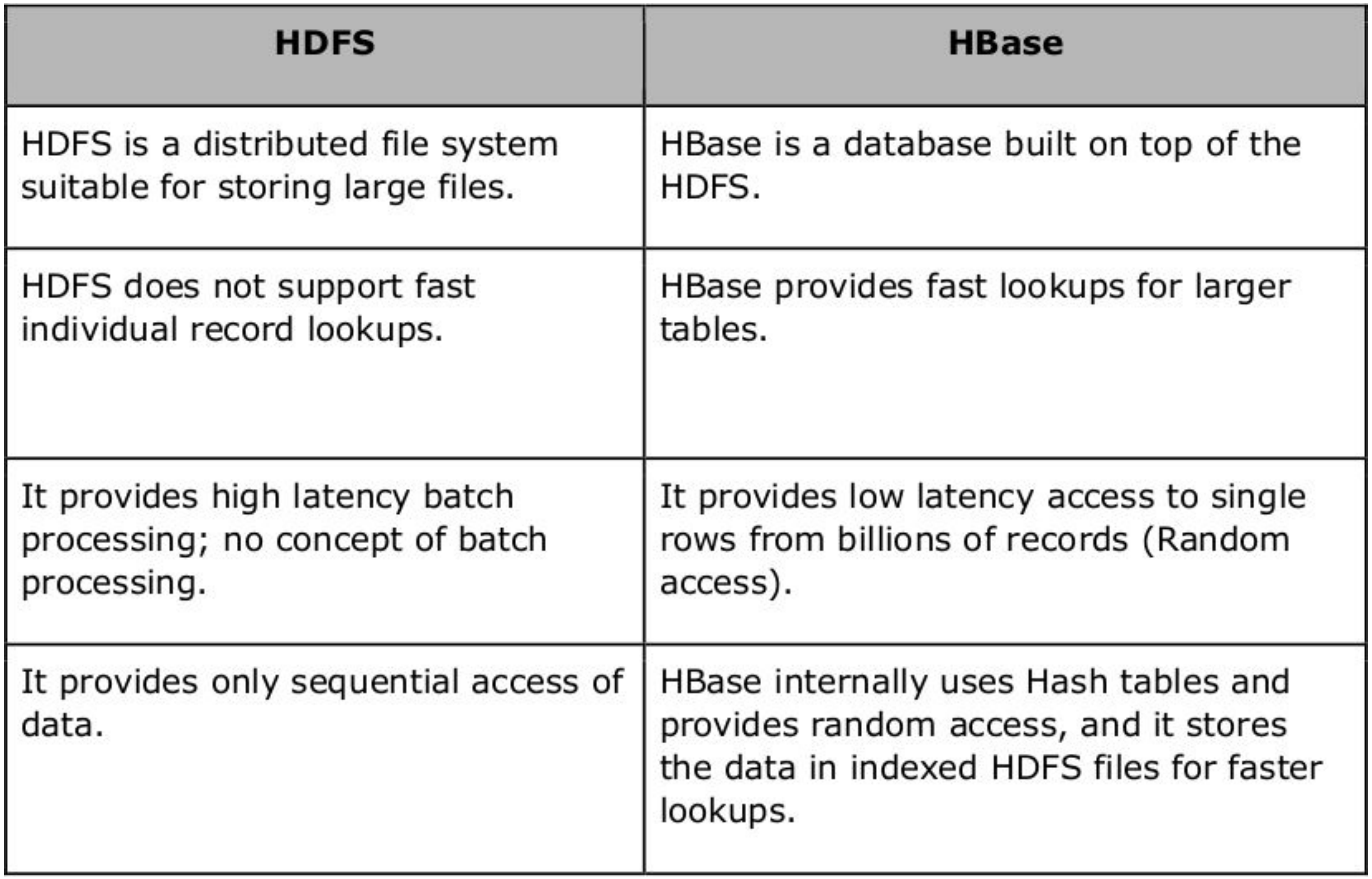
What is HBase?

- 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.
- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).
- It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.
- One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

HBase


Read
HBase
Write
Data Consumer
Read
Write
Data Producer
HDFS
Write

HDFS vs. HBase

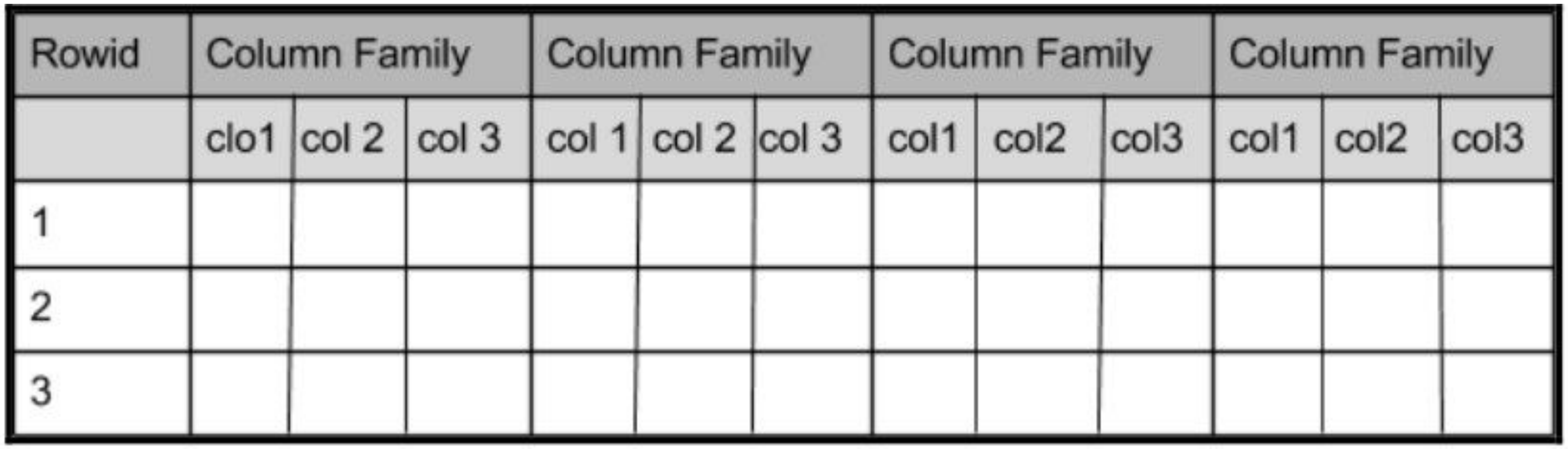


HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	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.

