

PUNE INSTITUTE OF COMPUTER TECHNOLOGY
DHANKAWADI, PUNE – 43.
LIST OF LAB EXPERIMENTS
ACADEMIC YEAR: 2016-17

Programme : Information Technology
Class : B.E. IT
Course : Software Laboratory VI
Laboratory No: Lab VI

Date: 14/12/2016
SEMESTER: II

LAB NO.	PROBLEM STATEMENT	Page No
1.	Study and Configure Hadoop for Big Data	1
2.	Study of NoSQL Databases such as Hive/Hbase/Cassandra/DynamoDB	5
3.	Design Data Model using NoSQL Databases such as Hive/ Hbase/ Cassandra/ DynamoDB	9
4.	Implement any one Partitioning technique in Parallel Databases	12
5.	Implement Two Phase commit protocol in Distributed Databases	22
6.	Design Persistent Objects using JDO and implement min 10 queries on objects using JDOQL in ObjectDB NOSQL DATABASE	34
7.	Create XML, XML schemas , DTD for any database application and implement min 10 queries using XQuery FLOWR expression and XPath	41
8.	Design database schemas and implement min 10 queries using Hive/ Hbase/ Cassandra column based databases	49
9.	Design database schemas and implement min 10 queries using DynamoDBkeyValue based databases	62
10.	Implement Web Page ranking algorithm	94
11.	Implement any one machine learning algorithm for classification / clustering task in BIG data Analytics	102
12.	Design and Implement social web mining application using NoSQL databases, machine learning algorithm, Hadoop and Java/.Net	117

Course Coordinator
(Prof. R. B. Murumkar)

HOD IT
(Dr. S. C. Dharmadhikari)

SEMESTER – II

Subject Code	Subject	Teaching Scheme			Examination Scheme					Total Marks
		Lecture	Practical	Tutorial	In-Semester Assessment	TW	PR	OR	End Semester Examination	
					Phase - I				Phase - II	
414461	Distributed System	3			30				70	100
414462	Advanced Databases	3			30				70	100
414463	Elective – III	3	2	--	30	25	--	25	70	150
414464	Elective – IV	3			30				70	100
414465	Software Laboratory - V	--	2	--		25	25	--		50
414466	Software Laboratory - VI	--	4	--		--	50	50		100
414467	Project Work	--	--	6		50	--	100		150
Total		12	8	6	120	100	75	175	280	750

Software Laboratory – V: (Distributed Systems)

Software Laboratory – VI: (Advanced Databases)

Elective – III	Elective – IV
414463 A :Mobile Computing	414464 A :Bio Informatics
414463 B :Advanced Graphics and Animation	414464 B :Real Time and Embedded Systems
414463 C :Information Storage and Retrieval	414464 C :Green IT - Principles and Practices
414463 D :IT Enabled Services	414464 D :Internet of Things
414463 E :Advanced Computer Networks	414464 E :Open Elective

414466 : SOFTWARE LABORATORY – VI

Teaching Scheme:

Practical : 4 Hours/Week

Examination Scheme:

Practical : 50 Marks

Oral : 50 Marks

Prerequisites : Database Management System

Course Objectives :

1. To learn and understand Database Modeling, Architectures.
2. To learn and understand Advanced Database Programming Frameworks.
3. To learn and understand web database language, XML, JDOQL.
4. To learn NoSQL Databases (Open source) such as Hive/ Hbase/ Cassandra/DynamoDB.

Course Outcomes :

1. Understanding of Advanced Database Programming Languages.
2. Master the basics of web and object oriented database languages and construct queries using XML and JDOQL.
3. Master the basic concepts of NoSQL Databases.
4. Understand how analytics and big data affect various functions now and in the future.
5. Appreciate the impact of analytics and big data on the information industry and the external ecosystem for analytical and data services.

Contents

1. Study and Configure Hadoop for Big Data
2. Study of NoSQL Databases such as Hive/Hbase/Cassandra/DynamoDB
3. Design Data Model using NoSQL Databases such as Hive/Hbase/Cassandra/DynamoDB
4. Implement any one Partitioning technique in Parallel Databases
5. Implement Two Phase commit protocol in Distributed Databases
6. Design Persistent Objects using JDO and implement min 10 queries on objects using JDOQL in ObjectDB NOSQL DATABASE
7. Create XML, XML schemas , DTD for any database application and implement min 10 queries using XQuery FLOWR expression and XPath
8. Design database schemas and implement min 10 queries using Hive/ Hbase/ Cassandra column based databases
9. Design database schemas and implement min 10 queries using DynamoDBkeyValue based databases
10. Implement Web Page ranking algorithm
11. Implement any one machine learning algorithm for classification / clustering task in BIG data Analytics
12. Design and Implement social web mining application using NoSQL databases, machine learning algorithm, Hadoop and Java/.Net

Instructor should maintain progress report of mini project throughout the semester from project group and assign marks as a part of the term work

Instructor should frame Practical Assignments based on above mentioned list of assignments. Submission of each Practical Assignment should be in the form of handwritten write-ups/ printout of

source code and output. Instructor should assign an assignment no. 12 to a group of 3 - 4 students. Practical Examination will be based on all topics covered and questions will be asked to judge understanding of practical performed at the time of practical examination.

Group of students should submit the Report for assignment no. 12 which will consist of Title of the Project, Abstract, Introduction, scope, Requirements, Data Modeling, Database design, Algorithms, Graphical User Interface, Source Code, Testing document, Conclusion.

All the assignments should be conducted on the latest version of Open Source Operating Systems, tools and Multi-core CPU supporting Virtualization and Multi-Threading.

Reference Books

1. <http://nosql-database.org/>
2. Hadoop, O'Reilly Publications.
3. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6th Edition, McGraw Hill Publishers, ISBN 0-07-120413-X.
4. <http://www.objectdb.com/database/jdo>
5. Data Mining: Concepts and Techniques by Jiawei Han, Micheline Kamber, Jian Pei, Elsevier.

LAB 1: Study and Configure Hadoop for Big Data

Objective:

1. To learn and understand Database Modeling, Architectures.
2. To learn and understand Advanced Database Programming Frameworks.

Introduction

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. This approach takes advantage of data locality— nodes manipulating the data they have access to— to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking. The base Apache Hadoop framework is composed of the following modules:

- ❑ Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
- ❑ Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- ❑ Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications; and
- ❑ Hadoop MapReduce – an implementation of the MapReduce programming model for large scale data processing.

Steps:

```
sudo apt-get update
```

```
sudo apt-get install openjdk-7-jre-headless
```

```
sudo apt-get install openjdk-7-jdk
```

```
sudo apt-get install ssh
```

```
sudo apt-get install rsync
```

```
# Download hadoop from : http://www.eu.apache.org/dist/hadoop/common/stable/hadoop-2.7.1.tar.gz
```

```
# copy and extract hadoop-2.7.1.tar.gz in home folder
```

```
# rename the name of the extracted folder from hadoop-2.7.1 to hadoop
```

```
readlink -f /usr/bin/javac
# find whether ubuntu is 32 bit (i686) or 64 bit (x86_64)
uname -i
gedit ~/hadoop/etc/hadoop/hadoop-env.sh
# add following line in it
# for 32 bit ubuntu
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
# for 64 bit ubuntu
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
# save and exit the file
# to display the usage documentation for the hadoop script try next command
~/hadoop/bin/hadoop
```

1. For standalone mode

```
mkdir input
cp ~/hadoop/etc/hadoop/*.xml input
~/hadoop/bin/hadoop jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar grep input output 'us[a-z.]+'
cat output/*
# Our task is done, so remove input and output folders
rm -r input output
```

2. Pseudo-Distributed mode

```
# get your user name
whoami
# remember your user name, we'll use it in the next step
gedit ~/hadoop/etc/hadoop/core-site.xml
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:1234</value>
</property>
```

```

</configuration>
gedit ~/hadoop/etc/hadoop/hdfs-site.xml
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.name.dir</name>
<value>file:///home/your_user_name/hadoop/name_dir</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:///home/your_user_name/hadoop/data_dir</value>
</property>
</configuration>

```

#Setup passphraseless/passwordless ssh

```

ssh-keygen -t dsa -P "" -f ~/.ssh/id_dsa
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
export HADOOP_PREFIX=/home/your_user_name/hadoop
ssh localhost
# type exit in the terminal to close the ssh connection (very important)
exit

```

The following instructions are to run a MapReduce job locally.

```

#Format the filesystem:( Do it only once )
~/hadoop/bin/hdfs namenode -format
#Start NameNode daemon and DataNode daemon:
~/hadoop/sbin/start-dfs.sh
#Browse the web interface for the NameNode; by default it is available at:

```

`http://localhost:50070/`

#Make the HDFS directories required to execute MapReduce jobs:

`~/hadoop/bin/hdfs dfs -mkdir /user`

`~/hadoop/bin/hdfs dfs -mkdir /user/your_user_name`

#Copy the sample files (from `~/hadoop/etc/hadoop`) into the distributed filesystem folder(input)

`~/hadoop/bin/hdfs dfs -put ~/hadoop/etc/hadoop input`

#Run the example map-reduce job

`~/hadoop/bin/hadoop jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar grep input output 'us[a-z.]+'`

#View the output files on the distributed filesystem

`~/hadoop/bin/hdfs dfs -cat output/*`

#Copy the output files from the distributed filesystem to the local filesystem and examine them:

`~/hadoop/bin/hdfs dfs -get output output`

#ignore warnings (if any)

`cat output/*`

remove local output folder

`rm -r output`

remove distributed folders (input & output)

`~/hadoop/bin/hdfs dfs -rm -r input output`

#When you're done, stop the daemons with

`~/hadoop/sbin/stop-dfs.sh`

Conclusion: In this way the Hadoop was installed & configured on Ubuntu for BigData.

Reference :: <http://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/SingleCluster.html>

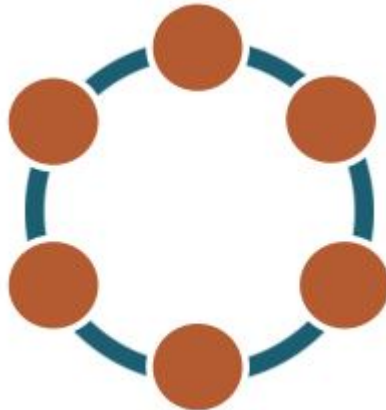
Lab 2: Study of NoSQL Database - *Cassandra*.

Objective:

1. To learn NoSQL Databases (Open source) such as Hive/ Hbase/ Cassandra/DynamoDB.

Introduction

The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages. Apache Cassandra, a top level Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable, is a distributed database for managing large amounts of structured data across many commodity servers, while providing highly available service and no single point of failure. Cassandra offers capabilities that relational databases and other NoSQL databases simply cannot match such as: continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud



availability zones.

Cassandra's architecture is responsible for its ability to scale, perform, and offer continuous uptime. Rather than using a legacy master-slave or a manual and difficult-to-maintain sharded architecture, Cassandra has a masterless "ring" design that is elegant, easy to setup, and easy to maintain.

In Cassandra, all nodes play an identical role; there is no concept of a master node, with all nodes communicating with each other equally. Cassandra's built-for-scale architecture means that it is capable of handling large amounts of data and thousands of concurrent users or operations per second—even across multiple data centers—as easily as it can manage much smaller amounts of data and user traffic. Cassandra's architecture also means that, unlike other master-slave or sharded systems, it has no single point of failure and therefore is capable of

offering true continuous availability and uptime — simply add new nodes to an existing cluster without having to take it down.

Many companies have successfully deployed and benefited from Apache Cassandra including some large companies such as: Apple, Comcast, Instagram, Spotify, eBay, Rackspace, Netflix, and many more. The larger production environments have PB's of data in clusters of over 75,000 nodes. Cassandra is available under the Apache 2.0 license.

Cassandra installation

The following instructions are applicable to Ubuntu 14.04, Ubuntu 15.04 & Ubuntu 15.10.

Steps:

Download Cassandra from following url (the most stable release)

<http://www.eu.apache.org/dist/cassandra/2.2.4/apache-cassandra-2.2.4-bin.tar.gz>

Copy and extract apache-cassandra-2.2.4-bin.tar.gz in home folder

Rename the extracted folder name from apache-cassandra-2.2.4 to cassandra

```
sudo apt-get update
```

```
sudo apt-get install openjdk-7-jre-headless
```

```
sudo apt-get install openjdk-7-jdk
```

```
java -version
```

```
mkdir ~/cassandra/lib/data
```

```
mkdir ~/cassandra/lib/commitlog
```

```
mkdir ~/cassandra/lib/saved_caches
```

```
mkdir ~/cassandra/log
```

remember your user name

```
whoami
```

```
gedit ~/cassandra/conf/cassandra.yaml
```

Go to the end of file and add following lines

```
data_file_directories:
```

```
- /home/your_user_name/cassandra/lib/data
```

```
commitlog_directory: /home/your_user_name/cassandra/lib/commitlog
```

```
saved_caches_directory: /home/your_user_name/cassandra/lib/saved_caches
```

save and exit the file

```
gedit ~/cassandra/conf/logback.xml
```

```
# replace line 35 by
<file>/home/your_user_name/cassandra/log/system.log</file>
# save and exit the file
~/cassandra/bin/cassandra -f
# The cassandra daemon should start in the foreground (don't press ctrl + c; as it'll terminate
the daemon)
# open a new terminal
~/cassandra/bin/cqlsh
# It'll have output like this
cqlsh>
# First, create a keyspace
CREATE KEYSPACE mykeyspace
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
# use the new keyspace:
USE mykeyspace;
# create a student table:
CREATE TABLE student (
roll int PRIMARY KEY,
fname text,
lname text
);
# store data into student table:
INSERT INTO student (roll, fname, lname)
VALUES (1, 'jack', 'castle');
INSERT INTO student (roll, fname, lname)
VALUES (2, 'james', 'doe');
INSERT INTO student (roll, fname, lname)
VALUES (3, 'john', 'smith');
# fetch the data from table:
SELECT * FROM student;
# output :
```

```
roll | fname | lname
```

```
-----+-----+-----
```

```
1 | jack | castle
```

```
2 | james | doe
```

```
3 | john | smith
```

```
# retrieve data about students whose last name is smith by creating an index
```

```
CREATE INDEX ON student (lname);
```

```
SELECT * FROM student WHERE lname = 'smith';
```

```
roll | fname | lname
```

```
-----+-----+-----
```

```
3 | john | smith
```

```
cqlsh> exit
```

```
# for more CQL queries please refer
```

```
http://docs.datastax.com/en/latest-cql/cql/cql\_using/useAboutCQL.html
```

```
# stop cassandra
```

```
Just press ctrl + c in the terminal where the cassandra daemon is running.
```

Reference :

```
http://wiki.apache.org/cassandra/GettingStarted
```

```
http://www.planetcassandra.org/what-is-apache-cassandra/
```

Lab 3: Design Data Model using NoSQL Database – *Cassandra*

Objective:

1. To learn and understand Database Modeling, Architectures
2. To learn NoSQL Databases (Open source) such as Hive/ Hbase/ Cassandra/DynamoDB.

Steps:

First install and configure **Cassandra** from,

<https://sl6it.wordpress.com/2015/12/10/4-installation-of-nosql-database-cassandra/>

We will create data model for a social music service.

start cassandra daemon

~/cassandra/bin/cassandra -f

The cassandra daemon should start in the foreground

(don't press ctrl + c; as it'll terminate the daemon)

open a **new** terminal

~/cassandra/bin/cqlsh

It'll have output like this

cqlsh>

First, create a keyspace

CREATE KEYSPACE mykeyspace100

WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

use the new keyspace:

USE mykeyspace100;

Create a songs table having a title, album, and artist column, plus a column (called data) for the

actual audio file itself.

CREATE TABLE songs (

id **int PRIMARY KEY**,

title **text**,

album **text**,

artist **text**,

data blob

);

Create playlists table

CREATE TABLE playlists (

id **int**,

song_order **int**,

song_id **int**,

title **text**,

album **text**,

artist **text**,

PRIMARY KEY (id, song_order));

The combination of the **id** and **song_order** (**Compound Primary Key**) in the playlists table uniquely identifies a row in the playlists table.

```

# Use the INSERT command to insert data in the playlists table.
INSERT INTO playlists (id, song_order, song_id, title, artist, album)
VALUES (100, 1, 1001, 'La Grange', 'Mike', 'Tres Hombres');
INSERT INTO playlists (id, song_order, song_id, title, artist, album)
VALUES (100, 2, 1002, 'Moving in', 'Swift', 'We must Obey');
INSERT INTO playlists (id, song_order, song_id, title, artist, album)
VALUES (100, 3, 1003, 'One day', 'Justin', 'Roll Away');
INSERT INTO playlists (id, song_order, song_id, title, artist, album)
VALUES (100, 4, 1004, 'Ojo Rojo', 'Swift', 'No One');
# Now use the SELECT query to display the table's data
SELECT * FROM playlists;
# Create index on artist
CREATE INDEX ON playlists( artist );
# Search albums and titles of artist 'Swift'
SELECT album, title FROM playlists WHERE artist = 'Swift';
# You can query a single sequential set of data on disk to get the songs for a playlist.
SELECT * FROM playlists WHERE id = 100
ORDER BY song_order DESC LIMIT 50;
# Adding a collection to a table
Collection in Cassandra is of three types
- Set, List, Map
Each element of a set, list, or map is internally stored as one Cassandra column.
# Alter playlist table to add a collection set, tags:
ALTER TABLE playlists ADD tags set<text>;
# Updating a collection
To update a set, use the UPDATE command and the addition (+) operator to add an element or
the
subtraction (-) operator to remove an element.
# Update the playlists table to insert the tags data:
UPDATE playlists SET tags = tags + {'2010'} WHERE id = 100 AND song_order = 4;
SELECT * FROM playlists;
UPDATE playlists SET tags = tags + {'2007'} WHERE id = 100 AND song_order = 2;
SELECT * FROM playlists;
UPDATE playlists SET tags = tags + {'classic'} WHERE id = 100 AND song_order = 2;
SELECT * FROM playlists;
UPDATE playlists SET tags = tags + {'1973'} WHERE id = 100 AND song_order = 1;
SELECT * FROM playlists;
UPDATE playlists SET tags = tags + {'blues'} WHERE id = 100 AND song_order = 1;
SELECT * FROM playlists;
UPDATE playlists SET tags = tags + {'rock'} WHERE id = 100 AND song_order = 4;
SELECT * FROM playlists;
# Alter playlist table to add a list collection, reviews:
ALTER TABLE playlists ADD reviews list<text>;
To update a list, a similar syntax using square brackets instead of curly brackets is used.