Storage Mechanism

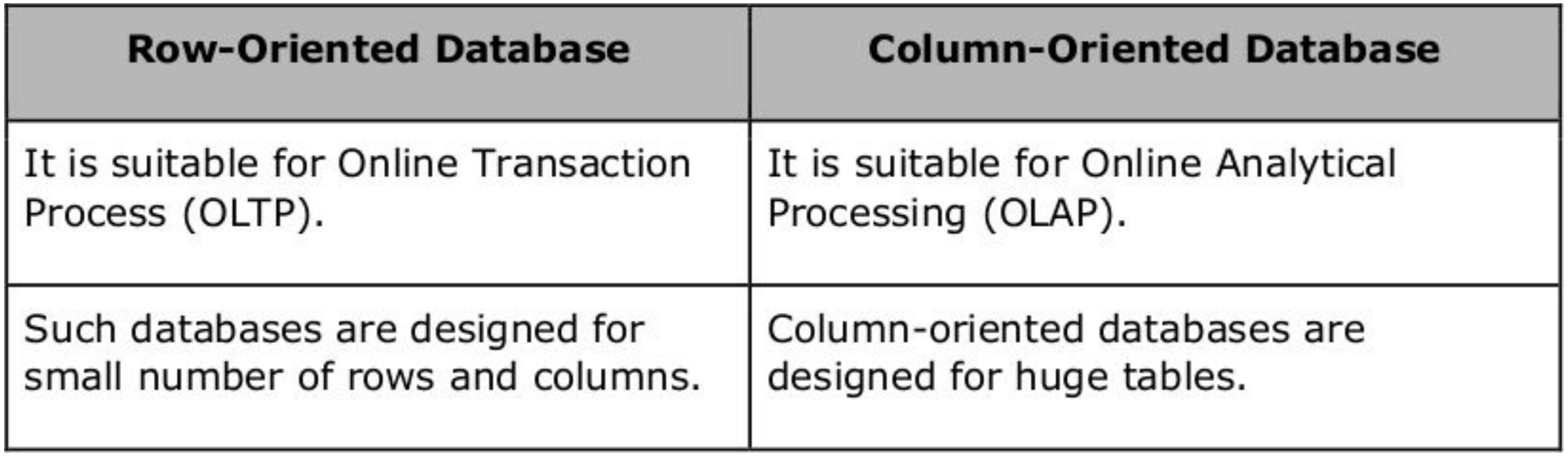
- HBase is a column-oriented database and the tables in it are sorted by row.
- The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns.
- Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an Hbase:
  - Table is a collection of rows.
  - Row is a collection of column families.
  - Column family is a collection of columns.
  - Column is a collection of key value pairs.

Storage Mechanism



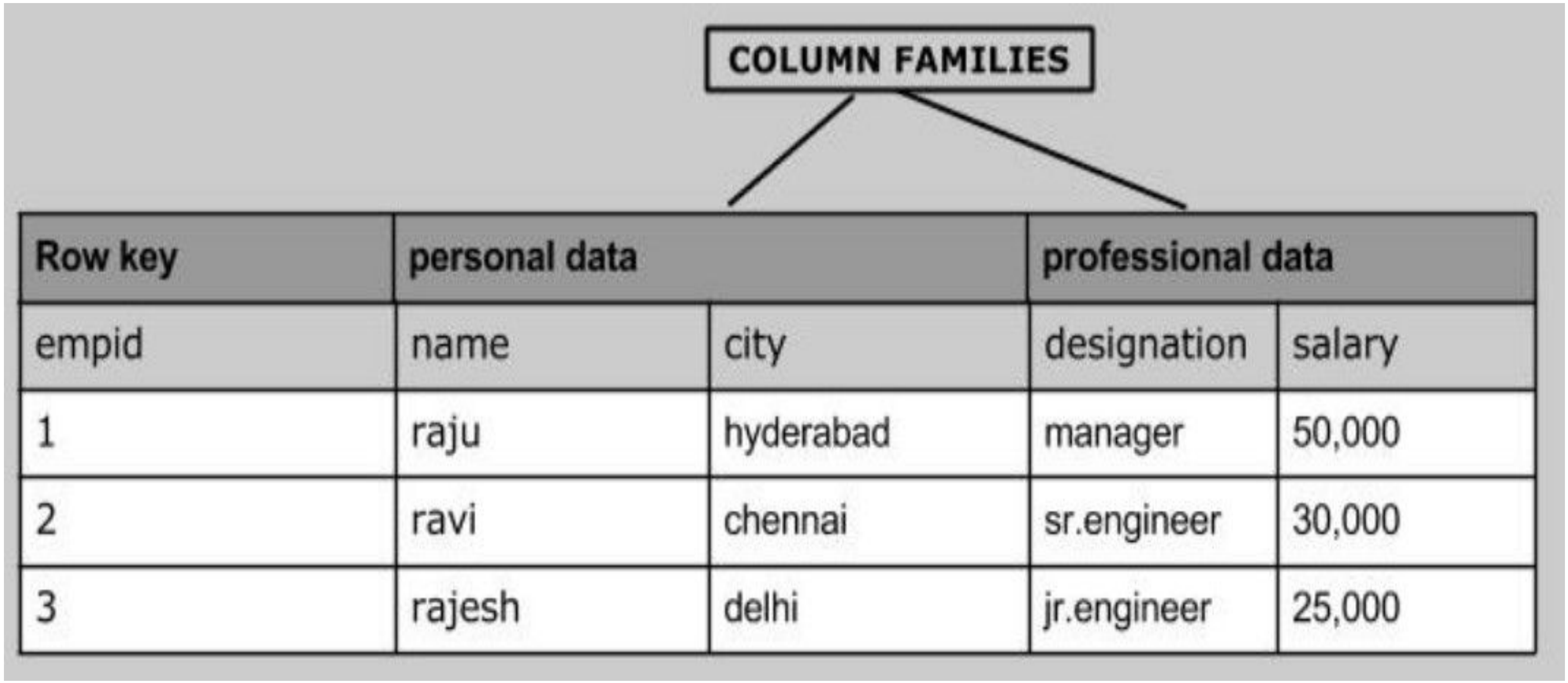
Rowid	Column Family			Column Family			Column Family			Column Family		
	clo1	col 2	col 3	col 1	col 2	col 3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Column Oriented vs. Row Oriented



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.

Example:


COLUMN FAMILIES

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

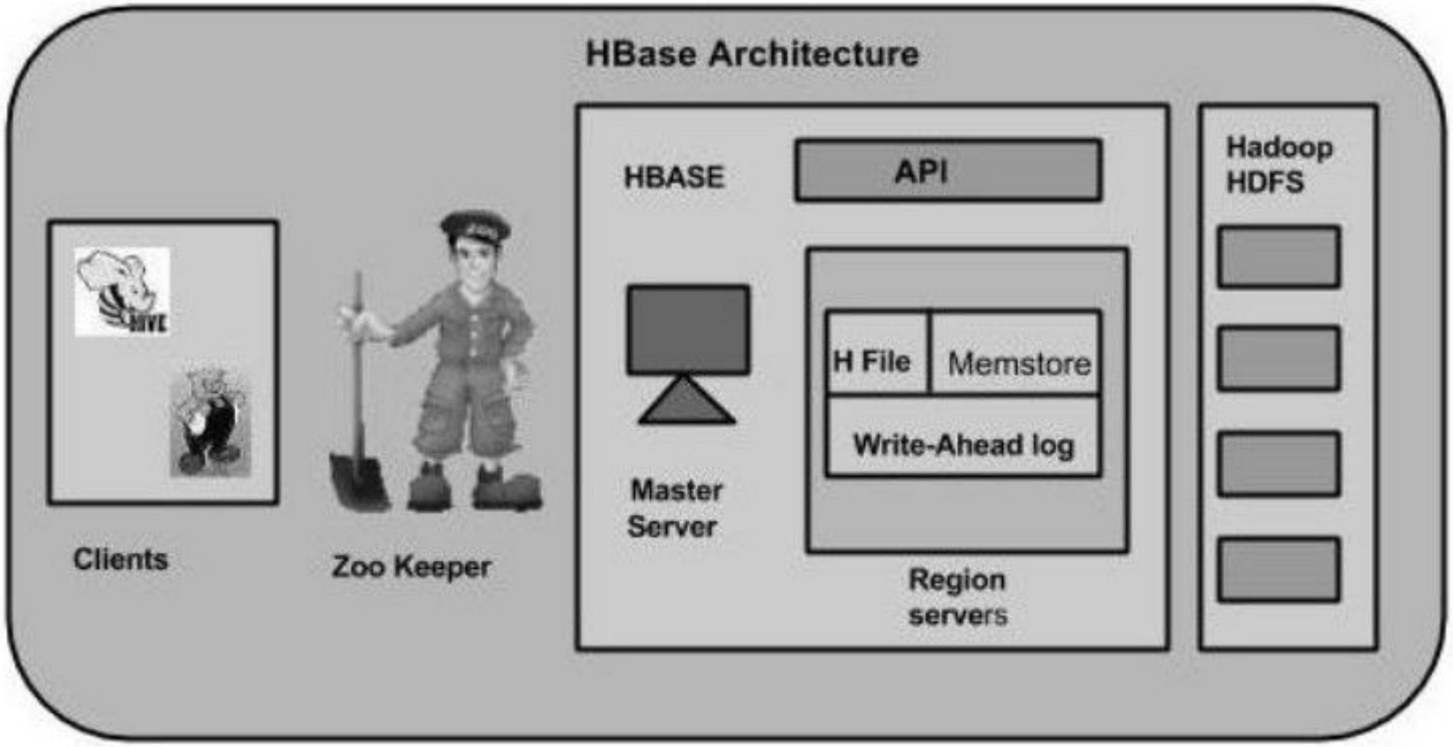
Features

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Uses

- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use Hbase internally.

Architecture



Master server

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Region server

- Regions
- Regions are nothing but tables that are split up and spread across the region servers.
- Region server
  - The region servers have regions that -
  - Communicate with the client and handle data-related operations. Handle read and write requests for all the regions under it.
  - Decide the size of the region by following the region size thresholds.

Zookeeper

- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

Hbase Shell

- HBase contains a shell using which you can communicate with HBase.
- Hbase uses the Hadoop File System to store its data.
- It will have a master server and region servers. The data storage will be in the form of regions (tables).
- These regions will be split up and stored in region servers.

General Commands

- status: Provides the status of HBase, for example, the number of servers.
- version: Provides the version of HBase being used.
- table_help: Provides help for table-reference commands.
- whoami: Provides information about the user.

DDL Commands

- create: Creates a table.
- list: Lists all the tables in HBase.
- disable: Disables a table.
- is_disabled: Verifies whether a table is disabled.
- enable: Enables a table.
- is_enabled: Verifies whether a table is enabled.
- describe: Provides the description of a table. alter:
- Alters a table.
- exists: Verifies whether a table exists.
- drop: Drops a table from HBase.

DML Commands

- put: Puts a cell value at a specified column in a specified row in a particular table.
- get: Fetches the contents of row or a cell. delete: Deletes a cell value in a table.
- deleteall: Deletes all the cells in a given row. scan: Scans and returns the table data.
- count: Counts and returns the number of rows in a table.
- truncate: Disables, drops, and recreates a specified table.