```

```

UPDATE playlists SET reviews =reviews + [ 'best lyrics' ] WHERE id=100 and song_order= 4;
SELECT * FROM playlists;
UPDATE playlists SET reviews =reviews + [ 'magical' ] WHERE id=100 and song_order= 4;
SELECT * FROM playlists;
# Alter playlist table to add a map collection, venue (a schedule of live appearances).
ALTER TABLE playlists ADD venue map<timestamp, text>;
# To update a map, use INSERT to specify the data in a map collection.
INSERT INTO playlists (id, song_order, venue)
VALUES (100, 4, { '2016-9-22 22:00' : 'The Fillmore', '2016-10-1 21:00' : 'The Apple Barrel'});
SELECT * FROM playlists;
INSERT INTO playlists (id, song_order, venue)
VALUES (100, 3, { '2016-1-12 22:00' : 'Cactus Cafe', '2016-01-22 20:00' : 'Mohawk'});
SELECT * FROM playlists;
Inserting data into the map replaces the entire map.
# Indexing a collection
We can index collections and query the database to find a collection containing a particular
value.
Suppose we want to find songs tagged blues and that debuted at the Fillmore.
# First index the tags set and venue map.
CREATE INDEX ON playlists (tags);
CREATE INDEX mymapindex ON playlists (venue);
# Filter data in a collection
SELECT album, tags FROM playlists;
# Query for values in the tags set.
SELECT album, tags FROM playlists WHERE tags CONTAINS 'blues';
# Query for values in the venue map.
SELECT artist, venue FROM playlists WHERE venue CONTAINS 'The Fillmore';
# Exit cqlsh
cqlsh> exit

```

Conclusion: Data model was designed using NoSQL database – Cassandra, a distributed database for managing large amounts of structured data. Cassandra offers capabilities that relational databases and other NoSQL databases simply cannot match such as: continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones

Reference:

http://docs.datastax.com/en/cql/3.1/cql/ddl/ddl_intro_c.html

--

LAB 4: Implementing Partitioning Technique in Parallel Database.

Objective:

1. To learn and understand Database Modeling, Architectures.

Introduction:

Overview of Parallel Execution

Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with decision support systems (DSS) and data warehouses. You can also implement **parallel execution** on certain types of online transaction processing (OLTP) and hybrid systems. Parallel execution is sometimes called **parallelism**. Simply expressed, parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time. An example of this is when four processes handle four different quarters in a year instead of one process handling all four quarters by itself. The improvement in performance can be quite high. In this case, each quarter will be a **partition**, a smaller and more manageable unit of an index or table.

When to Implement Parallel Execution

The most common use of parallel execution is in DSS and data warehousing environments. Complex queries, such as those involving joins of several tables or searches of very large tables, are often best executed in parallel.

Parallel execution is useful for many types of operations that access significant amounts of data. Parallel execution improves processing for:

1. Large table scans and joins
2. Creation of large indexes
3. Partitioned index scans
4. Bulk inserts, updates, and deletes
5. Aggregations and copying

You can also use parallel execution to access object types within an Oracle database. For example, use parallel execution to access LOBs (large objects).

Parallel execution benefits systems that have *all* of the following characteristics:

Symmetric multi-processors (SMP), clusters, or massively parallel systems

Sufficient I/O bandwidth

Underutilized or intermittently used CPUs (for example, systems where CPU usage is typically less than 30%)

Sufficient memory to support additional memory-intensive processes such as sorts, hashing, and I/O buffers

If your system lacks any of these characteristics, parallel execution might not significantly improve performance. In fact, parallel execution can reduce system performance on overutilized systems or systems with small I/O bandwidth.

Granules of Parallelism

Different parallel operations use different types of parallelism. The optimal physical database layout depends on the parallel operations that are most prevalent in your application or even of the necessity of using partitions.

The basic unit of work in parallelism is called a granule. Oracle Database divides the operation being parallelized (for example, a table scan, table update, or index creation) into granules. Parallel execution processes execute the operation one granule at a time. The number of granules and their size correlates with the degree of parallelism (DOP). It also affects how well the work is balanced across query server processes. There is no way you can enforce a specific granule strategy as Oracle Database makes this decision internally.

Block Range Granules

Block range granules are the basic unit of most parallel operations, even on partitioned tables. Therefore, from an Oracle Database perspective, the degree of parallelism is not related to the number of partitions.

Block range granules are ranges of physical blocks from a table. The number and the size of the granules are computed during runtime by Oracle Database to optimize and balance the work distribution for all affected parallel execution servers. The number and size of granules are dependent upon the size of the object and the DOP. Block range granules do not depend on static preallocation of tables or indexes. During the computation of the granules, Oracle Database takes the DOP into account and tries to assign granules from different datafiles to each of the parallel execution servers to avoid contention whenever possible. Additionally,

Oracle Database considers the disk affinity of the granules on MPP systems to take advantage of the physical proximity between parallel execution servers and disks.

When block range granules are used predominantly for parallel access to a table or index, administrative considerations (such as recovery or using partitions for deleting portions of data) might influence partition layout more than performance considerations.

Partition Granules

When partition granules are used, a query server process works on an entire partition or subpartition of a table or index. Because partition granules are statically determined by the structure of the table or index when a table or index is created, partition granules do not give you the flexibility in parallelizing an operation that block granules do. The maximum allowable DOP is the number of partitions. This might limit the utilization of the system and the load balancing across parallel execution servers.

When partition granules are used for parallel access to a table or index, you should use a relatively large number of partitions (ideally, three times the DOP), so that Oracle can effectively balance work across the query server processes.

Partition granules are the basic unit of parallel index range scans and of parallel operations that modify multiple partitions of a partitioned table or index. These operations include parallel creation of partitioned indexes, and parallel creation of partitioned tables.

Partitioning Design Considerations

In conjunction with parallel execution, partitioning can improve performance in data warehouses. The following are the main design considerations for partitioning:

- 1.Types of Partitioning
- 2.Partition Pruning
- 3.Partition-Wise Joins

Types of Partitioning

This section describes the partitioning features that significantly enhance data access and improve overall application performance. This is especially true for applications that access tables and indexes with millions of rows and many gigabytes of data.

Partitioned tables and indexes facilitate administrative operations by enabling these operations to work on subsets of data. For example, you can add a new partition, organize an

existing partition, or drop a partition and cause less than a second of interruption to a read-only application.

Using the partitioning methods described in this section can help you tune SQL statements to avoid unnecessary index and table scans (using partition pruning). You can also improve the performance of massive join operations when large amounts of data (for example, several million rows) are joined together by using partition-wise joins. Finally, partitioning data greatly improves manageability of very large databases and dramatically reduces the time required for administrative tasks such as backup and restore.

Granularity can be easily added or removed to the partitioning scheme by splitting partitions. Thus, if a table's data is skewed to fill some partitions more than others, the ones that contain more data can be split to achieve a more even distribution. Partitioning also allows one to swap partitions with a table. By being able to easily add, remove, or swap a large amount of data quickly, swapping can be used to keep a large amount of data that is being loaded inaccessible until loading is completed, or can be used as a way to stage data between different phases of use. Some examples are current day's transactions or online archives.

Partitioning Methods

Oracle offers four partitioning methods:

[Range Partitioning](#)

[Hash Partitioning](#)

[List Partitioning](#)

[Composite Partitioning](#)

Each partitioning method has different advantages and design considerations. Thus, each method is more appropriate for a particular situation.

Range Partitioning

Range partitioning maps data to partitions based on ranges of partition key values that you establish for each partition. It is the most common type of partitioning and is often used with dates. For example, you might want to partition sales data into monthly partitions.

Range partitioning maps rows to partitions based on ranges of column values. Range partitioning is defined by the partitioning specification for a table or index

in `PARTITION BY RANGE(column_list)` and by the partitioning specifications for each individual partition in `VALUES LESS THAN(value_list)`, where `column_list` is an ordered list of columns that determines the partition to which a row or an index entry belongs. These columns are called the partitioning columns. The values in the partitioning columns of a particular row constitute that row's partitioning key.

Hash Partitioning

Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to a partitioning key that you identify. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size. Hash partitioning is the ideal method for distributing data evenly across devices. Hash partitioning is a good and easy-to-use alternative to range partitioning when data is not historical and there is no obvious column or column list where logical range partition pruning can be advantageous.

List Partitioning

List partitioning enables you to explicitly control how rows map to partitions. You do this by specifying a list of discrete values for the partitioning column in the description for each partition. This is different from range partitioning, where a range of values is associated with a partition and with hash partitioning, where you have no control of the row-to-partition mapping. The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a natural way. The following example creates a list partitioned table grouping states according to their sales regions

Composite Partitioning

Composite partitioning combines range and hash or list partitioning. Oracle Database first distributes data into partitions according to boundaries established by the partition ranges. Then, for range-hash partitioning, Oracle uses a hashing algorithm to further divide the data into subpartitions within each range partition. For range-list partitioning, Oracle divides the data into subpartitions within each range partition based on the explicit list you chose.

Index Partitioning

You can choose whether or not to inherit the partitioning strategy of the underlying tables. You can create both local and global indexes on a table partitioned by range, hash, or composite methods. Local indexes inherit the partitioning attributes of their related tables. For

example, if you create a local index on a composite table, Oracle automatically partitions the local index using the composite method.

Performance Issues for Range, List, Hash, and Composite Partitioning

This section describes performance issues for:

1. [When to Use Range Partitioning](#)
2. [When to Use Hash Partitioning](#)
3. [When to Use List Partitioning](#)
4. [When to Use Composite Range-Hash Partitioning](#)
5. [When to Use Composite Range-List Partitioning](#)

Input code:

```
import java.sql.*;
import java.util.*;
public class RangePartitioning {
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/sales";

    // Database credentials
    static final String USER = "root";
    static final String PASS = "gaurav63";

    public static void main(String[] args) {
        Connection conn=null;
        Statement stmt=null;
        Scanner sc=new Scanner(System.in);
        try{
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("Connecting to database");
            conn=DriverManager.getConnection(DB_URL,USER,PASS);

            System.out.println("Press 1. To view JAN 2015 sales");
            System.out.println("Press 2. To view FEB 2015 sales");
            int choice=sc.nextInt();
            switch(choice)
            {
                case 1: System.out.println("-----Sales-----");
                    stmt=conn.createStatement();
                    String sql="select * from sales_jan2015";
                    ResultSet rs=stmt.executeQuery(sql);
                    while(rs.next()){
                        int salesman_id=rs.getInt("salesman_id");
                        String salesman_name=rs.getString("salesman_name");
                        float sales_amount=rs.getFloat("sales_amount");
```

```

        String sales_date=rs.getString("sales_date");

        System.out.println("Salesman_id:"+salesman_id);

        System.out.println("Salesman_name:"+salesman_name);
        System.out.println("Sales_amount:"+sales_amount);
        System.out.println("Sales_Date:"+sales_date);
        System.out.println("-----");
    }
    rs.close();
    stmt.close();
    conn.close();
    break;
case 2: System.out.println("-----Sales-----");
    stmt=conn.createStatement();
    sql="select * from sales_feb2015";
    rs=stmt.executeQuery(sql);
    while(rs.next()){
        int salesman_id=rs.getInt("salesman_id");
        String salesman_name=rs.getString("salesman_name");
        float sales_amount=rs.getFloat("sales_amount");
        String sales_date=rs.getString("sales_date");

        System.out.println("Salesman_id:"+salesman_id);
        System.out.println("Salesman_name:"+salesman_name);
        System.out.println("Sales_amount:"+sales_amount);
        System.out.println("Sales_Date:"+sales_date);
        System.out.println("-----");
    }
    rs.close();
    stmt.close();
    conn.close();
    break;
default: System.out.println("Invalid choice");
}

} catch (Exception e)
{
    e.printStackTrace();
}

}
}

```

For MySQL versions 5 and above following commands can be used to create partition in the database, insert values in the same and view the contents of the partitions:

[Modified part by Shivee Gupta]

create a table with partitions

```
CREATE TABLE r1 (  
    a INT,  
    b INT  
)  
PARTITION BY RANGE (a) (  
    PARTITION p0 VALUES LESS THAN (5),  
    PARTITION p1 VALUES LESS THAN (MAXVALUE)  
);
```

insert values which are automatically sorted into partitions

```
mysql> INSERT INTO r1 VALUES (5,10), (5,11), (5,12);
```

see partition information

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS  
-> FROM INFORMATION_SCHEMA.PARTITIONS  
-> WHERE TABLE_NAME = 'r1';
```

Output:

```
MySQL 5.5 Command Line Client

+-----+
| Tables_in_sales |
+-----+
| sales_feb2015   |
| sales_jan2015   |
| sales_range     |
+-----+
3 rows in set (0.00 sec)

mysql> select * from sales_range;

+-----+-----+-----+-----+
| salesman_id | salesman_name | sales_amount | sales_date |
+-----+-----+-----+-----+
| 111         | Arun          | 1200.00      | 2015-01-01 |
| 112         | Vijay         | 1200.00      | 2015-02-01 |
| 113         | rahul         | 2200.20      | 2015-02-10 |
| 113         | saurabh       | 2230.20      | 2015-01-10 |
| 114         | suresh        | 4230.20      | 2015-01-20 |
| 115         | ramesh        | 8230.20      | 2015-02-20 |
| 115         | ram           | 10230.20     | 2015-01-06 |
| 116         | rema          | 10230.20     | 2015-02-06 |
+-----+-----+-----+-----+
8 rows in set (0.08 sec)

mysql>
```

```
MySQL 5.5 Command Line Client

8 rows in set (0.08 sec)

mysql> select * from sales_jan2015;

+-----+-----+-----+-----+
| salesman_id | salesman_name | sales_amount | sales_date |
+-----+-----+-----+-----+
| 111         | Arun          | 1200.00      | 2015-01-01 |
| 113         | saurabh       | 2230.20      | 2015-01-10 |
| 114         | suresh        | 4230.20      | 2015-01-20 |
| 115         | ram           | 10230.20     | 2015-01-06 |
+-----+-----+-----+-----+
4 rows in set (0.08 sec)

mysql> select * from sales_feb2015;