Create table

Row key	personal data	professional data

- create 'emp', 'personal data', ‘professional data’

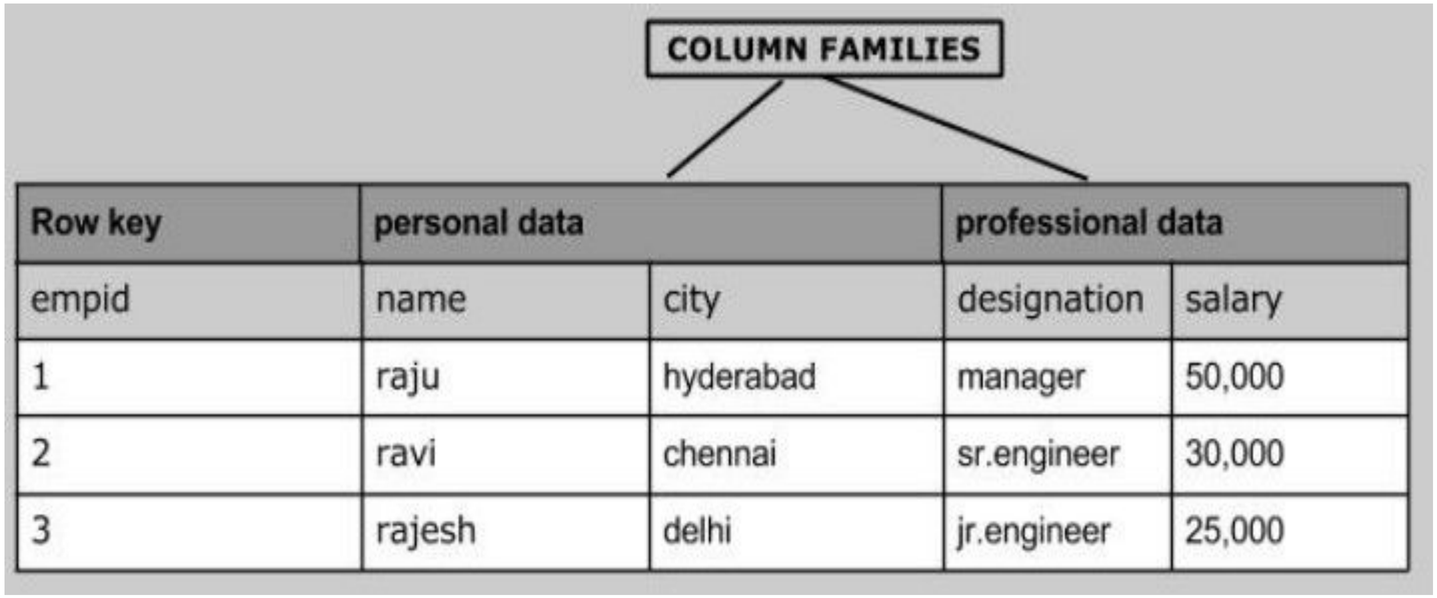
Describe table

- describe ‘tablename’

Create data

- To create data in an HBase table, the following commands and methods are used:
  - put command,
  - add() method of Put class, and
  - put() method of HTable class.

Example:


COLUMN FAMILIES

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

Example:

- put 'emp','1','personal data:name','raju'
- put 'emp','1','personal data:city','mumbai'
- put 'emp','1','professional data:designation','manager'
- put 'emp','1','professional data:salary','50000'
-

Display:

- scan 'tablename'

Update:

- put 'emp',1 ,'personal:city','Delhi'

Read Data:

- get 'emp', '1'
- get 'emp', '1', {COLUMN=>'personal:name'}

Read Data:

- delete 'emp', '1', 'personal data:city'

Count and truncate

- You can count the number of rows of a table using the count command.

  count ‘emp’
- This command disables drops and recreates a table.

  truncate ‘tablename’

Examples of Other Key-Value Stores

- Oracle key-value store
  - Oracle NOSQL Database
- Redis key-value cache and store
  - Caches data in main memory to improve performance
  - Offers master-slave replication and high availability
  - Offers persistence by backing up cache to disk
- Apache Cassandra
  - Offers features from several NOSQL categories
  - Used by Facebook and others

24.5 Column-Based or Wide Column NOSQL Systems

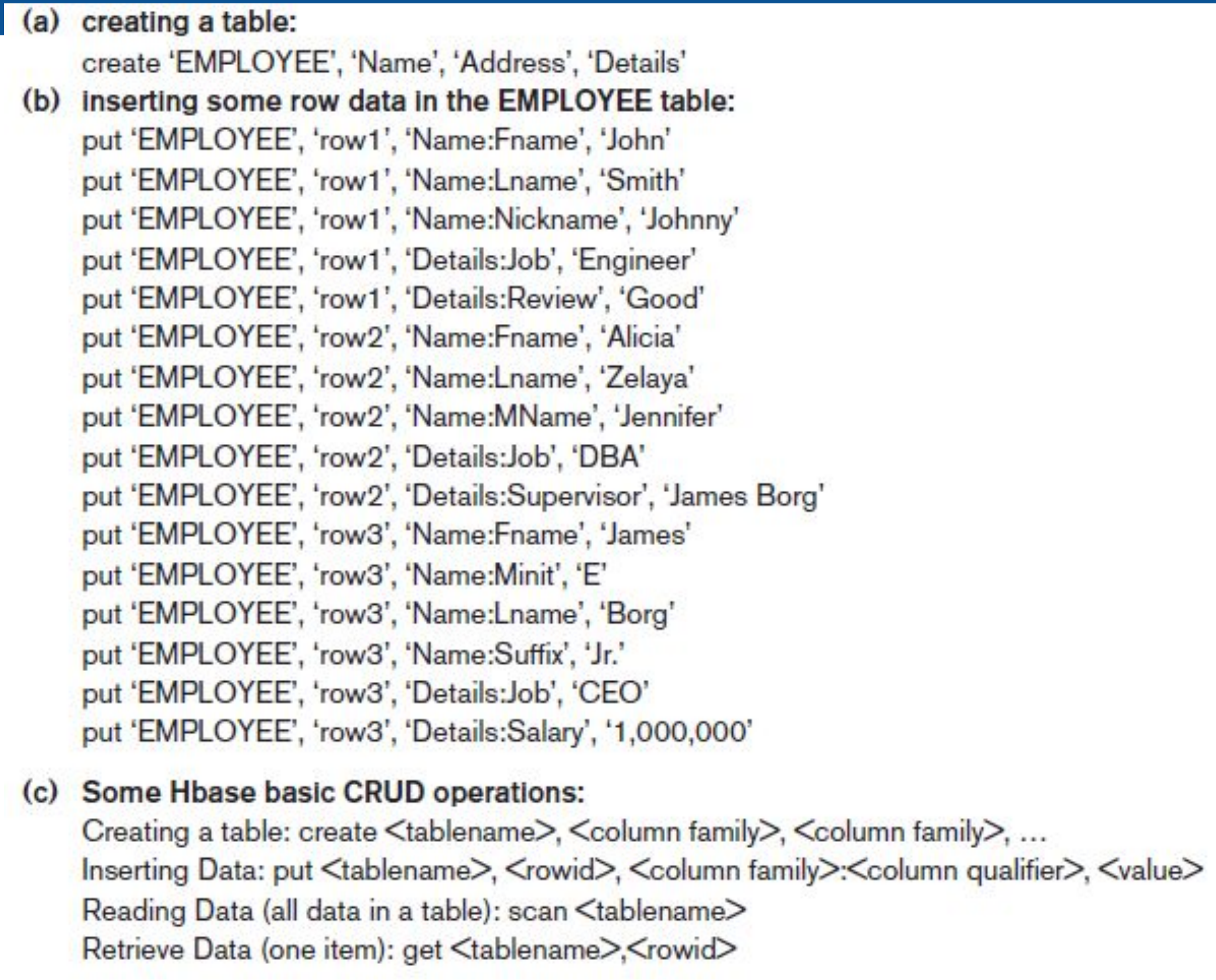
- BigTable: Google’s distributed storage system for big data
    - Used in Gmail
    - Uses Google File System for data storage and distribution
- Apache Hbase a similar, open source system
    - Uses Hadoop Distributed File System (HDFS) for data storage
    - Can also use Amazon’s Simple Storage System (S3)