+-----+-----+-----+-----+
| salesman_id | salesman_name | sales_amount | sales_date |
+-----+-----+-----+-----+
| 112         | Vijay         | 1200.00      | 2015-02-01 |
| 113         | rahul         | 2200.20      | 2015-02-10 |
| 115         | ramesh        | 8230.20      | 2015-02-20 |
| 116         | rema          | 10230.20     | 2015-02-06 |
+-----+-----+-----+-----+
4 rows in set (0.06 sec)

mysql>
```



```
<terminated> RangePartitioning [Java Application] C:\Program Files\Java\jre1.8.0
Connecting to database
Press 1. To view JAN 2015 sales
Press 2. To view FEB 2015 sales
1
-----Sales-----
Salesman_id:111
Salesman_name:Arun
Sales_amount:1200.0
Sales_Date:2015-01-01
-----
Salesman_id:113
Salesman_name:saurabh
Sales_amount:2230.2
Sales_Date:2015-01-10
-----
Salesman_id:114
Salesman_name:suresh
Sales_amount:4230.2
Sales_Date:2015-01-20
-----
Salesman_id:115
Salesman_name:ram
Sales_amount:10230.2
Sales_Date:2015-01-06
-----
```

Conclusion:

Parallel execution is sometimes called **parallelism**. Simply expressed, parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time. An example of this is when four processes handle four different quarters in a year instead of one process handling all four quarters by itself. The improvement in performance can be quite high.

References:

- [1] Internet:<https://web.stanford.edu/dept/itss/docs/oracle/10g/server.101/b10736/parpart.htm#i1006407>

LAB 5: TWO PHASE COMMIT PROTOCOL IN DISTRIBUTED DATABASES

Objective:

1. To learn and understand Database Modeling, Architectures.

Introduction

For the distributed transaction to be ACID-compliant, all the resource managers spanning the transaction and the TPM must enforce the ACID properties. Traditionally, this is accomplished using the two-phase commit (2-PC) protocol. The 2-PC protocol ensures that all resource managers either all commit to completing the transaction or they all abort the transaction, thus leaving the state of their resources unchanged from the pre-transactional state. When an application requests a transaction to commit, the TPM issues a PREPARE_TO_COMMIT request to all the resource managers. Each of these resources managers must reply to the TPM indicating whether it is ready to commit or not. Only when all the resource managers are ready for a commit does the TPM issue a commit request to all resource managers. If any resource manager does not respond positively to the PREPARE_TO_COMMIT request, either because the resource is not available or the resource manager is not reachable, the TPM issues a ROLLBACK request to all the applicable resource managers. The distributed transaction is successfully completed and permanent at the point when all resource managers successfully commit their portion of the distributed transaction and acknowledge this to the TPM. The astute reader may note that even after issuing a COMMIT request to each resource manager, there is a remote possibility one might not respond with a successful commit because the resource manager became unavailable sometime between its response to the PREPARE_TO_COMMIT request and the TPM's request to COMMIT. More on this will follow. It is important to note that the 2-PC protocol is synchronous in nature and thus not well suited for long-lived transactions. Although 2-PC provides autonomy of a transaction, the required processing load is rather heavy. The transaction speed is always limited by the resource manager with the slowest response, and the network traffic and latency is double that of a normal transaction because of the intermediate, PREPARE_TO_COMMIT request. Although 2-PC provides some reliability in achieving ACID compliance for distributed transactions, as insinuated in the previous paragraph, this is not a guarantee. We shall see later that 2-PC is not "bullet-proof."

The protocol works in the following manner: one node is designated the coordinator, which is the master site, and the rest of the nodes in the network are designated the cohorts. The protocol assumes that there is stable storage at each node with a write-ahead log, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other. The last assumption is not too restrictive, as network communication can typically be rerouted. The first two assumptions are much stronger; if a node is totally destroyed then data can be lost.

ALGORITHM:

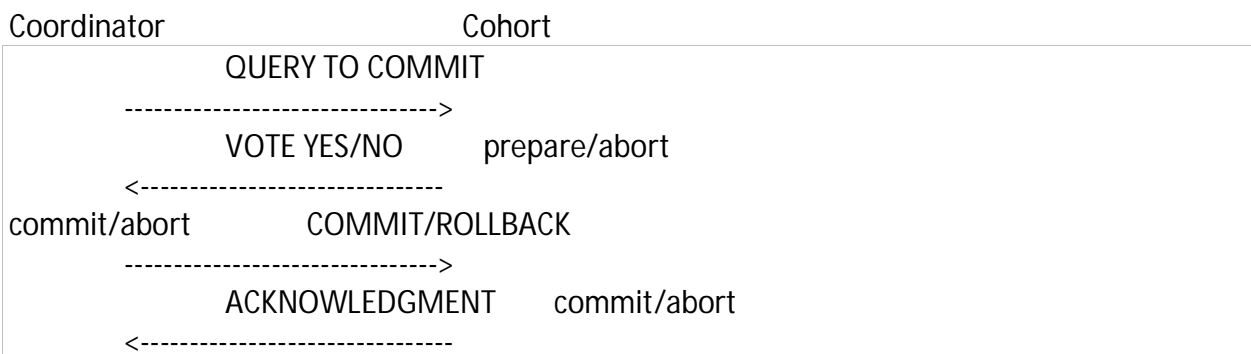
Commit request phase Or voting phase

- 4.The coordinator sends a query to commit message to all cohorts and waits until it has received a reply from all cohorts.
- 5.The cohorts execute the transaction up to the point where they will be asked to commit. They each write an entry to their undo log and an entry to their **redo log**.
- 6.Each cohort replies with an agreement message (cohort votes Yes to commit), if the cohort's actions succeeded, or an abort message (cohort votes No, not to commit), if the cohort experiences a failure that will make it impossible to commit.

Commit phase Or Completion phase

Success

- 7.If the coordinator received an agreement message from all cohorts during the commit-request phase:
- 8.The coordinator sends a commit message to all the cohorts.
- 9.Each cohort completes the operation, and releases all the locks and resources held during the transaction.
- 10.Each cohort sends an acknowledgment to the coordinator.
- 11.The coordinator completes the transaction when all acknowledgments have been received.
- 12.
- 13.Failure
- 14.If any cohort votes No during the commit-request phase (or the coordinator's timeout expires):
- 15.The coordinator sends a rollback message to all the cohorts.
- 16.Each cohort undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
- 17.Each cohort sends an acknowledgement to the coordinator.
- 18.The coordinator undoes the transaction when all acknowledgements have been received.



end

Disadvantages

The greatest disadvantage of the two-phase commit protocol is that it is a blocking protocol. If the coordinator fails permanently, some cohorts will never resolve their transactions: After a cohort has sent an agreement message to the coordinator, it will block until a commit or rollback is received.

Common architecture

In many cases the 2PC protocol is distributed in a computer network. It is easily distributed by implementing multiple dedicated 2PC components similar to each other, typically named Transaction managers (TMs; also referred to as 2PC agents or Transaction Processing Monitors), that carry out the protocol's execution for each transaction (e.g., The Open Group's X/Open XA). The databases involved with a distributed transaction, the participants, both the coordinator and cohorts, register to close TMs (typically residing on respective same network nodes as the participants) for terminating that transaction using 2PC. Each distributed transaction has an ad hoc set of TMs, the TMs to which the transaction participants register. A leader, the coordinator TM, exists for each transaction to coordinate 2PC for it, typically the TM of the coordinator database. However, the coordinator role can be transferred to another TM for performance or reliability reasons. Rather than exchanging 2PC messages among themselves, the participants exchange the messages with their respective TMs. The relevant TMs communicate among themselves to execute the 2PC protocol schema above, "representing" the respective participants, for terminating that transaction. With this architecture the protocol is fully distributed (does not need any central processing component or data structure), and scales up with number of network nodes (network size) effectively.

This common architecture is also effective for the distribution of other atomic commitment protocols besides 2PC, since all such protocols use the same voting mechanism and outcome propagation to protocol participants.

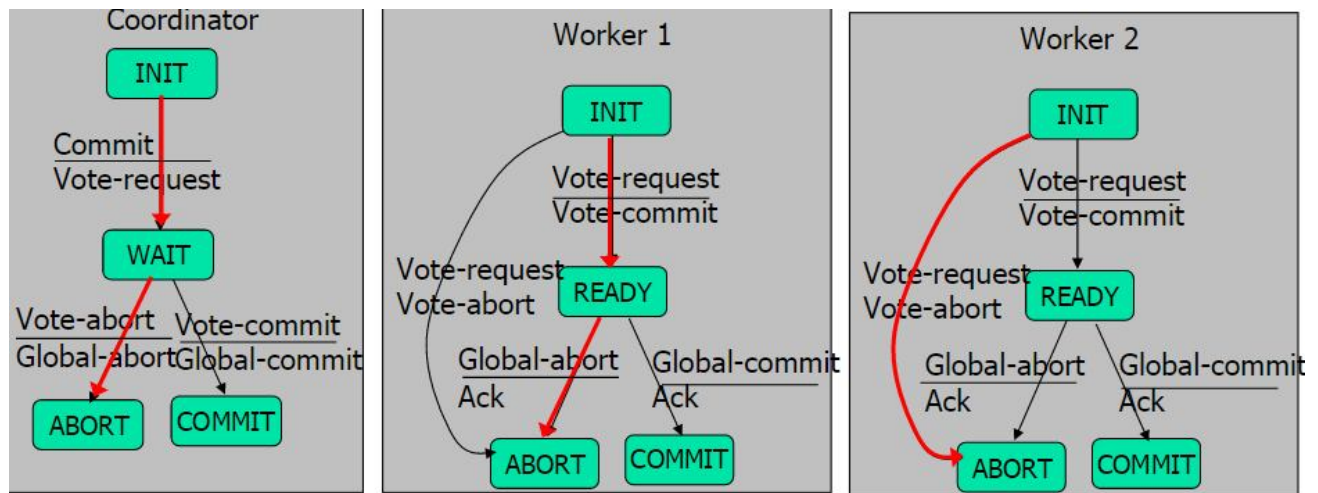
Protocol optimizations

Database research has been done on ways to get most of the benefits of the two-phase commit protocol while reducing costs by protocol optimizations and protocol operations saving under certain system's behavior assumptions.

Presume abort and Presume commit

Presumed abort or Presumed commit are common such optimizations. An assumption about the outcome of transactions, either commit, or abort, can save both messages and logging operations by the participants during the 2PC protocol's execution. For example, when

presumed abort, if during system recovery from failure no logged evidence for commit of some transaction is found by the recovery procedure, then it assumes that the transaction has been aborted, and acts accordingly. This means that it does not matter if aborts are logged at all, and such logging can be saved under this assumption. Typically a penalty of additional operations is paid during recovery from failure, depending on optimization type. Thus the best variant of optimization, if any, is chosen according to failure and transaction outcome statistics.



Tree two-phase commit protocol

The Tree 2PC protocol (also called Nested 2PC, or Recursive 2PC) is a common variant of 2PC in a computer network, which better utilizes the underlying communication infrastructure. The participants in a distributed transaction are typically invoked in an order which defines a tree structure, the invocation tree, where the participants are the nodes and the edges are the invocations (communication links). The same tree is commonly utilized to complete the transaction by a 2PC protocol, but also another communication tree can be utilized for this, in principle. In a tree 2PC the coordinator is considered the root ("top") of a communication tree (inverted tree), while the cohorts are the other nodes. The coordinator can be the node that originated the transaction (invoked recursively (transitively) the other participants), but also another node in the same tree can take the coordinator role instead. 2PC messages from the coordinator are propagated "down" the tree, while messages to the coordinator are "collected" by a cohort from all the cohorts below it, before it sends the appropriate message "up" the tree (except an abort message, which is propagated "up" immediately upon receiving it or if the current cohort initiates the abort).

The Dynamic two-phase commit (Dynamic two-phase commitment, D2PC) protocol is a variant of Tree 2PC with no predetermined coordinator. It subsumes several optimizations that have been proposed earlier. Agreement messages (Yes votes) start to propagate from all the leaves, each leaf when completing its tasks on behalf of the transaction (becoming ready). An intermediate (non leaf) node sends when ready an agreement message to the last (single)

neighboring node from which agreement message has not yet been received. The coordinator is determined dynamically by racing agreement messages over the transaction tree, at the place where they collide. They collide either at a transaction tree node, to be the coordinator, or on a tree edge. In the latter case one of the two edge's nodes is elected as a coordinator (any node). D2PC is time optimal (among all the instances of a specific transaction tree, and any specific Tree 2PC protocol implementation; all instances have the same tree; each instance has a different node as coordinator): By choosing an optimal coordinator D2PC commits both the coordinator and each cohort in minimum possible time, allowing the earliest possible release of locked resources in each transaction participant (tree node).

Input code:

Client code:

```
import java.io.*;
import java.net.*;

public class Client implements Runnable{
    static Socket clientSocket = null;
    static PrintStream os = null;
    static DataInputStream is = null;
    static BufferedReader inputLine = null;
    static boolean closed = false;

    public static void main(String[] args)
    {
        int port_number=8989;
        String host="localhost";

        try {
            clientSocket = new Socket(host, port_number);
            inputLine = new BufferedReader(new InputStreamReader(System.in));
            os = new PrintStream(clientSocket.getOutputStream());
            is = new DataInputStream(clientSocket.getInputStream());
        } catch (Exception e) {
            System.out.println("Exception occurred : "+e.getMessage());
        }

        if (clientSocket != null && os != null && is != null)
        {
            try
            {
                new Thread(new Client()).start();
            }
        }
    }
}
```

```

        while (!closed)
        {
            os.println(inputLine.readLine());
        }

os.close();
is.close();
clientSocket.close();

    } catch (IOException e)
    {
        System.err.println("IOException: " + e);
    }
}

public void run()
{
    String responseLine;
    try
    {
        while ((responseLine = is.readLine()) != null)
        {
            System.out.println("\n"+responseLine);
            if (responseLine.equalsIgnoreCase("GLOBAL_COMMIT")==true ||
                responseLine.equalsIgnoreCase("GLOBAL_ABORT")==true )
            {
                break;
            }
        }
        closed=true;
    } catch (IOException e)
    {
        System.err.println("IOException: " + e);
    }
}
}

```

Server code:

```
import java.io.*;
```

```

import java.net.*;
import java.util.*;

public class Server
{
    boolean closed=false,inputFromAll=false;
    List<clientThread> t;
    List<String> data;

    Server()
    {
        t= new ArrayList<clientThread>();
        data= new ArrayList<String>();
    }

    public static void main(String args[])
    {
        Socket clientSocket = null;
        ServerSocket serverSocket = null;
        int port_number=8989;

        Server ser=new Server();

        try {
            serverSocket = new ServerSocket(port_number);
        }
        catch (IOException e)
        {System.out.println(e);}

        while(!ser.closed)
        {
            try {
                clientSocket = serverSocket.accept();
                clientThread th=new clientThread(ser,clientSocket);
                (ser.t).add(th);
                System.out.println("\n\nNow Total clients are : "+(ser.t).size());
                (ser.data).add("NOT_SENT");
                th.start();
            }
            catch (IOException e) {}
        }

        try
        {

```



```

        serverSocket.close();
    }catch(Exception e1){}

}
}

class clientThread extends Thread
{

    DataInputStream is = null;
    String line;
    String destClient="";
    String name;
    PrintStream os = null;
    Socket clientSocket = null;
    String clientIdentity;
    Server ser;

    public clientThread(Server ser,Socket clientSocket)
    {
        this.clientSocket=clientSocket;
        this.ser=ser;
    }

    public void run()
    {
        try
        {
            is = new DataInputStream(clientSocket.getInputStream());
            os = new PrintStream(clientSocket.getOutputStream());
            os.println("Enter your name.");
            name = is.readLine();
            clientIdentity=name;
            os.println("Welcome "+name+" to this 2 Phase Application.\nYou will receive a vote Request
            now...");
            os.println("VOTE_REQUEST\nPlease enter COMMIT or ABORT to proceed : ");
            for(int i=0; i<(ser.t).size(); i++)
            {
                if((ser.t).get(i)!=this)
                {
                    ((ser.t).get(i)).os.println("---A new user "+name+" entered the Appilcation---");
                }
            }
        }
    }
}

```

```

while (true)
{
    line = is.readLine();

    if(line.equalsIgnoreCase("ABORT"))
    {
        System.out.println("\nFrom '"+clientIdentity+"' : ABORT\n\nSince aborted we
will not wait for inputs from other clients.");
        System.out.println("\nAborted....");
        for(int i=0; i<(ser.t).size(); i++)
        {
            ((ser.t).get(i)).os.println("GLOBAL_ABORT");
            ((ser.t).get(i)).os.close();
            ((ser.t).get(i)).is.close();
        }

        break;
    }

    if(line.equalsIgnoreCase("COMMIT"))
    {
        System.out.println("\nFrom '"+clientIdentity+"' : COMMIT");
        if((ser.t).contains(this))
        {
            (ser.data).set((ser.t).indexOf(this), "COMMIT");
            for(int j=0;j<(ser.data).size();j++)
            {
                if(!(((ser.data).get(j)).equalsIgnoreCase("NOT_SENT")))
                {
                    ser.inputFromAll=true;
                    continue;
                }
            }
            else
            {
                ser.inputFromAll=false;
                System.out.println("\nWaiting for inputs from other clients.");
                break;
            }
        }

        if(ser.inputFromAll)
        {

```

```

        System.out.println("\n\nCommitted....");

        for(int i=0; i<(ser.t).size(); i++)
        {
            ((ser.t).get(i)).os.println("GLOBAL_COMMIT");
            ((ser.t).get(i)).os.close();
            ((ser.t).get(i)).is.close();
        }
        break;
    }

    }//if t.contains
    }//commit

    }//while
    ser.closed=true;
    clientSocket.close();

    } catch(IOException e){};
    }
}

```

Output

```

pict@pict-OptiPlex-390: ~/TwoPhaseCommit/src
pict@pict-OptiPlex-390:~/TwoPhaseCommit/src$ java Client

Enter your name.
Client2

Welcome Client2 to this 2 Phase Application.

You will receive a vote Request now...

VOTE_REQUEST

Please enter COMMIT or ABORT to proceed :
COMMIT

GLOBAL_COMMIT
☐

```

```
pict@pict-OptiPlex-390: ~/TwoPhaseCommit/src
pict@pict-OptiPlex-390:~/TwoPhaseCommit/src$ java Client

Enter your name.
Client1

Welcome Client1 to this 2 Phase Application.

You will receive a vote Request now...

VOTE_REQUEST

Please enter COMMIT or ABORT to proceed :

---A new user Client2 entered the Appilcation---
COMMIT

GLOBAL_COMMIT
```

```
pict@pict-OptiPlex-390: ~/TwoPhaseCommit/src
pict@pict-OptiPlex-390:~/TwoPhaseCommit/src$ java Server

Now Total clients are : 1

Now Total clients are : 2

From 'Client2' : COMMIT

Waiting for inputs from other clients.