Hbase Data Model and Versioning

- Data organization concepts
  - Namespaces
  - Tables
  - Column families
  - Column qualifiers
  - Columns
  - Rows
  - Data cells
- Data is self-describing

Hbase Data Model and Versioning (cont'd.)

- HBase stores multiple versions of data items
    - Timestamp associated with each version
- Each row in a table has a unique row key
- Table associated with one or more column families
- Column qualifiers can be dynamically specified as new table rows are created and inserted
- Namespace is collection of tables
- Cell holds a basic data item


(a) **creating a table:**

```
create 'EMPLOYEE', 'Name', 'Address', 'Details'
```

(b) **inserting some row data in the EMPLOYEE table:**

```
put 'EMPLOYEE', 'row1', 'Name:Fname', 'John'
put 'EMPLOYEE', 'row1', 'Name:Lname', 'Smith'
put 'EMPLOYEE', 'row1', 'Name:Nickname', 'Johnny'
put 'EMPLOYEE', 'row1', 'Details:Job', 'Engineer'
put 'EMPLOYEE', 'row1', 'Details:Review', 'Good'
put 'EMPLOYEE', 'row2', 'Name:Fname', 'Alicia'
put 'EMPLOYEE', 'row2', 'Name:Lname', 'Zelaya'
put 'EMPLOYEE', 'row2', 'Name:MName', 'Jennifer'
put 'EMPLOYEE', 'row2', 'Details:Job', 'DBA'
put 'EMPLOYEE', 'row2', 'Details:Supervisor', 'James Borg'
put 'EMPLOYEE', 'row3', 'Name:Fname', 'James'
put 'EMPLOYEE', 'row3', 'Name:Minit', 'E'
put 'EMPLOYEE', 'row3', 'Name:Lname', 'Borg'
put 'EMPLOYEE', 'row3', 'Name:Suffix', 'Jr.'
put 'EMPLOYEE', 'row3', 'Details:Job', 'CEO'
put 'EMPLOYEE', 'row3', 'Details:Salary', '1,000,000'
```

(c) **Some Hbase basic CRUD operations:**

```
Creating a table: create <tablename>, <column family>, <column family>, …
Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
Reading Data (all data in a table): scan <tablename>
Retrieve Data (one item): get <tablename>,<rowid>
```

Figure 24.3 Examples in Hbase (a) Creating a table called EMPLOYEE with three column families: Name, Address, and Details (b) Inserting some in the EMPLOYEE table; different rows can have different self-describing column qualifiers (Fname, Lname, Nickname, Mname, Minit, Suffix, … for column family Name; Job, Review, Supervisor, Salary for column family Details). (c) Some CRUD operations of Hbase

Hbase Crud Operations

- Provides only low-level CRUD (create, read, update, delete) operations
- Application programs implement more complex operations
- Create
  - Creates a new table and specifies one or more column families associated with the table
- Put
  - Inserts new data or new versions of existing data items

Hbase Crud Operations (cont'd.)

- Get
  - Retrieves data associated with a single row
- Scan
  - Retrieves all the rows

Hbase Storage and Distributed System Concepts

- Each Hbase table divided into several regions
  - Each region holds a range of the row keys in the table
  - Row keys must be lexicographically ordered
  - Each region has several stores
    - Column families are assigned to stores
- Regions assigned to region servers for storage
  - Master server responsible for monitoring the region servers
- Hbase uses Apache Zookeeper and HDFS

24.6 NOSQL Graph Databases and Neo4j

- Graph databases
  - Data represented as a graph
  - Collection of vertices (nodes) and edges
  - Possible to store data associated with both individual nodes and individual edges
- Neo4j
  - Open source system
  - Uses concepts of nodes and relationships