From 'Client1' : COMMIT

Committed....
```

Conclusion: The two phase commit protocol is a distributed algorithm which lets all sites in a distributed system agree to commit or rollback a transaction based upon consensus of all participating sites. The two phase commit strategy is designed to ensure that either all the

databases are updated or none of them, so that the databases remain synchronized. The protocol achieves its goal even in many cases of temporary system.

References:

[1] Internet: https://en.wikipedia.org/wiki/Two-phase_commit_protocol.

[2] Internet: <http://slideplayer.com/slide/5092309/>

—

LAB 6: Design Persistent Objects using JDO and implement min 10 queries on objects using JDOQL in ObjectDB NOSQL DATABASE

Objective:

1. To learn and understand Database Modeling, Architectures
2. To learn and understand web database language, XML, JDOQL.

Introduction

Persistent objects are the fundamental logical units of data storage in an Objectivity/DB federated database. Persistent objects are created by applications. Each persistent object has an object identifier (OID) and is an instance of a class.

Java Data Objects (JDO) is a specification of Java object persistence. One of its features is a transparency of the persistence services to the domain model. The JDO object model is determined by a set of Java classes and XML metadata file. The metadata file contains modeling directives that either override the semantics specified in Java or provides semantics that cannot be expressed in Java. An enhancer is provided that enhances the Java classes based on these modeling directives.

The JDO specification provides a standard interface for accessing, storing, and processing persistent objects. The primary aspects in the JDO model are:

- **PersistenceManager:** PersistenceManagers negotiate accesses, transactions, and queries between applications and the underlying data store. More on the JDO PersistenceManager.
- **Transaction:** Transactions provide for atomic, consistent, isolated, and durable management of data (ACID properties). More on ACID properties.
- **Query:** The JDO Query Language (JDOQL) allows users to search for persistent objects matching specific criteria. JDOQL is meant to be query language neutral so that the underlying query language could be SQL, an object database query language such as OQL, or a specialized API to a hierarchical database or EIS system. More on the JDO Query Language.
- **PersistenceCapable classes:** The actual entities that are stored and fetched. There are three identity models to allow for different underlying database management systems. More on JDO identity models.

The JDO specification uses the Java language as much as possible, which allows the transparent integration of Java. This is illustrated by the following diagram and contrasts with the database sublanguage of SQL and its variants. In this diagram, you only see the host programming language and no database sublanguage or call-level interface as in JDBC.

Steps:

Install and configure *Cassandra*

Refer: <http://cassandra.apache.org/download/>

```
# Start cassandra daemon
~/cassandra/bin/cassandra -f
# The cassandra daemon should start in the foreground
# (don't press ctrl + c; as it'll terminate the daemon)
# In another terminal
~/cassandra/bin/cqlsh
# Create keyspace
cqlsh> CREATE KEYSPACE keyspace1
      WITH REPLICATION = { 'class' : 'SimpleStrategy',
replication_factor' : 1 };
cqlsh> USE keyspace1;
# Open Eclipse Editor
# Install DataNucleus plugin
In Eclipse-> Go to Help -> Install new software -> Add-> Set name as DataNucleus
Set Location as, http://www.datanucleus.org/downloads/eclipse-update/
->ok
After searching the plugin on internet, it will display one checkbox "DataNucleus
Eclipse plugin"
-> Check the checkbox->Next->Next->I accept->Finish-> Ignore the warning (if any) ->
Restart eclipse -> yes
# In eclipse ->Create new java project -> exp6
# Download DataNucleus AccessPlatform from
http://sourceforge.net/projects/datanucleus/files/datanucleus-accessplatform/4.2.3/datanucleusaccessplatform-cassandra-4.2.3.zip/download
# Extract the downloaded zip file
When you open the zip you will find DataNucleus jars in the lib directory, and dependency jars
in the deps directory.
# From the lib directory copy all jar files except following
datanucleus-jodatime-4.1.1.jar
datanucleus-java8-4.2.1.jar
datanucleus-guava-4.1.3.jar
# In eclipse -> right click on project exp6 -> Paste
# One by one, right click on pasted jarfile -> Build path -> add to build path
# Add all pasted jar files to build path
# Similarly from the deps folder copy all jar files
# Once again In eclipse -> right click on project exp6 -> Paste
# One by one, right click on pasted jarfile -> Build path -> add to build path
# Add all pasted jar files to build path
# In eclipse -> Create a new class - > Product -> add following code in it
package exp6;
import javax.jdo.annotations.IdGeneratorStrategy;
```

```

import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;
@PersistenceCapable
public class Product
{
    @PrimaryKey
    @Persistent(valueStrategy=IdGeneratorStrategy.INCREMENT)
    long id;
    String name = null;
    String description = null;
    double price = 0.0;
    public Product(String name, String desc, double price)
    {
        this.name = name;
        this.description = desc;
        this.price = price;
    }
}
# save the file
# In eclipse -> Create a new class - > Book -> add following code in it
package exp6;
import javax.jdo.annotations.PersistenceCapable;
@PersistenceCapable
public class Book extends Product
{ String author=null;
  String isbn=null;
  String publisher=null;
  public Book(String name, String desc, double price, String author,
    String isbn, String publisher)
  { super(name,desc,price);
    this.author = author;
    this.isbn = isbn;
    this.publisher = publisher;
  }
}
# save the file
# In eclipse -> Create a new class - > Inventory -> add following code in it
package exp6;
import java.util.HashSet;
import java.util.Set;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.PrimaryKey;
@PersistenceCapable

```



```

public class Inventory
{ @PrimaryKey
String name = null;
@SuppressWarnings({ "rawtypes", "unchecked" })
Set<Product> products = new HashSet();
public Inventory(String name)
{
this.name = name;
}
public Set<Product> getProducts() {return products;}
}

```

save the file

Plugin configuration

Right click on exp6 project-> Properties-> DataNucleus -> check "Enable project specific settings" ->Add JARS- > add all 18 jar files (10 from lib folder, 8 from deps folder)-> apply -> ok

Plugin configuration – Enhancer

Right click on exp6 project-> Properties -> double click on DataNucleus -> **Enhancer** -> check "Enable project specific settings" ->check verbose mode -> check Capture Output -> Give persistence-unit name -> TEST

Plugin configuration – SchemaTool

Right click on exp6 project-> Properties ->Under DataNucleus -> SchemaTool-> check "Enable project specific settings" ->check verbose mode -> Give persistence-unit name -> TEST

Right-click project exp6 -> select DataNucleus->"Add DataNucleus Support"

Right-click on package exp6 -> DataNucleus-> select "Create JDO XML Metadata File" ->

Give Filename-> package.jdo -> finish

Right-click on package exp6 -> DataNucleus-> select "Create persistence.xml file"

Edit persistence.xml file

```

<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
<persistence-unit name="TEST">
<mapping-file>/home/s/workspace/exp6/src/exp6/package.jdo</mapping-file>
<properties>
<property name="javax.jdo.option.ConnectionURL" value="cassandra:"/>
<property name="javax.jdo.mapping.Schema" value="keyspace1"/>
<property name="datanucleus.schema.autoCreateAll" value="true"/>
</properties>
</persistence-unit>
</persistence>

```

Save the file

Download the cassandra-java driver from

<http://downloads.datastax.com/java-driver/cassandra-java-driver-2.0.2.tar.gz>

```

# Extract the downloaded file.
# Paste all jar files (total 12) from the extracted folder into your eclipse project
(Note :: 10 jar files are present under lib folder)
# In eclipse -> One by one, right click on pasted jarfile -> Build path -> add to build path
# Add all pasted jar files to the build path
# In eclipse -> Create new class- > MyApp -> add following code in it
package exp6;
import java.util.Iterator;
import java.util.List;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;
import javax.jdo.Query;
import javax.jdo.Transaction;
public class MyApp {
public static void main(String[] args)
{//TEST is the persistent unit name
PersistenceManagerFactory pmf =
JDOHelper.getPersistenceManagerFactory("TEST");
PersistenceManager pm = pmf.getPersistenceManager();
Transaction tx=pm.currentTransaction();
try
{
tx.begin();
Inventory inv = new Inventory("My Inventory");
Product product = new Product("Sony Discman", "A standard
discman from Sony", 49.99);
Product product1 = new Product("Sony xperia z1", "A smart
phone", 149.99);
inv.getProducts().add(product);
inv.getProducts().add(product1);
pm.makePersistent(inv);
Query q = pm.newQuery("SELECT FROM " + Product.class.getName()
+" WHERE price < 150.00 ORDER BY price ASC");
// add some more JDOQL queries here
List<Product> products = (List<Product>)q.execute();
Iterator<Product> iter = products.iterator();
while (iter.hasNext())
{
Product p = iter.next();
System.out.println("Name: "+p.name+"\t Description:
"+p.description+"\tPrice: "+p.price);
}
tx.commit();
}
}

```

```

} finally
{
if (tx.isActive())
{
tx.rollback();
}
pm.close();
}
}
}
}
# Save the file
# Right-click on project exp6 -> DataNucleus-> select "Run Enhancer tool"
# Output
ENHANCED (Persistable) : exp6.Product
ENHANCED (Persistable) : exp6.Inventory
ENHANCED (Persistable) : exp6.Book
DataNucleus Enhancer completed with success for 3 classes.
# Make sure that cassandra is running before doing next step
# Right-click on project exp6 -> DataNucleus-> select "Run Schema tool" -> Set Connection
URL -> cassandra: -> next -> finish
# Output
DataNucleus SchemaTool : Input Files
>> /home/s/workspace1/exp6/bin/exp6/Book.class
>> /home/s/workspace1/exp6/bin/exp6/Inventory.class
>> /home/s/workspace1/exp6/bin/exp6/Product.class
>> /home/s/workspace1/exp6/bin/exp6/MyApp.class
>> /home/s/workspace1/exp6/bin/exp6/package.jdo
>> /home/s/workspace1/exp6/src/exp6/package.jdo
DataNucleus SchemaTool completed successfully
# Right click on MyApp.java -> Run as -> Java Application
# Output
log4j:WARN No appenders could be found for logger (DataNucleus.General).
log4j:WARN Please initialize the log4j system properly.
Name: Sony Discman Description: A standard discman from Sony Price: 49.99
Name: Sony xperia z1 Description: A smart phone Price: 149.99
# In cqlsh prompt, verify the output
cqlsh> SELECT * FROM Product;
# Output
id | description | name | price
-----+-----+-----+-----
41 | A standard discman from Sony | Sony Discman | 49.99
42 | A smart phone | Sony xperia z1 | 149.99
# Now edit MyApp.java to add some more JDOQL queries one by one (minimum 10 are
expected)

```

Then, Run MyApp.java
Verify the output in cqlsh prompt by using SELECT query.
A very good tutorial on JDOQL queries is available in following link,
<https://cloud.google.com/appengine/docs/java/datastore/jdo/queries>

Conclusion: The Java Data Objects (JDO) specification, Java Specification Request (JSR) 12, defines an API for a standard way to transparently persist plain Java technology object and database access. The JDO specification requires that vendors provide a query capability using JDOQL, which is a query language oriented around the objects that are persisted. The Persistence Manager class defines methods for constructing instances of Query-instance implementation classes. A query filter is specified as a boolean expression that resembles SQL's boolean operators.

References:

<http://www.datanucleus.org/products/datanucleus/jdo/guides/eclipse.html>
http://www.datanucleus.org/products/datanucleus/jdo/samples/tutorial_cassandra.html
<http://www.datanucleus.org/products/datanucleus/datastores/cassandra.html>
<https://cloud.google.com/appengine/docs/java/datastore/jdo/queries>
http://www.service-architecture.com/articles/database/java_data_objects_jdo.html

LAB 7: Create XML, XML schemas , DTD for any database application and implement min 10 queries using XQuery FLOWR expression and XPath

Objective:

1. To learn and understand web database language, XML, JDOQL.

Introduction:

XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the W3C's XML 1.0 Specification and by several other related specifications all of which are free open standards. The design goals of XML emphasize simplicity, generality and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services. Several schema systems exist to aid in the definition of XML-based languages, while many application programming interfaces (APIs) have been developed to aid the processing of XML data.

XML Schema

An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself. These constraints are generally expressed using some combination of grammatical rules governing the order of elements, Boolean predicates that the content must satisfy, data types governing the content of elements and attributes, and more specialized rules such as uniqueness and referential integrity constraints. There are languages developed specifically to express XML schemas. The Document Type Definition (DTD) language, which is native to the XML specification, is a schema language that is of relatively limited capability, but that also has other uses in XML aside from the expression of schemas. Two more expressive XML schema languages in widespread use are XML Schema (with a capital S) and RELAX NG. The mechanism

for associating an XML document with a schema varies according to the schema language. The association may be achieved via markup within the XML document itself, or via some external means.

DTD

A document type definition (DTD) is a set of markup declarations that define a document type for an SGML-family markup language (SGML, XML, HTML). A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference. XML uses a subset of SGML DTD. As of 2009, newer XML namespace-aware schema languages (such as W3C XML Schema and ISO RELAX NG) have largely superseded DTDs. A namespace-aware version of DTDs is being developed as Part 9 of ISO DSDL. DTDs persist in applications that need special publishing characters, such as the XML and HTML Character Entity References, which derive from larger sets defined as part of the ISO SGML standard effort.

Input Code:

```
import javax.xml.parsers.*;
//import oracle.hadoop.xquery.*;
import java.io.*;
import org.xml.sax.SAXException;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import com.saxonica.xqj.SaxonXQDataSource;
import javax.xml.xpath.*;
import javax.xml.xquery.*;
import java.util.*;
class XMLDemo
{
```

```

public static void main(String[] args)
{
    DocumentBuilderFactory
builderFactory=DocumentBuilderFactory.newInstance();
    DocumentBuilder builder=null;
    Document document=null;
    XPath xPath=XPathFactory.newInstance().newXPath();
    String expression=null;
    Scanner sc=new Scanner(System.in);
    Try
    {
        builder=builderFactory.newDocumentBuilder();
        document=builder.parse(new FileInputStream("D:\\xml_parsing\\emplo
yees.xml"));
        //first query
        System.out.println("\nEnter employee id to get location:");
        int id=sc.nextInt();
        expression="/employees/employee[@id='"+id+"']/location";
        String location=xPath.compile(expression).evaluate(document);
        if(!location.equals(""))
            System.out.println("Location of Employee having id='"+id+"':"+location);
        else
            System.out.println("Employee id '"+id+"' does not exist");
        //second query
        System.out.println("\nReading firstname of all employees:");
        expression="/employees/employee/firstname";
        NodeList nodeList = (NodeList)
xPath.compile(expression).evaluate(document, XPathConstants.NODESET);
        for (int i = 0; i < nodeList.getLength(); i++)

```

```

System.out.println(nodeList.item(i).getFirstChild().getNodeValue());

//third query
System.out.println("\nEnter employee id:");
id=sc.nextInt();
expression="employees/employee[@id='"+id+"']";
Node node = (Node) XPath.compile(expression).evaluate(document,
XPathConstants.NODE);
if(null != node) {
    nodeList = node.getChildNodes();
    for (int i = 0;null!=nodeList && i < nodeList.getLength(); i++) {
        Node nod = nodeList.item(i);
        if(nod.getNodeType() == Node.ELEMENT_NODE)
            System.out.println(nodeList.item(i).getNodeName() + " : " +
nod.getFirstChild().getNodeValue());
    }
}

//Fourth Query
System.out.println("\nEmployees firstname whose age<40");
expression = "/employees/employee[age<40]/firstname";
nodeList = (NodeList) XPath.compile(expression).evaluate(document,
XPathConstants.NODESET);
for (int i = 0; i < nodeList.getLength(); i++)
{
    System.out.println(nodeList.item(i).getFirstChild().getNodeValue());
}

//Five Query
System.out.println("\nFirst Employee registered");
expression="/employees/employee[1]";
node=(Node)XPath.compile(expression).evaluate(document,XPathConstants.NODE);

```



```

        if(null != node)
        {
            nodeList = node.getChildNodes();
            for (int i = 0;null!=nodeList && i < nodeList.getLength(); i++)
            {
                Node nod = nodeList.item(i);
                if(nod.getNodeType() == Node.ELEMENT_NODE)
                    System.out.println(nodeList.item(i).getNodeName() + " : " +
nod.getFirstChild().getNodeValue());
            }
        }
//Sixth Query
System.out.println("\nLast Employee registered");
expression="/employees/employee[last()]";
node=(Node)xPath.compile(expression).evaluate(document,
XPathConstants.NODE);
if(null != node)
{
    nodeList = node.getChildNodes();
    for (int i = 0;null!=nodeList && i < nodeList.getLength(); i++)
    {
        Node nod = nodeList.item(i);
        if(nod.getNodeType() == Node.ELEMENT_NODE)
            System.out.println(nodeList.item(i).getNodeName() + " : " +
nod.getFirstChild().getNodeValue());
    }
}
//Seventh Query
System.out.println("\nFirstname of all Technical Assistant");

```

```

        Expression = "/employees/employee[designation='Team Lead']/firstname";
        nodeList = (NodeList) XPath.compile(expression).evaluate(document,
XPathConstants.NODESET);
        for (int i = 0; i < nodeList.getLength(); i++)
        {
            System.out.println(nodeList.item(i).getFirstChild().getNodeValue());
        }

        //Eighth Query
        System.out.println("\nFirstname of all Manager");
        Expression = "/employees/employee[designation='Manager']/firstname";
        nodeList = (NodeList) XPath.compile(expression).evaluate(document,
XPathConstants.NODESET);
        for (int i = 0; i < nodeList.getLength(); i++)
        {
            System.out.println(nodeList.item(i).getFirstChild().getNodeValue());
        }

        //Nineth Query
        System.out.println("\nFirstname of all Intern");
        expression = "/employees/employee[designation='Intern']/firstname";
        nodeList = (NodeList) XPath.compile(expression).evaluate(document,
XPathConstants.NODESET);
        for (int i = 0; i < nodeList.getLength(); i++)
        {
            System.out.println(nodeList.item(i).getFirstChild().getNodeValue());
        }

        //Tenth Query
        System.out.println("\nFirstname of all Technical Assistant");
        Expression = "/employees/employee[designation='Technical Assistant'
]/firstname";

```

```

        nodeList = (NodeList) XPath.compile(expression).evaluate(document,
XPathConstants.NODESET);
        for (int i = 0; i < nodeList.getLength(); i++)
        {
            System.out.println(nodeList.item(i).getFirstChild().getNodeValue());
        }
    }
}
catch(ParserConfigurationException e)
{
    e.printStackTrace();
}
catch(IOException e)
{
    e.printStackTrace();
}
catch(SAXException e)
{
    e.printStackTrace();
}
catch(XPathExpressionException e)
{e.printStackTrace();}
}}

```

OUTPUT:

```
<terminated> XMLDemo [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\ja

Enter employee id to get location:
111
Location of Employee having id=111:Bangalore

Reading firstname of all employees:
Rakesh
John
Rajesh
Rahul

Enter employee id:
111
firstname : Rakesh
lastname : Mishra
age : 23
mobilenos : 9856321475
email : rakeshm@gmail.com
location : Bangalore
designation : Intern

Employees firstname whose age<40
Rakesh
Rajesh
Rahul
```

First Employee registered
firstname : Rakesh
lastname : Mishra
age : 23
mobilenos : 9856321475
email : rakeshm@gmail.com
location : Bangalore
designation : Intern

Last Employee registered
firstname : Rahul
lastname : Salvi
age : 24
mobilenos : 9757321475
email : rahul11@gmail.com
location : Pune
designation : Team Lead

Firstname of all Technical Assistant
Rahul

Firstname of all Manager
John

Firstname of all Intern
Rakesh

Firstname of all Technical Assistant
Rajesh

Conclusion:

XML,XML scemas,DTD for employee database application is created and queries are implemented using using XQuery, Flower expression and XPath.

REFERENCES:

- [1] Internet : http://en.wikipedia.org/wiki/XML_schema
- [2] Internet: https://en.wikipedia.org/wiki/Document_type_definition
- [3] Internet: <http://searchsoa.techtarget.com/definition/Document-Type-Definition>

LAB 8a: Design database schemas and implement min 10 queries using Hive/ Hbase/ Cassandra column based databases

Objective:

1. To learn NoSQL Databases (Open source) such as Hive/ Hbase/ Cassandra/DynamoDB.

Hive:

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

- **Prerequisites:** Core Java,
Database concepts of SQL,
Hadoop File system, and any of Linux operating system
- **Features:**
 - It stores schema in a database and processed data into HDFS.
 - It is designed for OLAP.
 - It provides SQL type language for querying called HiveQL or HQL.
 - It is familiar, fast, scalable, and extensible.
- **Architecture:**

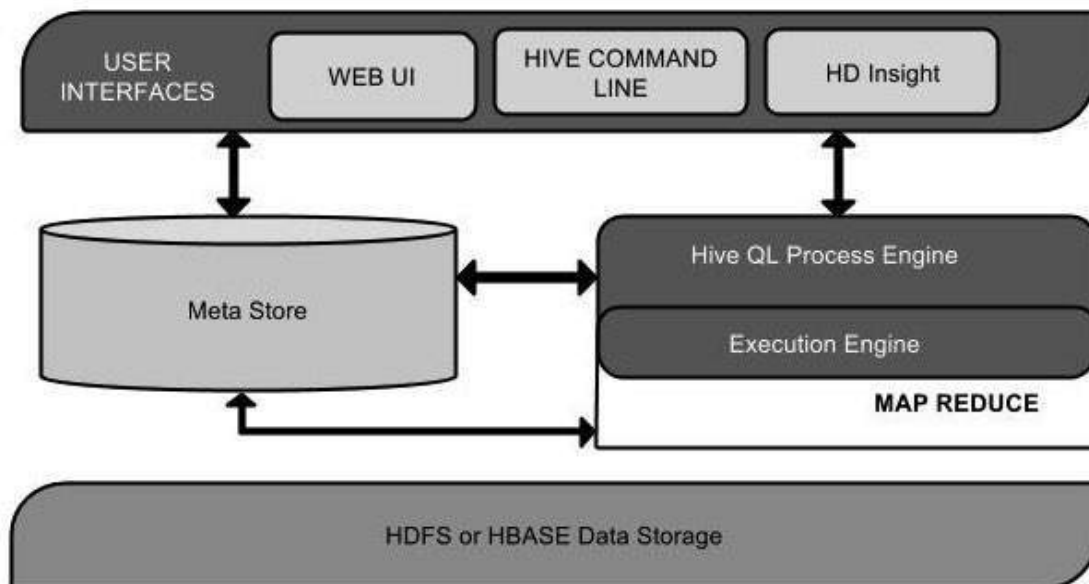


Figure 1: Hive Architecture

This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Table 1: Hive components and their operations.

- **Verifying Hadoop Installation :**

Hadoop must be installed on system before installing Hive. Verify the Hadoop installation using the following command:

```
$ hadoop version
```

If Hadoop is already installed on system, then you will get the following response:

```
Hadoop 2.4.1 Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
```

If Hadoop is not installed on your system, then download and configure the Hadoop.

- **Installing Hive:**

The following steps are required for installing Hive.

- **Extracting and verifying Hive Archive**

The following command is used to verify the download and extract the hive archive:

```
$ tar zxvf apache-hive-0.14.0-bin.tar.gz
$ ls
```

On successful download, we get the following response:

```
apache-hive-0.14.0-bin apache-hive-0.14.0-bin.tar.gz
```

- **Copying files to /usr/local/hive directory**

We need to copy the files from the super user "su -". The following commands are used to copy the files from the extracted directory to the /usr/local/hive" directory.

```
$ su -  
  
passwd:  
  
# cd /home/user/Download  
  
# mv apache-hive-0.14.0-bin /usr/local/hive  
  
# exit
```

- **Setting up environment for Hive**

We can set up the Hive environment by appending the following lines to `~/.bashrc` file:

```
export HIVE_HOME=/usr/local/hive  
export PATH=$PATH:$HIVE_HOME/bin  
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:..  
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:..
```

The following command is used to execute `~/.bashrc` file.

```
$ source ~/.bashrc
```

- **Configuring Hive:**

To configure Hive with Hadoop, we need to edit the **hive-env.sh** file, which is placed in the **\$HIVE_HOME/conf** directory. The following commands redirect to Hive **config** folder and copy the template file:

```
$ cd $HIVE_HOME/conf  
  
$ cp hive-env.sh.template hive-env.sh
```

Edit the **hive-env.sh** file by appending the following line:


```
export HADOOP_HOME=/usr/local/hadoop
```

Hive installation is completed successfully.

- **Creating Database:**

Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it.

Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

```
CREATE DATABASE | SCHEMA [IF NOT EXISTS] <database name>
```

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

or

```
hive> CREATE SCHEMA userdb;
```

The following query is used to verify a databases list:

```
hive> SHOW DATABASES;
```

```
default
```

```
userdb
```

Drop Database Statement

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

```
DROP DATABASE Statement DROP (DATABASE | SCHEMA) [IF EXISTS] database_name  
[RESTRICT | CASCADE];
```

The following queries are used to drop a database. Let us assume that the database name is **userdb**.

```
hive> DROP DATABASE IF EXISTS userdb;
```

The following query drops the database using **CASCADE**. It means dropping respective tables before dropping the database.

```
hive> DROP DATABASE IF EXISTS userdb CASCADE;
```

The following query drops the database using **SCHEMA**.

```
hive> DROP SCHEMA userdb;
```

Create Table Statement

Create Table is a statement used to create a table in Hive.

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name  
  
[(col_name data_type [COMMENT col_comment], ...)]  
[COMMENT table_comment]  
[ROW FORMAT row_format]  
[STORED AS file_format]
```

Load Data Statement

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement. While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system. The syntax for load data is as follows:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename  
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

Alter Table Statement

It is used to alter a table in Hive. The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

```
ALTER TABLE name RENAME TO new_name  
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])  
ALTER TABLE name DROP [COLUMN] column_name  
ALTER TABLE name CHANGE column_name new_name new_type  
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

Drop Table Statement

The syntax is as follows:

```
DROP TABLE [IF EXISTS] table_name;
```

On successful execution of the query, you get to see the following response:

```
OK  
Time taken: 5.3 seconds  
hive>
```

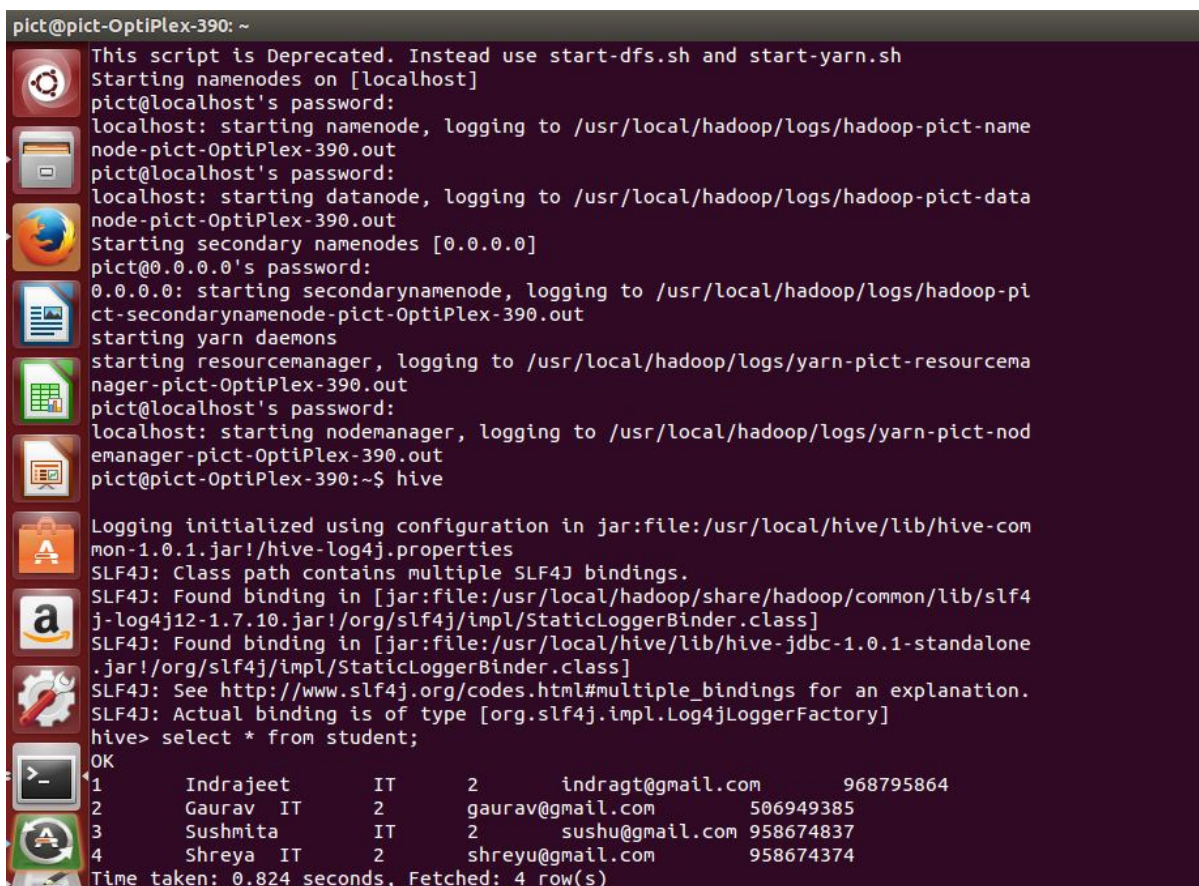
The Hive Query Language (HiveQL) is a query language for Hive to process and analyze structured data in a Metastore. SELECT statement is used to retrieve the data from a table.

WHERE clause works similar to a condition. It filters the data using the condition and gives a finite result. The built-in operators and functions generate an expression, which fulfils the condition. Below is the syntax of the SELECT query:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]
```

3. Implementation:

- Screenshot no.1



The screenshot shows a terminal window on a system named 'pict-OptiPlex-390'. The user is running a script to start Hadoop services. The output shows the starting of namenodes, datanodes, and secondary namenodes, followed by starting yarn daemons, resource manager, and node manager. The user then runs 'hive' and enters the Hive CLI. The Hive CLI shows the logging initialized and SLF4J bindings. The user enters the query 'select * from student;' and the result is displayed as a table with 5 columns and 4 rows.

```
pict@pict-OptiPlex-390: ~
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
pict@localhost's password:
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-pict-name
node-pict-OptiPlex-390.out
pict@localhost's password:
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-pict-data
node-pict-OptiPlex-390.out
Starting secondary namenodes [0.0.0.0]
pict@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-pi
ct-secondarynamenode-pict-OptiPlex-390.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-pict-resourcemana
ger-pict-OptiPlex-390.out
pict@localhost's password:
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-pict-nod
emanager-pict-OptiPlex-390.out
pict@pict-OptiPlex-390:~$ hive

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-com
mon-1.0.1.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4
j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/hive-jdbc-1.0.1-standalone
.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hive> select * from student;
OK
1      Indrajeet      IT      2      indragt@gmail.com      968795864
2      Gaurav      IT      2      gaurav@gmail.com      506949385
3      Sushmita      IT      2      sushu@gmail.com      958674837
4      Shreya      IT      2      shreyu@gmail.com      958674374
Time taken: 0.824 seconds, Fetched: 4 row(s)
```

- Screenshot no.2

```
pict@pict-OptiPlex-390: ~
Time taken: 0.824 seconds, Fetched: 4 row(s)
hive> select distinct name from student;
Query ID = pict_20151127110404_cb0740a5-908d-43bd-a658-04924b87aeb3
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1448602432016_0001, Tracking URL = http://pict-OptiPlex-390:8088/proxy/application_1448602432016_0001/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1448602432016_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-11-27 11:04:52,792 Stage-1 map = 0%, reduce = 0%
2015-11-27 11:04:58,049 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.86 sec
2015-11-27 11:05:04,288 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.15 sec
MapReduce Total cumulative CPU time: 2 seconds 150 msec
Ended Job = job_1448602432016_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.15 sec HDFS Read: 382 HDFS Write: 33 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 150 msec
OK
Gaurav
Indrajeet
Shreya
Sushmita
```

- Screenshot no.3

```
pict@pict-OptiPlex-390: ~
Time taken: 0.112 seconds
hive> select * from student where year>1;
OK
1      Indrajeet      IT      2      indragt@gmail.com      968795864
2      Gaurav IT      2      gaurav@gmail.com      506949385
3      Sushmita      IT      2      sushu@gmail.com      958674837
4      Shreya IT      2      shreyu@gmail.com      958674374
Time taken: 0.053 seconds, Fetched: 4 row(s)
hive> select Ucase(name) from student;
OK
INDRAJEET
GAURAV
SUSHMITA
SHREYA
Time taken: 0.057 seconds, Fetched: 4 row(s)
```


- Screenshot no.4

```
pict@pict-OptiPlex-390: ~
hive> select rollno,name,dept from student order by rollno;
Query ID = pict_20151127110808_c2a293e2-d553-4e03-8a91-5f31b4e2eef6
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1448602432016_0002, Tracking URL = http://pict-OptiPlex-390:8088/proxy/application_1448602432016_0002/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1448602432016_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-11-27 11:08:07,100 Stage-1 map = 0%, reduce = 0%
2015-11-27 11:08:12,284 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.86 sec
2015-11-27 11:08:18,509 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.09 sec
MapReduce Total cumulative CPU time: 2 seconds 90 msec
Ended Job = job_1448602432016_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.09 sec HDFS Read: 382 HDFS Write: 53 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 90 msec
OK
1      Indrajeet      IT
2      Gaurav      IT
3      Sushmita      IT
4      Shreya      IT
Time taken: 17.287 seconds, Fetched: 4 row(s)
```

- Screenshot no.5

```
hive> select dept,count(*) from student group by dept;
Query ID = pict_20151127111111_ebadd014-8e94-46dc-be42-c45d620f5df9
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1448602432016_0003, Tracking URL = http://pict-OptiPlex-390:8088/proxy/application_1448602432016_0003/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1448602432016_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-11-27 11:11:22,833 Stage-1 map = 0%, reduce = 0%
2015-11-27 11:11:28,005 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.96 sec
2015-11-27 11:11:34,221 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.27 sec
MapReduce Total cumulative CPU time: 2 seconds 270 msec
Ended Job = job_1448602432016_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.27 sec HDFS Read: 382 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 270 msec
OK
IT      4
Time taken: 17.336 seconds, Fetched: 1 row(s)
```

- Screenshot no.6

```
hive> select name,emailid,dept from student where year=2;
OK
Indrajeet      indragt@gmail.com      IT
Gaurav gaurav@gmail.com      IT
Sushmita      sushu@gmail.com      IT
Shreya shreyu@gmail.com      IT
Time taken: 0.075 seconds, Fetched: 4 row(s)
```