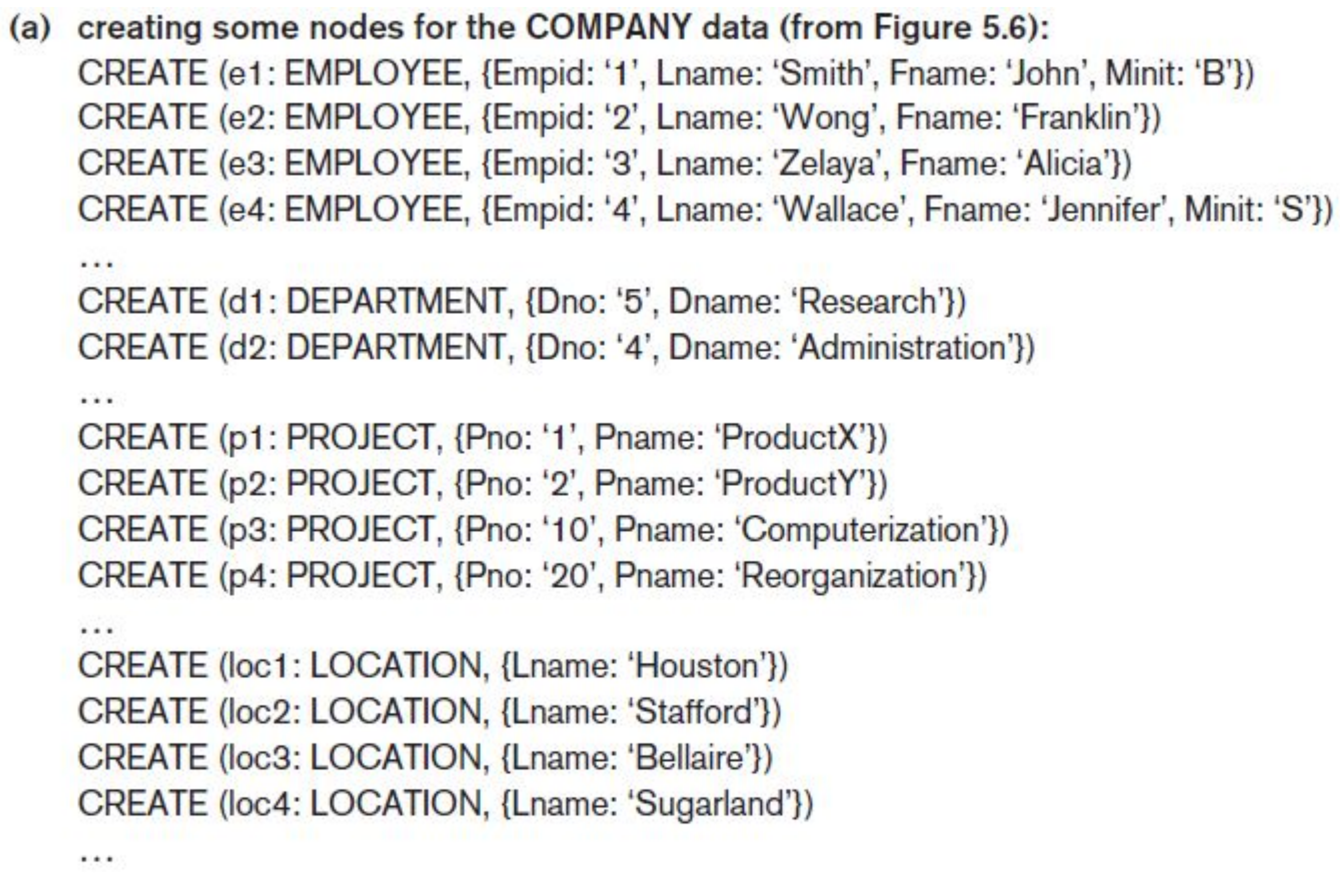
Neo4j (cont’d.)

- Nodes can have labels
  - Zero, one, or several
- Both nodes and relationships can have properties
- Each relationship has a start node, end node, and a relationship type
- Properties specified using a map pattern
- Somewhat similar to ER/EER concepts

Neo4j (cont'd.)

- Creating nodes in Neo4j
  - CREATE command
  - Part of high-level declarative query language Cypher
  - Node label can be specified when node is created
  - Properties are enclosed in curly brackets

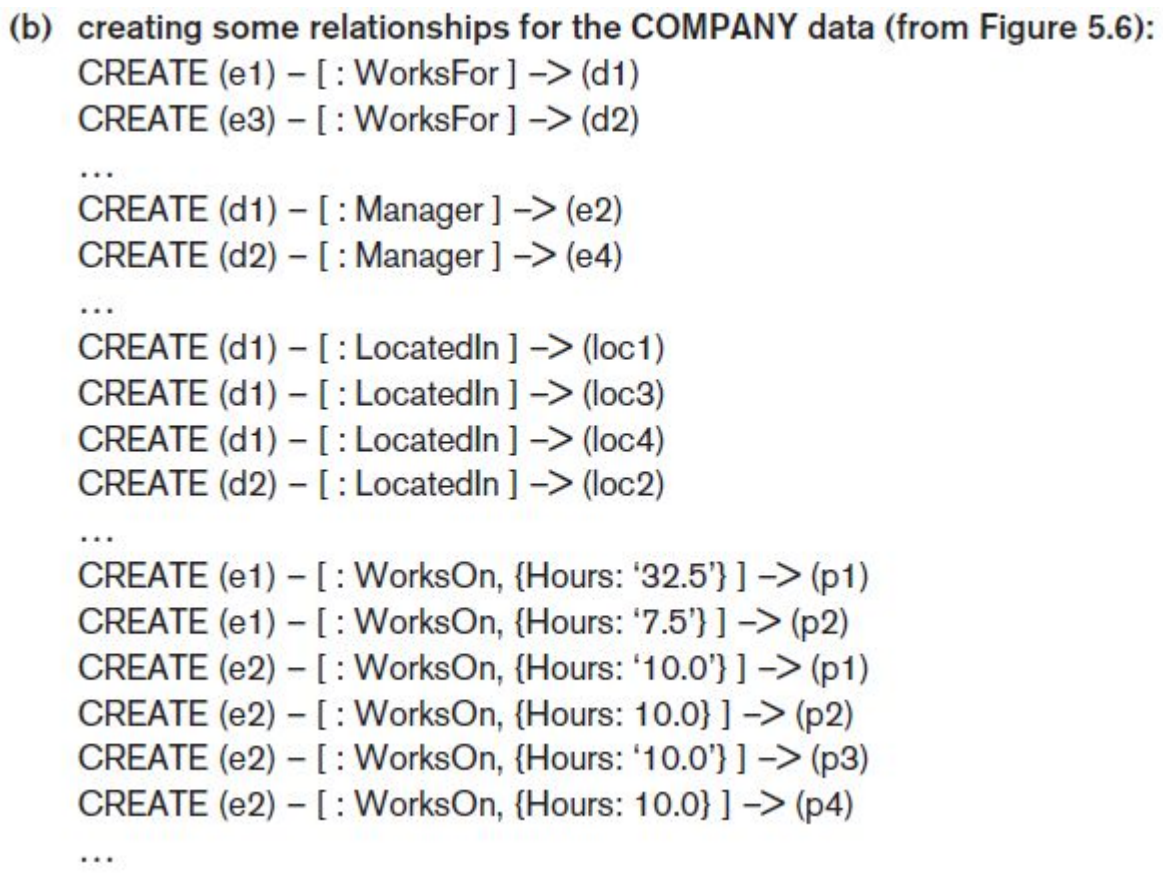
Neo4j (cont'd.)



```
(a)  creating some nodes for the COMPANY data (from Figure 5.6):
     CREATE (e1: EMPLOYEE, {Empid: '1', Lname: 'Smith', Fname: 'John', Minit: 'B'})
     CREATE (e2: EMPLOYEE, {Empid: '2', Lname: 'Wong', Fname: 'Franklin'})
     CREATE (e3: EMPLOYEE, {Empid: '3', Lname: 'Zelaya', Fname: 'Alicia'})
     CREATE (e4: EMPLOYEE, {Empid: '4', Lname: 'Wallace', Fname: 'Jennifer', Minit: 'S'})
     ...
     CREATE (d1: DEPARTMENT, {Dno: '5', Dname: 'Research'})
     CREATE (d2: DEPARTMENT, {Dno: '4', Dname: 'Administration'})
     ...
     CREATE (p1: PROJECT, {Pno: '1', Pname: 'ProductX'})
     CREATE (p2: PROJECT, {Pno: '2', Pname: 'ProductY'})
     CREATE (p3: PROJECT, {Pno: '10', Pname: 'Computerization'})
     CREATE (p4: PROJECT, {Pno: '20', Pname: 'Reorganization'})
     ...
     CREATE (loc1: LOCATION, {Lname: 'Houston'})
     CREATE (loc2: LOCATION, {Lname: 'Stafford'})
     CREATE (loc3: LOCATION, {Lname: 'Bellaire'})
     CREATE (loc4: LOCATION, {Lname: 'Sugarland'})
     ...
```