- **Screenshot no.7**

```
hive> select rollno,name,dept from student order by rollno;
Query ID = pict_20151127111616_238713b7-6337-4226-8468-9a6b0ab34a7d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1448602432016_0004, Tracking URL = http://pict-OptiPlex-390:8088/proxy/application_1448602432016_0004/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1448602432016_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-11-27 11:16:14,923 Stage-1 map = 0%, reduce = 0%
2015-11-27 11:16:19,104 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.93 sec
2015-11-27 11:16:25,316 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.22 sec
MapReduce Total cumulative CPU time: 2 seconds 220 msec
Ended Job = job_1448602432016_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.22 sec HDFS Read: 382 HDFS Write: 53 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 220 msec
OK
1      Indrajeet      IT
2      Gaurav      IT
3      Sushmita      IT
4      Shreya      IT
Time taken: 18.317 seconds, Fetched: 4 row(s)
```


- Screenshot no.8

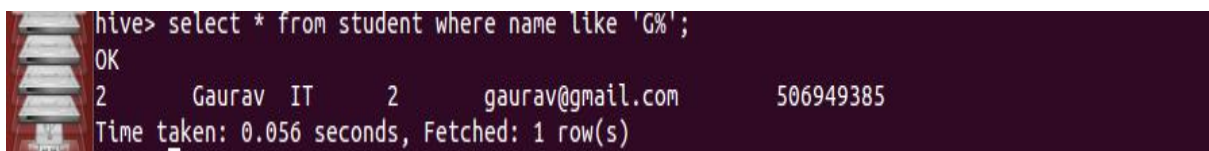


```

hive> select rollno from student order by rollno desc;
Query ID = pict_20151127111717_a81dd5f9-078d-4668-b7af-da4cb3ee7c39
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1448602432016_0005, Tracking URL = http://pict-OptiPlex-390:8088/proxy/application_1448602432016_0005/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1448602432016_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-11-27 11:17:49,062 Stage-1 map = 0%, reduce = 0%
2015-11-27 11:17:54,226 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.83 sec
2015-11-27 11:18:00,436 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.11 sec
MapReduce Total cumulative CPU time: 2 seconds 110 msec
Ended Job = job_1448602432016_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.11 sec HDFS Read: 382 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 110 msec
OK
4
3
2
1
Time taken: 18.271 seconds, Fetched: 4 row(s)

```

- Screenshot no.9

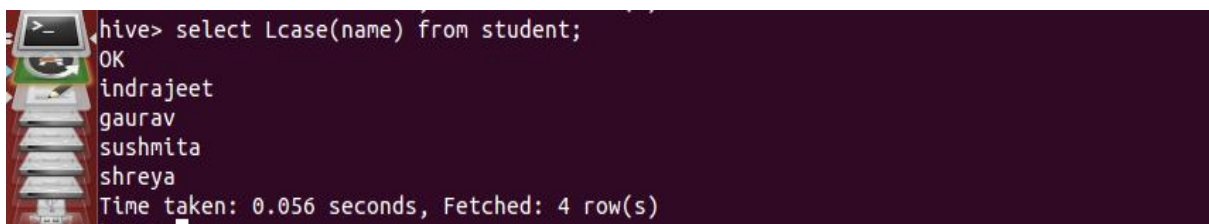


```

hive> select * from student where name like 'G%';
OK
2      Gaurav IT      2      gaurav@gmail.com      506949385
Time taken: 0.056 seconds, Fetched: 1 row(s)

```

- Screenshot no.10



```

hive> select Lcase(name) from student;
OK
indrajeet
gaurav
sushmita
shreya
Time taken: 0.056 seconds, Fetched: 4 row(s)

```


Conclusion: Hive is best suited for Data Warehousing Applications where data is stored, mined and reporting is done based on processing. As most Data Warehousing applications are based on relational database models, Hive bridges the gap between these applications and Hadoop.

References:

[1] Internet: <http://www.tutorialspoint.com/hive/index.htm>

[2] Internet: <http://hadooptutorials.co.in/tutorials/hive/introduction-to-apache-hive.html>

—

LAB 8b: Design database schemas and implement min 10 queries using Hive/ Hbase/ Cassandra column based databases

Objective:

1. To learn NoSQL Databases (Open source) such as Hive/ Hbase/ Cassandra/DynamoDB.

Introduction

Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It is a type of NoSQL database. Let us first understand what a NoSQL database does.

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Listed below are some of the notable points of Apache Cassandra:

- It is scalable, fault-tolerant, and consistent.
- It is a column-oriented database.
- Its distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable.
- Created at Facebook, it differs sharply from relational database management systems.
- Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful "column family" data model.
- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- **Elastic scalability** - Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- **Always on architecture** - Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.

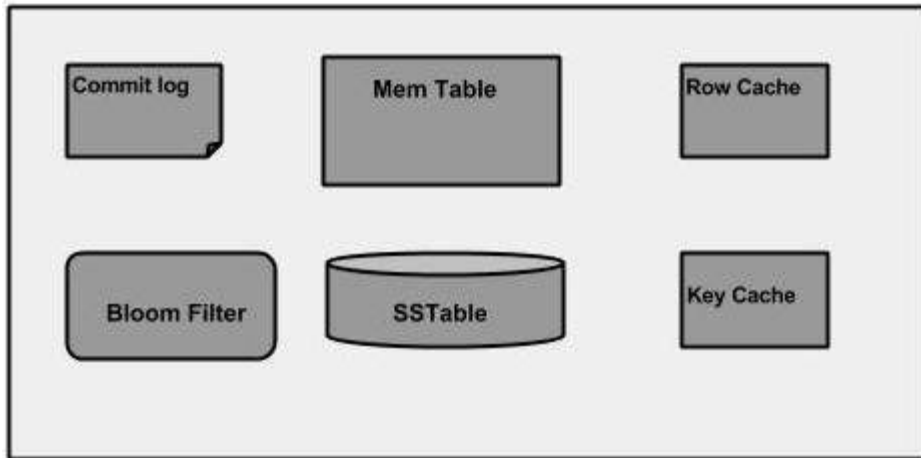
- **Fast linear-scale performance** - Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- **Flexible data storage** - Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- **Easy data distribution** - Cassandra provides the flexibility to distribute data where you need by replicating data across multiple data centers.
- **Transaction support** - Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- **Fast writes** - Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

History of Cassandra

- Cassandra was developed at Facebook for inbox search.
- It was open-sourced by Facebook in July 2008.
- Cassandra was accepted into Apache Incubator in March 2009.
- It was made an Apache top-level project since February 2010.

Cassandra - Architecture

The design goal of Cassandra is to handle big data workloads across multiple nodes without single point of failure. Cassandra has peer-to-peer distributed system across nodes, and data is distributed among all nodes in the cluster. The following figure shows the architecture of Cassandra.



Components of Cassandra

Given below are the key components of Cassandra:

- **Node** - It is the place where data is stored.
- **Data center** - It is a collection of related nodes.
- **Commit log** - The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- **Cluster** - A cluster is a component that contains one or more data centers.
- **Mem-table** - A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable** - It is a disk file, to which the data is flushed to, from mem-table, when its contents reach a threshold value.
- **Bloom filter** - These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.
- **Compaction** - The process of freeing up space by merging large accumulated data files is called compaction. During compaction, the data is merged, indexed, sorted, and stored in a new SSTable. Compaction also reduces the number of required seeks.

Users can access Cassandra through nodes using **Cassandra Query Language**. CQL treats the database (**Keyspace**) as a container of tables. Programmers use **cqlsh**: a prompt to work with CQL or separate application language drivers.

Clients approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

Write Operations

Every write activity of nodes is captured by the **commit logs** written in the nodes. Later the data will be captured and stored in the **mem-table**. Whenever the memtable is full, data will be written into the **SStable** data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates SSTables, discarding unnecessary data.

Read Operations

During read operations, Cassandra gets values from the mem-table, checks bloom filter to find appropriate SSTables, and gets values from SSTables.

Cassandra - Data Model

Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica; in case of a failure, replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. The basic attributes of Keyspace in Cassandra are:

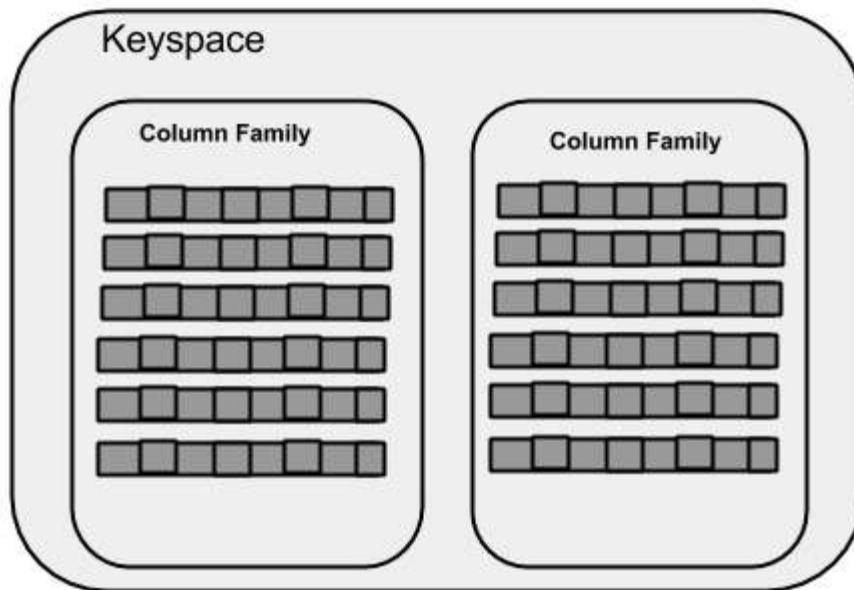
- **Replication factor** - It is the number of machines in the cluster that will receive copies of the same data.
- **Replica placement strategy** - It is nothing but the strategy to place replicas in the ring. We have strategies such as **simple strategy** (rackaware strategy), **old network topology strategy** (rack-aware strategy), and **network topology strategy** (datacenter-shared strategy).

- **Column families** - Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows:

```
CREATE KEYSPACE Keyspace name
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

Given below is the schematic view of a Keyspace.



Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

Relational Table	Cassandra column Family
A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be	In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column

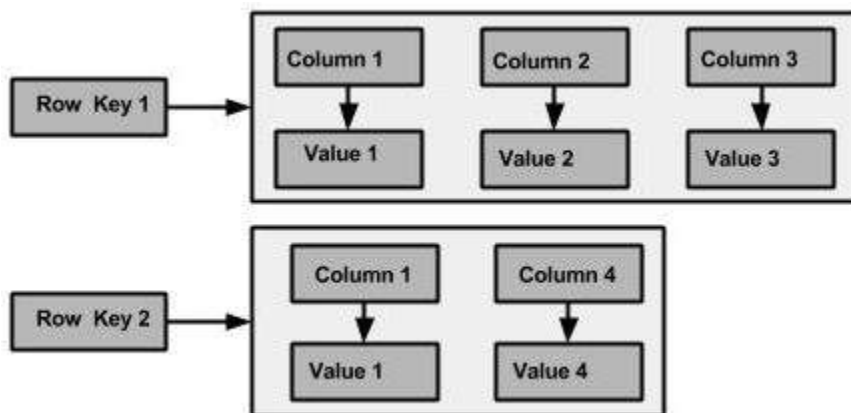
filled at least with a null value.	family at any time.
Relational tables define only columns and the user fills in the table with values.	In Cassandra, a table contains columns, or can be defined as a super column family.

A Cassandra column family has the following attributes:

- **keys_cached** - It represents the number of locations to keep cached per SSTable.
- **rows_cached** - It represents the number of rows whose entire contents will be cached in memory.
- **preload_row_cache** - It specifies whether you want to pre-populate the row cache.

Note: Unlike relational tables where a column family's schema is not fixed, Cassandra does not force individual rows to have all the columns.

The following figure shows an example of a Cassandra column family.



Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

Column		
name : byte[]	value : byte[]	clock : clock[]

SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns.

Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same column family, and a super column can be helpful here. Given below is the structure of a super column.

Super Column	
name : byte[]	cols : map<byte[], column>

Data Models of Cassandra and RDBMS

Below given is the difference between complete data model of RDBMS and cassandra.

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays.(ROW x COLUMN)	In Cassandra, a table is a list of "nested key-value pairs". (ROW x COLUMN key x COLUMN value)
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.

Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

Cassandra - Installation

Cassandra can be accessed using cqlsh as well as drivers of different languages. This chapter explains how to set up both cqlsh and java environments to work with Cassandra.

Pre-Installation Setup

Before installing Cassandra in Linux environment, we require to set up Linux using **ssh** (Secure Shell). Follow the steps given below for setting up Linux environment.

Create a User

At the beginning, it is recommended to create a separate user for Hadoop to isolate Hadoop file system from Unix file system. Follow the steps given below to create a user.

- Open root using the command **"su"**.
- Create a user from the root account using the command **"useradd username"**.
- Now you can open an existing user account using the command **"su username"**.

Open the Linux terminal and type the following commands to create a user.

```
$ su
password:
# useradd hadoop
# passwd hadoop
New passwd:
Retype new passwd
```

SSH Setup and Key Generation

SSH setup is required to perform different operations on a cluster such as starting, stopping, and distributed daemon shell operations. To authenticate different users of Hadoop, it is required to provide public/private key pair for a Hadoop user and share it with different users.

The following commands are used for generating a key value pair using SSH:

- copy the public keys from id_rsa.pub to authorized_keys,
- and provide owner,
- read and write permissions to authorized_keys file respectively.

```
$ ssh-keygen -t rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

- Verify ssh:

```
ssh localhost
```

Installing Java

Java is the main prerequisite for Cassandra. First of all, you should verify the existence of Java in your system using the following command:

```
$ java -version
```

If everything works fine it will give you the following output.

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

If you don't have Java in your system, then follow the steps given below for installing Java.

Step 1

Download java (JDK <latest version> - X64.tar.gz) from the following [link](#):

Then jdk-7u71-linux-x64.tar.gz will be downloaded onto your system.

Step 2

Generally you will find the downloaded java file in the Downloads folder. Verify it and extract the **jdk-7u71-linux-x64.gz** file using the following commands.

```
$ cd Downloads/  
$ ls  
jdk-7u71-linux-x64.gz  
$ tar xzf jdk-7u71-linux-x64.gz  
$ ls  
jdk1.7.0_71 jdk-7u71-linux-x64.gz
```

Step 3

To make Java available to all users, you have to move it to the location `"/usr/local/"`. Open root, and type the following commands.

```
$ su  
password:  
# mv jdk1.7.0_71 /usr/local/  
# exit
```

Step 4

For setting up **PATH** and **JAVA_HOME** variables, add the following commands to `~/.bashrc` file.

```
export JAVA_HOME = /usr/local/jdk1.7.0_71  
export PATH = $PATH:$JAVA_HOME/bin
```

Now apply all the changes into the current running system.

```
$ source ~/.bashrc
```

Step 5

Use the following commands to configure java alternatives.

```
# alternatives --install /usr/bin/java java usr/local/java/bin/java 2  
# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2  
# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2  
  
# alternatives --set java usr/local/java/bin/java  
# alternatives --set javac usr/local/java/bin/javac
```

```
# alternatives --set jar usr/local/java/bin/jar
```

Now use the **java -version** command from the terminal as explained above.

Setting the Path

Set the path of Cassandra path in `"/.bashrc"` as shown below.

```
[hadoop@linux ~]$ gedit ~/.bashrc
export CASSANDRA_HOME = ~/cassandra
export PATH = $PATH:$CASSANDRA_HOME/bin
```

Download Cassandra

Apache Cassandra is available at <http://cassandra.apache.org/download/> Cassandra using the following command.

```
$ wget http://supergsego.com/apache/cassandra/2.1.2/apache-cassandra-2.1.2-bin.tar.gz
```

Unzip Cassandra using the command **zxvf** as shown below.

```
$ tar zxvf apache-cassandra-2.1.2-bin.tar.gz.
```

Create a new directory named **cassandra** and move the contents of the downloaded file to it as shown below.

```
$ mkdir Cassandra
$ mv apache-cassandra-2.1.2/* cassandra.
```

Configure Cassandra

Open the **cassandra.yaml** file, which will be available in the **bin** directory of Cassandra.

```
$ gedit cassandra.yaml
```

Note: If you have installed Cassandra from a deb or rpm package, the configuration files will be located in **/etc/cassandra** directory of Cassandra.

The above command opens the **cassandra.yaml** file. Verify the following configurations. By default, these values will be set to the specified directories.

- `data_file_directories "/var/lib/cassandra/data"`
- `commitlog_directory"/var/lib/cassandra/commitlog"`

- saved_caches_directory"/var/lib/cassandra/saved_caches"

Make sure these directories exist and can be written to, as shown below.

Create Directories

As super-user, create the two directories **/var/lib/cassandra** and **/var/log/cassandra** into which Cassandra writes its data.

```
[root@linux cassandra]# mkdir /var/lib/cassandra  
[root@linux cassandra]# mkdir /var/log/cassandra
```

Give Permissions to Folders

Give read-write permissions to the newly created folders as shown below.

```
[root@linux /]# chmod 777 /var/lib/cassandra  
[root@linux /]# chmod 777 /var/log/cassandra
```

Start Cassandra

To start Cassandra, open the terminal window, navigate to Cassandra home directory/home, where you unpacked Cassandra, and run the following command to start your Cassandra server.

```
$ cd $CASSANDRA_HOME  
$ ./bin/cassandra -f
```

Using the **-f** option tells Cassandra to stay in the foreground instead of running as a background process. If everything goes fine, you can see the Cassandra server starting.

Programming Environment

To set up Cassandra programmatically, download the following jar files:

- slf4j-api-1.7.5.jar
- cassandra-driver-core-2.0.2.jar
- guava-16.0.1.jar
- metrics-core-3.0.2.jar
- netty-3.9.0.Final.jar

Place them in a separate folder. For example, we are downloading these jars to a folder named **"Cassandra_jars"**.

Set the classpath for this folder in “**.bashrc**”file as shown below.

```
[hadoop@linux ~]$ gedit ~/.bashrc

//Set the following class path in the .bashrc file.

export CLASSPATH = $CLASSPATH:/home/hadoop/Cassandra_jars/*
```

Cassandra - Shell Commands

Cassandra provides documented shell commands in addition to CQL commands. Given below are the Cassandra documented shell commands.

Help

The HELP command displays a synopsis and a brief description of all cqlsh commands. Given below is the usage of help command.

```
cqlsh> help

Documented shell commands:
=====
CAPTURE COPY DESCRIBE EXPAND PAGING SOURCE
CONSISTENCY DESC EXIT HELP SHOW TRACING.

CQL help topics:
=====
ALTER      CREATE_TABLE_OPTIONS  SELECT
ALTER_ADD  CREATE_TABLE_TYPES      SELECT_COLUMNFAMILY
ALTER_ALTER CREATE_USER          SELECT_EXPR
ALTER_DROP DELETE                SELECT_LIMIT
ALTER_RENAME DELETE_COLUMNS      SELECT_TABLE
```

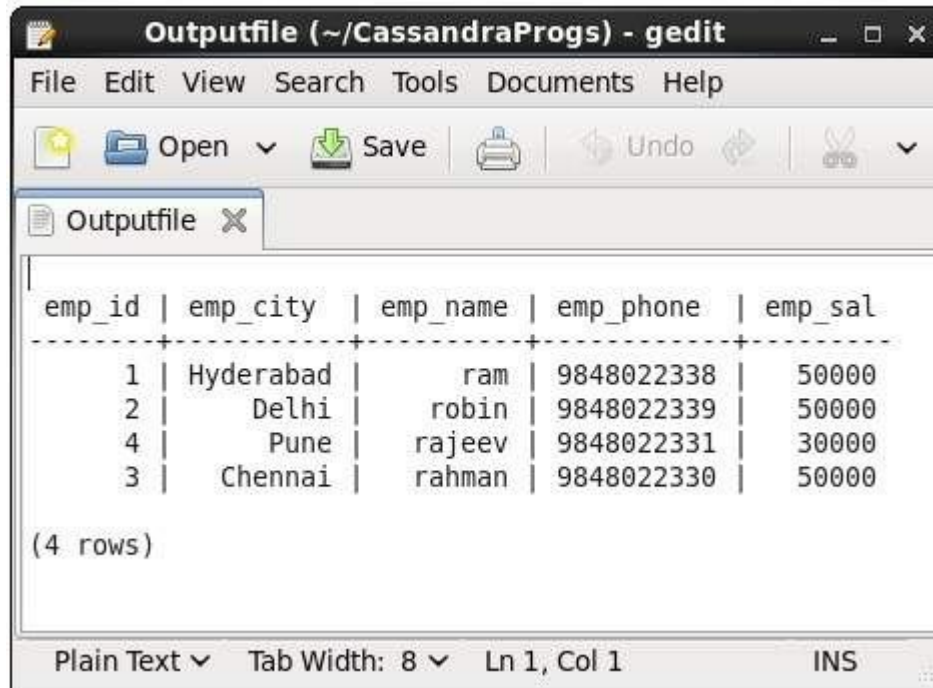
Capture

This command captures the output of a command and adds it to a file. For example, take a look at the following code that captures the output to a file named **Outputfile**.

```
cqlsh> CAPTURE '/home/hadoop/CassandraProgs/Outputfile'
```

When we type any command in the terminal, the output will be captured by the file given. Given below is the command used and the snapshot of the output file.

```
cqlsh:tutorialspoint> select * from emp;
```



emp_id	emp_city	emp_name	emp_phone	emp_sal
1	Hyderabad	ram	9848022338	50000
2	Delhi	robin	9848022339	50000
4	Pune	rajeev	9848022331	30000
3	Chennai	rahman	9848022330	50000

(4 rows)

You can turn capturing off using the following command.

```
cqlsh:tutorialspoint> capture off;
```

Consistency

This command shows the current consistency level, or sets a new consistency level.

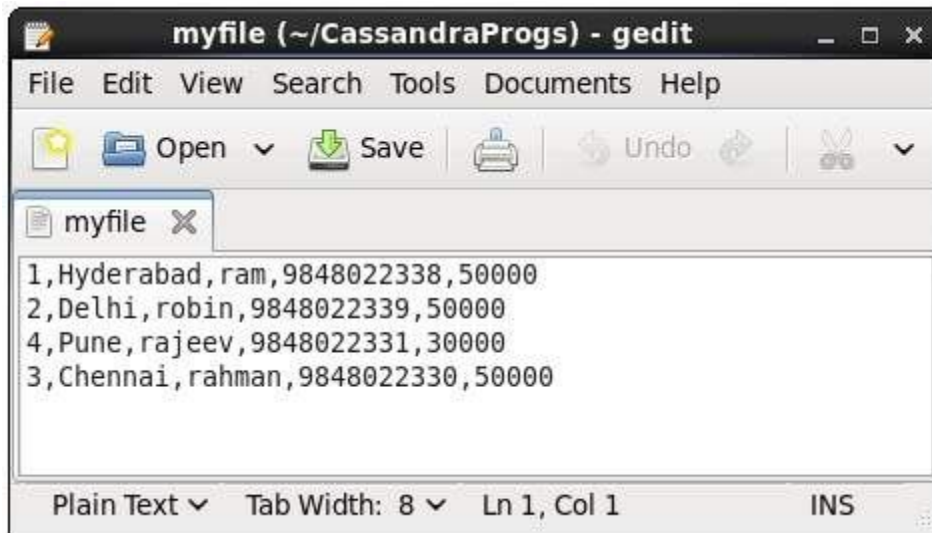
```
cqlsh:tutorialspoint> CONSISTENCY  
Current consistency level is 1.
```

Copy

This command copies data to and from Cassandra to a file. Given below is an example to copy the table named **emp** to the file **myfile**.

```
cqlsh:tutorialspoint> COPY emp (emp_id, emp_city, emp_name, emp_phone, emp_sal) TO  
'myfile';  
4 rows exported in 0.034 seconds.
```

If you open and verify the file given, you can find the copied data as shown below.



Describe

This command describes the current cluster of Cassandra and its objects. The variants of this command are explained below.

Describe cluster - This command provides information about the cluster.

```
cqlsh:tutorialspoint> describe cluster;
```

Cluster: Test Cluster

Partitioner: Murmur3Partitioner

Range ownership:

-658380912249644557 [127.0.0.1]

-2833890865268921414 [127.0.0.1]

-6792159006375935836 [127.0.0.1]

Describe Keyspaces - This command lists all the keyspaces in a cluster. Given below is the usage of this command.

```
cqlsh:tutorialspoint> describe keyspaces;
```

system_traces system tp tutorialspoint

Describe tables - This command lists all the tables in a keyspace. Given below is the usage of this command.

```
cqlsh:tutorialspoint> describe tables;
```


emp

Describe table - This command provides the description of a table. Given below is the usage of this command.

```
cqlsh:tutorialspoint> describe table emp;
```

```
CREATE TABLE tutorialspoint.emp (  
  emp_id int PRIMARY KEY,  
  emp_city text,  
  emp_name text,  
  emp_phone varint,  
  emp_sal varint  
) WITH bloom_filter_fp_chance = 0.01  
  AND caching = {'keys':"ALL", "rows_per_partition":"NONE"}  
  AND comment = ''  
  AND compaction = {'min_threshold': '4', 'class':  
  'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',  
  'max_threshold': '32'}  
  
  AND compression = {'sstable_compression':  
  'org.apache.cassandra.io.compress.LZ4Compressor'}  
  
  AND dclocal_read_repair_chance = 0.1  
  AND default_time_to_live = 0  
  AND gc_grace_seconds = 864000  
  AND max_index_interval = 2048  
  AND memtable_flush_period_in_ms = 0  
  AND min_index_interval = 128  
  AND read_repair_chance = 0.0  
  AND speculative_retry = '99.0PERCENTILE';  
CREATE INDEX emp_emp_sal_idx ON tutorialspoint.emp (emp_sal);
```

Describe Type

This command is used to describe a user-defined data type. Given below is the usage of this command.

```
cqlsh:tutorialspoint> describe type card_details;
```

```
CREATE TYPE tutorialspoint.card_details (  
  num int,  
  pin int,  
  name text,  
  cvv int,  
  phone set<int>,  
  mail text  
);
```

Describe Types

This command lists all the user-defined data types. Given below is the usage of this command. Assume there are two user-defined data types: **card** and **card_details**.

```
cqlsh:tutorialspoint> DESCRIBE TYPES;
```

```
card_details card
```

Expand

This command is used to expand the output. Before using this command, you have to turn the expand command on. Given below is the usage of this command.

```
cqlsh:tutorialspoint> expand on;  
cqlsh:tutorialspoint> select * from emp;
```

```
@ Row 1
```

```
-----+-----  
emp_id | 1  
emp_city | Hyderabad  
emp_name | ram  
emp_phone | 9848022338  
emp_sal | 50000
```

```
@ Row 2
```

```
-----+-----  
emp_id | 2  
emp_city | Delhi  
emp_name | robin  
emp_phone | 9848022339
```

```
emp_sal | 50000
```

```
@ Row 3
```

```
-----+-----
```

```
emp_id | 4
```

```
emp_city | Pune
```

```
emp_name | rajeev
```

```
emp_phone | 9848022331
```

```
emp_sal | 30000
```

```
@ Row 4
```

```
-----+-----
```

```
emp_id | 3
```

```
emp_city | Chennai
```

```
emp_name | rahman
```

```
emp_phone | 9848022330
```

```
emp_sal | 50000
```

```
(4 rows)
```

Note: You can turn the expand option off using the following command.

```
cqlsh:tutorialspoint> expand off;  
Disabled Expanded output.
```

Exit

This command is used to terminate the cql shell.

Show

This command displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions. Given below is the usage of this command.

```
cqlsh:tutorialspoint> show host;  
Connected to Test Cluster at 127.0.0.1:9042.
```

```
cqlsh:tutorialspoint> show version;  
[cqlsh 5.0.1 | Cassandra 2.1.2 | CQL spec 3.2.0 | Native protocol v3]
```

Source

Using this command, you can execute the commands in a file. Suppose our input file is as follows:



Then you can execute the file containing the commands as shown below.

```
cqlsh:tutorialspoint> source '/home/hadoop/CassandraProgs/inputfile';
```

```
emp_id | emp_city | emp_name | emp_phone | emp_sal
-----+-----+-----+-----+-----
  1 | Hyderabad | ram   | 9848022338 | 50000
  2 | Delhi    | robin | 9848022339 | 50000
  3 | Pune     | rajeev | 9848022331 | 30000
  4 | Chennai  | rahman | 9848022330 | 50000
(4 rows)
```

Cassandra - Create Table

Creating a Table using Cqlsh

You can create a table using the command **CREATE TABLE**. Given below is the syntax for creating a table.

Syntax

```
CREATE (TABLE | COLUMNFAMILY) <tablename>
('<column-definition>' , '<column-definition>')
```

(WITH <option> AND <option>)

Defining a Column

You can define a column as shown below.

```
column name1 data type,  
column name2 data type,
```

example:

```
age int,  
name text
```

Primary Key

The primary key is a column that is used to uniquely identify a row. Therefore, defining a primary key is mandatory while creating a table. A primary key is made of one or more columns of a table. You can define a primary key of a table as shown below.

```
CREATE TABLE tablename(  
    column1 name datatype PRIMARYKEY,  
    column2 name data type,  
    column3 name data type.  
)
```

or

```
CREATE TABLE tablename(  
    column1 name datatype PRIMARYKEY,  
    column2 name data type,  
    column3 name data type,  
    PRIMARY KEY (column1)  
)
```

Example

Given below is an example to create a table in Cassandra using cqlsh. Here we are:

- Using the keyspace tutorialspoint
- Creating a table named **emp**

It will have details such as employee name, id, city, salary, and phone number. Employee id is the primary key.

```
cqlsh> USE tutorialspoint;  
cqlsh:tutorialspoint>; CREATE TABLE emp(  
    emp_id int PRIMARY KEY,  
    emp_name text,  
    emp_city text,  
    emp_sal varint,  
    emp_phone varint  
);
```

Verification

The select statement will give you the schema. Verify the table using the select statement as shown below.

```
cqlsh:tutorialspoint> select * from emp;  
  
emp_id | emp_city | emp_name | emp_phone | emp_sal  
-----+-----+-----+-----+-----  
  
(0 rows)
```

Here you can observe the table created with the given columns. Since we have deleted the keyspace tutorialspoint, you will not find it in the keyspaces list.

Creating a Table using Java API

You can create a table using the `execute()` method of `Session` class. Follow the steps given below to create a table using Java API.

Step1: Create a Cluster Object

First of all, create an instance of the **Cluster.builder** class of **com.datastax.driver.core** package as shown below.

```
//Creating Cluster.Builder object  
Cluster.Builder builder1 = Cluster.builder();
```

Add a contact point (IP address of the node) using the **addContactPoint()** method of **Cluster.Builder** object. This method returns **Cluster.Builder**.

```
//Adding contact point to the Cluster.Builder object  
Cluster.Builder builder2 = build.addContactPoint( "127.0.0.1" );
```

Using the new builder object, create a cluster object. To do so, you have a method called **build()** in the **Cluster.Builder** class. The following code shows how to create a cluster object.

```
//Building a cluster  
Cluster cluster = builder.build();
```

You can build a cluster object using a single line of code as shown below.

```
Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
```

Step 2: Create a Session Object

Create an instance of Session object using the **connect()** method of **Cluster** class as shown below.

```
Session session = cluster.connect();
```

This method creates a new session and initializes it. If you already have a keyspace, you can set it to the existing one by passing the keyspace name in string format to this method as shown below.

```
Session session = cluster.connect(" Your keyspace name " );
```

Here we are using the keyspace named **tp**. Therefore, create the session object as shown below.

```
Session session = cluster.connect(" tp" );
```

Step 3: Execute Query

You can execute CQL queries using the **execute()** method of Session class. Pass the query either in string format or as a Statement class object to the **execute()** method. Whatever you pass to this method in string format will be executed on the cqlsh.

In the following example, we are creating a table named **emp**. You have to store the query in a string variable and pass it to the **execute()** method as shown below.

```
//Query  
String query = "CREATE TABLE emp(emp_id int PRIMARY KEY, "
```

```
+ "emp_name text, "  
+ "emp_city text, "  
+ "emp_sal varint, "  
+ "emp_phone varint );";  
session.execute(query);
```

Given below is the complete program to create and use a keyspace in Cassandra using Java API.

```
import com.datastax.driver.core.Cluster;  
import com.datastax.driver.core.Session;  
  
public class Create_Table {  
  
    public static void main(String args[]){  
  
        //Query  
        String query = "CREATE TABLE emp(emp_id int PRIMARY KEY, "  
            + "emp_name text, "  
            + "emp_city text, "  
            + "emp_sal varint, "  
            + "emp_phone varint );";  
  
        //Creating Cluster object  
        Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();  
  
        //Creating Session object  
        Session session = cluster.connect("tp");  
  
        //Executing the query  
        session.execute(query);  
    }  
}
```



```
System.out.println("Table created");  
}  
}
```

Save the above program with the class name followed by .java, browse to the location where it is saved. Compile and execute the program as shown below.

```
$javac Create_Table.java  
$java Create_Table
```

Under normal conditions, it should produce the following output:

```
Table created
```

Cassandra - Alter Table

Altering a Table using Cqlsh

You can alter a table using the command **ALTER TABLE**. Given below is the syntax for creating a table.

Syntax

```
ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>
```

Using ALTER command, you can perform the following operations:

- Add a column
- Drop a column
- Update the options of a table using **with** keyword

Adding a Column

Using ALTER command, you can add a column to a table. While adding columns, you have to take care that the column name is not conflicting with the existing column names and that the table is not defined with compact storage option. Given below is the syntax to add a column to a table.

```
ALTER TABLE table name  
ADD new column datatype;
```

Example

Given below is an example to add a column to an existing table. Here we are adding a column called **emp_email** of text datatype to the table named **emp**.

```
cqlsh:tutorialspoint> ALTER TABLE emp
... ADD emp_email text;
```

Verification

Use the SELECT statement to verify whether the column is added or not. Here you can observe the newly added column emp_email.

```
cqlsh:tutorialspoint> select * from emp;

emp_id | emp_city | emp_email | emp_name | emp_phone | emp_sal
-----+-----+-----+-----+-----+-----
```

Dropping a Column

Using ALTER command, you can delete a column from a table. Before dropping a column from a table, check that the table is not defined with compact storage option. Given below is the syntax to delete a column from a table using ALTER command.

```
ALTER table name
DROP column name;
```

Example

Given below is an example to drop a column from a table. Here we are deleting the column named **emp_email**.

```
cqlsh:tutorialspoint> ALTER TABLE emp DROP emp_email;
```

Verification

Verify whether the column is deleted using the **select** statement, as shown below.

```
cqlsh:tutorialspoint> select * from emp;

emp_id | emp_city | emp_name | emp_phone | emp_sal
-----+-----+-----+-----+-----
```

(0 rows)

Since **emp_email** column has been deleted, you cannot find it anymore.