Figure 24.4 Examples in Neo4j using the Cypher language (a) Creating some nodes

Neo4j (cont'd.)



(b) creating some relationships for the COMPANY data (from Figure 5.6):

```
CREATE (e1) – [ : WorksFor ] –> (d1)
CREATE (e3) – [ : WorksFor ] –> (d2)
…
CREATE (d1) – [ : Manager ] –> (e2)
CREATE (d2) – [ : Manager ] –> (e4)
…
CREATE (d1) – [ : LocatedIn ] –> (loc1)
CREATE (d1) – [ : LocatedIn ] –> (loc3)
CREATE (d1) – [ : LocatedIn ] –> (loc4)
CREATE (d2) – [ : LocatedIn ] –> (loc2)
…
CREATE (e1) – [ : WorksOn, {Hours: '32.5'} ] –> (p1)
CREATE (e1) – [ : WorksOn, {Hours: '7.5'} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p1)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p3)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p4)
…
```

Figure 24.4 (cont'd.) Examples in Neo4j using the Cypher language
(b) Creating some relationships

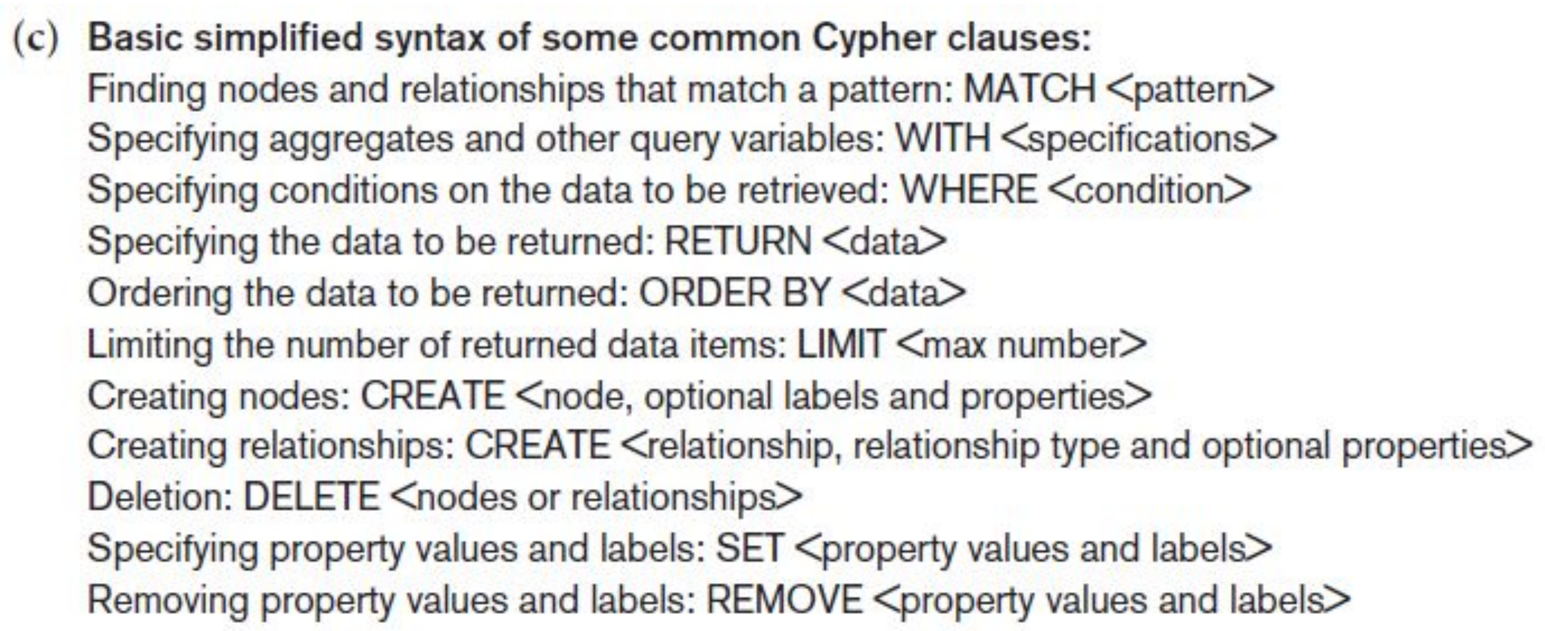
Neo4j (cont’d.)

- Path
  - Traversal of part of the graph
  - Typically used as part of a query to specify a pattern
- Schema optional in Neo4j
- Indexing and node identifiers
  - Users can create for the collection of nodes that have a particular label
  - One or more properties can be indexed

The Cypher Query Language of Neo4j

- Cypher query made up of clauses
- Result from one clause can be the input to the next clause in the query

The Cypher Query Language of Neo4j (cont’d.)

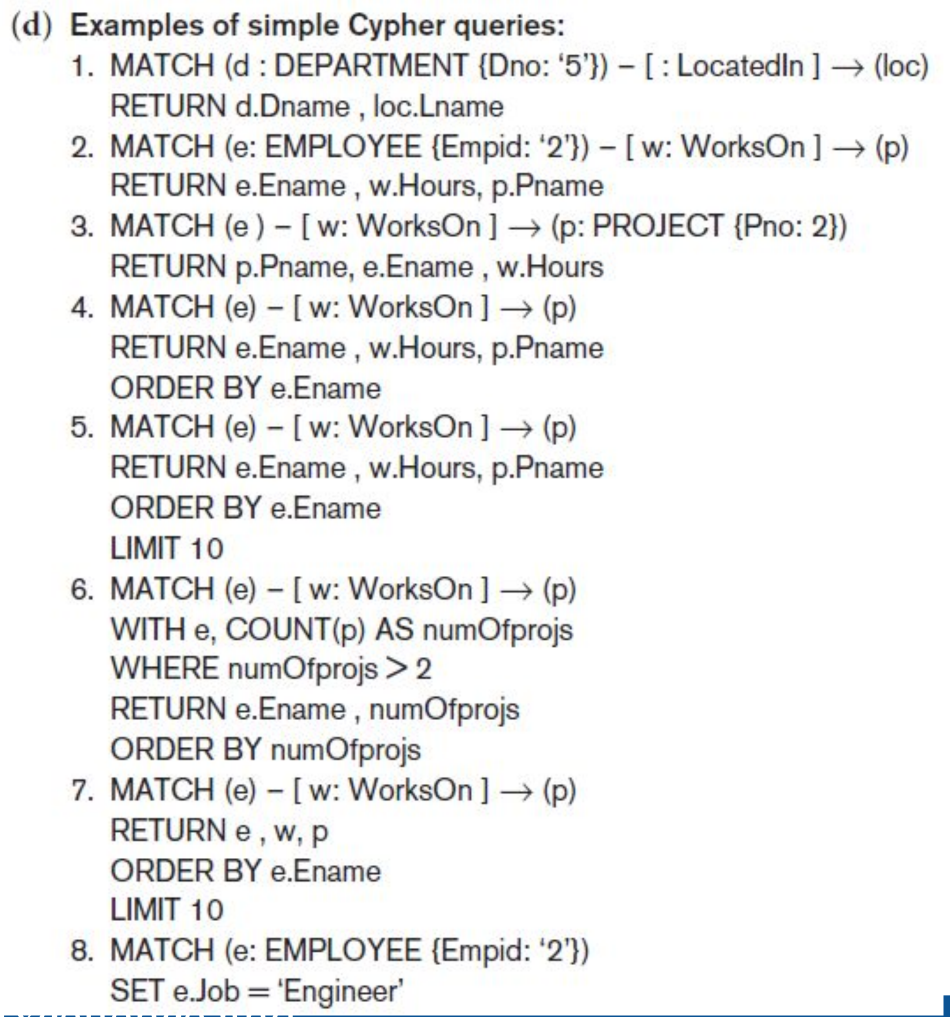


(c) **Basic simplified syntax of some common Cypher clauses:**
Finding nodes and relationships that match a pattern: MATCH <pattern>
Specifying aggregates and other query variables: WITH <specifications>
Specifying conditions on the data to be retrieved: WHERE <condition>
Specifying the data to be returned: RETURN <data>
Ordering the data to be returned: ORDER BY <data>
Limiting the number of returned data items: LIMIT <max number>
Creating nodes: CREATE <node, optional labels and properties>
Creating relationships: CREATE <relationship, relationship type and optional properties>
Deletion: DELETE <nodes or relationships>
Specifying property values and labels: SET <property values and labels>
Removing property values and labels: REMOVE <property values and labels>

Figure 24.4 (cont’d.) Examples in Neo4j using the Cypher language
(c) Basic syntax of Cypher queries

The Cypher Query Language of Neo4j (cont'd.)

Figure 24.4 (cont'd.) Examples in Neo4j using the Cypher language (d) Examples of Cypher queries



(d) **Examples of simple Cypher queries:**

```
1. MATCH (d : DEPARTMENT {Dno: '5'}) – [ : LocatedIn ] → (loc)
   RETURN d.Dname , loc.Lname
2. MATCH (e: EMPLOYEE {Empid: '2'}) – [ w: WorksOn ] → (p)
   RETURN e.Ename , w.Hours, p.Pname
3. MATCH (e ) – [ w: WorksOn ] → (p: PROJECT {Pno: 2})
   RETURN p.Pname, e.Ename , w.Hours
4. MATCH (e) – [ w: WorksOn ] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
5. MATCH (e) – [ w: WorksOn ] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
   LIMIT 10
6. MATCH (e) – [ w: WorksOn ] → (p)
   WITH e, COUNT(p) AS numOfprojs
   WHERE numOfprojs > 2
   RETURN e.Ename , numOfprojs
   ORDER BY numOfprojs
7. MATCH (e) – [ w: WorksOn ] → (p)
   RETURN e , w, p
   ORDER BY e.Ename
   LIMIT 10
8. MATCH (e: EMPLOYEE {Empid: '2'})
   SET e.Job = 'Engineer'
```

Neo4j Interfaces and Distributed System Characteristics

- Enterprise edition versus community edition
  - Enterprise edition supports caching, clustering of data, and locking
- Graph visualization interface
  - Subset of nodes and edges in a database graph can be displayed as a graph
  - Used to visualize query results
- Master-slave replication
- Caching
- Logical logs

References

1. tusharkute.com
2. http://mitu.co.in http://tusharkute.com
3. http://digitallocha.blogspot.in http://kyamputar.blogspot.in
4. Ramez Elmasri and Shamkant B. Navathe

Thank you