Altering a Table using Java API

You can create a table using the `execute()` method of `Session` class. Follow the steps given below to alter a table using Java API.

Step1: Create a Cluster Object

First of all, create an instance of **Cluster.builder** class of **com.datastax.driver.core** package as shown below.

```
//Creating Cluster.Builder object  
Cluster.Builder builder1 = Cluster.builder();
```

Add a contact point (IP address of the node) using the **addContactPoint()** method of **Cluster.Builder** object. This method returns **Cluster.Builder**.

```
//Adding contact point to the Cluster.Builder object  
Cluster.Builder builder2 = builder1.addContactPoint( "127.0.0.1" );
```

Using the new builder object, create a cluster object. To do so, you have a method called **build()** in the **Cluster.Builder** class. The following code shows how to create a cluster object.

```
//Building a cluster  
Cluster cluster = builder2.build();
```

You can build a cluster object using a single line of code as shown below.

```
Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
```

Step 2: Create a Session Object

Create an instance of `Session` object using the `connect()` method of `Cluster` class as shown below.

```
Session session = cluster.connect();
```

This method creates a new session and initializes it. If you already have a keyspace, you can set it to the existing one by passing the `KeySpace` name in string format to this method as shown below.

```
Session session = cluster.connect(" Your keyspace name " );  
Session session = cluster.connect(" tp" );
```

Here we are using the KeySpace named tp. Therefore, create the session object as shown below.

Step 3: Execute Query

You can execute CQL queries using the `execute()` method of Session class. Pass the query either in string format or as a Statement class object to the `execute()` method. Whatever you pass to this method in string format will be executed on the **cqlsh**.

In the following example, we are adding a column to a table named **emp**. To do so, you have to store the query in a string variable and pass it to the `execute()` method as shown below.

```
//Query  
String query1 = "ALTER TABLE emp ADD emp_email text";  
session.execute(query);
```

Given below is the complete program to add a column to an existing table.

```
import com.datastax.driver.core.Cluster;  
import com.datastax.driver.core.Session;  
  
public class Add_column {  
  
    public static void main(String args[]){  
  
        //Query  
        String query = "ALTER TABLE emp ADD emp_email text";  
  
        //Creating Cluster object  
        Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();  
  
        //Creating Session object
```

```

Session session = cluster.connect("tp");

//Executing the query
session.execute(query);

System.out.println("Column added");
}
}

```

Save the above program with the class name followed by .java, browse to the location where it is saved. Compile and execute the program as shown below.

```

$javac Add_Column.java
$java Add_Column

```

Under normal conditions, it should produce the following output:

```

Column added

```

Deleting a Column

Given below is the complete program to delete a column from an existing table.

```

import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.Session;

public class Delete_Column {

    public static void main(String args[]){

        //Query
        String query = "ALTER TABLE emp DROP emp_email;";

        //Creating Cluster object
    }
}

```

```

Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();

//Creating Session object
Session session = cluster.connect("tp");

//executing the query
session.execute(query);

System.out.println("Column deleted");
}
}

```

Save the above program with the class name followed by .java, browse to the location where it is saved. Compile and execute the program as shown below.

```

$javac Delete_Column.java
$java Delete_Column

```

Under normal conditions, it should produce the following output:

```
Column deleted
```

Cassandra - Drop Table

Dropping a Table using Cqlsh

You can drop a table using the command **Drop Table**. Its syntax is as follows:

Syntax

```
DROP TABLE <tablename>
```

Example

The following code drops an existing table from a KeySpace.

```
cqlsh:tutorialspoint> DROP TABLE emp;
```

Verification

Use the Describe command to verify whether the table is deleted or not. Since the emp table has been deleted, you will not find it in the column families list.

```
cqlsh:tutorialspoint> DESCRIBE COLUMNFAMILIES;
```

```
employee
```

Deleting a Table using Java API

You can delete a table using the `execute()` method of Session class. Follow the steps given below to delete a table using Java API.

Step1: Create a Cluster Object

First of all, create an instance of **Cluster.builder** class of **com.datastax.driver.core** package as shown below:

```
//Creating Cluster.Builder object  
Cluster.Builder builder1 = Cluster.builder();
```

Add a contact point (IP address of the node) using **addContactPoint()** method of **Cluster.Builder** object. This method returns **Cluster.Builder**.

```
//Adding contact point to the Cluster.Builder object  
Cluster.Builder builder2 = builder1.addContactPoint( "127.0.0.1" );
```

Using the new builder object, create a cluster object. To do so, you have a method called **build()** in the **Cluster.Builder** class. The following code shows how to create a cluster object.

```
//Building a cluster  
Cluster cluster = builder2.build();
```

You can build a cluster object using a single line of code as shown below.

```
Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
```

Step 2: Create a Session Object

Create an instance of Session object using the `connect()` method of Cluster class as shown below.

```
Session session = cluster.connect();
```

This method creates a new session and initializes it. If you already have a keyspace, you can set it to the existing one by passing the KeySpace name in string format to this method as shown below.

```
Session session = cluster.connect("Your keyspace name");
```

Here we are using the keyspace named **tp**. Therefore, create the session object as shown below.

```
Session session = cluster.connect("tp");
```

Step 3: Execute Query

You can execute CQL queries using `execute()` method of Session class. Pass the query either in string format or as a Statement class object to the `execute()` method. Whatever you pass to this method in string format will be executed on the **cqlsh**.

In the following example, we are deleting a table named **emp**. You have to store the query in a string variable and pass it to the `execute()` method as shown below.

```
// Query  
  
String query = "DROP TABLE emp1;";  
session.execute(query);
```

Given below is the complete program to drop a table in Cassandra using Java API.

```
import com.datastax.driver.core.Cluster;  
import com.datastax.driver.core.Session;  
  
public class Drop_Table {  
  
    public static void main(String args[]){  
  
        //Query  
        String query = "DROP TABLE emp1;";  
        Cluster cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
```



```
//Creating Session object
Session session = cluster.connect("tp");

//Executing the query
session.execute(query);

System.out.println("Table dropped");
}
}
```

Save the above program with the class name followed by .java, browse to the location where it is saved. Compile and execute the program as shown below.

```
$javac Drop_Table.java
$java Drop_Table
```

Under normal conditions, it should produce the following output:

```
Table dropped
```

Conclusion:

Cassandra is capable of handling all of the big data challenges that might arise: massive scalability, an always on architecture, high performance, strong security, and ease of management.

Reference: http://www.tutorialspoint.com/cassandra/cassandra_quick_guide.htm

LAB 9: Designing database schemas and implement min 10 queries using DynamoDBkey Value based database.

Objective:

1. To learn NoSQL Databases (Open source) such as Hive/ Hbase/ Cassandra/DynamoDB.

Introduction:

Database Schema:- A database schema is a way to logically group objects such as tables, views, stored procedures etc. Think of a schema as a container of objects. You can assign a user login permissions to a single schema so that the user can only access the objects they are authorized to access.

DynamoDB Data Model:-

Tables, Items, and Attributes

In DynamoDB, a *table* is a collection of *items* and each item is a collection of *attributes*.

In a relational database, a table has a predefined schema such as the table name, primary key, list of its column names and their data types. All records stored in the table must have the same set of columns. In contrast, DynamoDB only requires that a table has a primary key, but does not require you to define all of the attribute names and data types in advance. Individual items in a DynamoDB table can have any number of attributes, although there is a limit of 400 KB on the item size. An item size is the sum of lengths of its attribute names and values (binary and UTF-8 lengths).

Each attribute in an item is a name-value pair. An attribute can be a scalar (single-valued), a JSON document, or a set. For example, consider storing a catalog of products in DynamoDB. You can create a table, Product Catalog, with the *Id* attribute as its primary key. The primary key uniquely identifies each item, so that no two products in the table can have the same *Id*.

Primary Key

When you create a table, in addition to the table name, you must specify the primary key of the table. The primary key uniquely identifies each item in the table, so that no two items can have the same key.

DynamoDB supports two different kinds of primary keys:

1]Partition Key – A simple primary key, composed of one attribute known as the partition key. DynamoDB uses the partition key's value as input to an internal hash function; the output from the hash function determines the partition where the item will be stored. No two items in a table can have the same partition key value.

Partition Key and Sort Key – A composite primary key, composed of two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*. DynamoDB uses the partition key value as input to an internal hash function; the output from the hash function determines the partition where the item will be stored. All items with the same partition key are stored together, in sorted order by sort key value. It is possible for two items to have the same partition key value, but those two items must have different sort key values.

Table Name	Primary Key Type	Partition Key Name	Sort Key Name
Forum (<u>Name</u> , ...)	Simple	Name	-
Thread (<u>ForumName</u> , <u>Subject</u> , ...)	Composite	ForumName	Subject
Reply (<u>Id</u> , <u>ReplyDateTime</u> , ...)	Composite	Id	ReplyDateTim

Secondary Indexes

DynamoDB supports two kinds of secondary indexes:

- **Global secondary index** – an index with a partition key and sort key that can be different from those on the table.
- **Local secondary index** – an index that has the same partition key as the table, but a different sort key.

DynamoDB Data Types

DynamoDB supports the following data types:

- **Scalar types** – Number, String, Binary, Boolean, and Null.
- **Document types** – List and Map.
- **Set types** – String Set, Number Set, and Binary Set.

For example, in the Product Catalog table, the *Id* is a Number type attribute and *Authors* is a String Set type attribute. Note that primary key attributes must be of type String, Number, or Binary.

Item Distribution

DynamoDB stores data in partitions. A *partition* is an allocation of storage for a table, backed by solid state drives (SSDs) and automatically replicated across three facilities within an AWS region. Partition management is handled entirely by DynamoDB—customers never need to manage partitions themselves. If your storage requirements exceed a partition's capacity, DynamoDB allocates additional partitions automatically.

When you create a table, the initial status of the table is *CREATING*. During this phase, DynamoDB allocates one partition for the table. You can begin writing and reading table data after the table status changes to *ACTIVE*.

As the amount of data in the table approaches the partition's maximum capacity, DynamoDB allocates another partition to the table, and then distributes the data items among the old partition and the new one. This activity occurs in the background, and is transparent to your applications. The more data you add to the table, the more partitions that DynamoDB will allocate—as many as necessary to store your table's data.

DynamoDB does not deallocate or coalesce partitions. If a table spans multiple partitions, and you delete most of the data (or all of it), the partitions will still be allocated to the table

Query

A *Query* operation uses the primary key of a table or a secondary index to directly access items from that table or index.

Use the *KeyConditionExpression* parameter to provide a specific value for the partition key. The *Query* operation will return all of the items from the table or index with that partition key value. You can optionally narrow the scope of the *Query* operation by specifying a sort key value and a comparison operator in *KeyConditionExpression*. You can use the *ScanIndexForward* parameter to get results in forward or reverse order, by sort key.

Queries that do not return results consume the minimum number of read capacity units for that type of read operation.

If the total number of items meeting the query criteria exceeds the result set size limit of 1 MB, the query stops and results are returned to the user with the *LastEvaluatedKey* element to continue the query in a subsequent operation. Unlike a *Scan* operation, a *Query* operation never returns both an empty result set and a *LastEvaluatedKey* value. *LastEvaluatedKey* is only provided if the results exceed 1 MB, or if you have used the *Limit* parameter.

You can query a table, a local secondary index, or a global secondary index. For a query on a table or on a local secondary index, you can set the *ConsistentRead* parameter to true and obtain a strongly consistent result. Global secondary indexes support eventually consistent reads only, so do not specify *ConsistentRead* when querying a global secondary index.

Input Code:

```
package com.mkyong.core;

import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;

/**
 * Java MongoDB : Query document
 *
 * @author mkyong
 */
```

```

*/
public class QueryApp {
    public static void insertDummyDocuments(DBCollection collection) {
        List<DBObject> list = new ArrayList<DBObject>();
        Calendar cal = Calendar.getInstance();
        for (int i = 1; i <= 5; i++) {
            BasicDBObject data = new BasicDBObject();
            data.append("number", i);
            data.append("name", "mkyong-" + i);
            // data.append("date", cal.getTime());
            // +1 day
            cal.add(Calendar.DATE, 1);
            list.add(data);
        }
        collection.insert(list);
    }

    public static void main(String[] args) {
        try { Mongo mongo = new Mongo("localhost", 27017);
            DB db = mongo.getDB("yourdb");
            // get a single collection
            DBCollection collection = db.getCollection("dummyColl");
            insertDummyDocuments(collection);
            System.out.println("1. Find first matched document");
            DBObject dbObject = collection.findOne();
            System.out.println(dbObject);
            System.out.println("\n1. Find all matched documents");
            DBCursor cursor = collection.find();
            while (cursor.hasNext())
            {System.out.println(cursor.next());

```

```

    }
    System.out.println("\n1. Get 'name' field only");
    BasicDBObject allQuery = new BasicDBObject();
    BasicDBObject fields = new BasicDBObject();
    fields.put("name", 1);
    DBCursor cursor2 = collection.find(allQuery, fields);
    while (cursor2.hasNext())
    {
        System.out.println(cursor2.next());
    }

    System.out.println("\n2. Find where number = 5");
    BasicDBObject whereQuery = new BasicDBObject();
    whereQuery.put("number", 5);
    DBCursor cursor3 = collection.find(whereQuery);
    while (cursor3.hasNext())
    {
        System.out.println(cursor3.next());
    }

    System.out.println("\n2. Find where number in 2,4 and 5");
    BasicDBObject inQuery = new BasicDBObject();
    List<Integer> list = new ArrayList<Integer>();
    list.add(2);
    list.add(4);
    list.add(5);
    inQuery.put("number", new BasicDBObject("$in", list));
    DBCursor cursor4 = collection.find(inQuery);
    while (cursor4.hasNext())
    {
        System.out.println(cursor4.next());
    }

```

```

    }
    System.out.println("\n2. Find where 5 > number > 2");
    BasicDBObject gtQuery = new BasicDBObject();
    gtQuery.put("number", new BasicDBObject("$gt", 2).append("$lt", 5));
    DBCursor cursor5 = collection.find(gtQuery);
    while (cursor5.hasNext())
    {
    System.out.println(cursor5.next());
    }
    System.out.println("\n2. Find where number != 4");
    BasicDBObject neQuery = new BasicDBObject();
    neQuery.put("number", new BasicDBObject("$ne", 4));
    DBCursor cursor6 = collection.find(neQuery);
    while (cursor6.hasNext())
    {
    System.out.println(cursor6.next());
    }
    System.out.println("\n3. Find when number = 2 and name = 'mkyong-2' example");
    BasicDBObject andQuery = new BasicDBObject();

    List<BasicDBObject> obj = new ArrayList<BasicDBObject>();
    obj.add(new BasicDBObject("number", 2));
    obj.add(new BasicDBObject("name", "mkyong-2"));
    andQuery.put("$and", obj);
    System.out.println(andQuery.toString());
    DBCursor cursor7 = collection.find(andQuery);
    while (cursor7.hasNext())
    {
    System.out.println(cursor7.next());

```



```

    }
    System.out.println("\n4. Find where name = 'Mky.*-[1-3]', case sensitive example");
    BasicDBObject regexQuery = new BasicDBObject();
    regexQuery.put("name", new BasicDBObject("$regex", "Mky.*-[1-3]"));
    .append("$options", "i");
    System.out.println(regexQuery.toString());
    DBCursor cursor8 = collection.find(regexQuery);
    while (cursor8.hasNext())
    {
    System.out.println(cursor8.next());
    }
    collection.drop();
    System.out.println("Done");
    }
    catch (UnknownHostException e)
    {
    e.printStackTrace();
    }
    catch (MongoException e)
    {
    e.printStackTrace();
    }
    }
    }

```

References:

[1] Internet : https://en.wikipedia.org/wiki/Database_schema

[2]Internet:<http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html>

—

LAB 10: Implementation of Web Page Ranking Algorithm

Objective:

1. To learn and understand web database language, XML, JDOQL.

Introduction:

PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google: PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites. It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known. Page Rank is a topic much discussed by Search Engine Optimization (SEO) experts. At the heart of PageRank is a mathematical formula that seems scary to look at but is actually fairly simple to understand. Despite this many people seem to get it wrong! In particular "Chris Ridings of www.searchenginesystems.net" has written a paper entitled "PageRank Explained: Everything you've always wanted to know about PageRank", pointed to by many people, that contains a fundamental mistake early on in the explanation! Unfortunately this means some of the recommendations in the paper are not quite accurate. By showing code to correctly calculate real PageRank I hope to achieve several things in this response:

1. Clearly explain how PageRank is calculated.
2. Go through every example in Chris' paper, and add some more of my own, showing the correct PageRank for each diagram. By showing the code used to calculate each diagram I've opened myself up to peer review - mostly in an effort to make sure the examples are correct, but also because the code can help explain the PageRank calculations.
3. Describe some principles and observations on website design based on these correctly calculated examples.

Any good web designer should take the time to fully understand how PageRank really works - if you don't then your site's layout could be seriously hurting your Google listings!

How is PageRank Used?

PageRank is one of the methods Google uses to determine a page's relevance or importance. It is only one part of the story when it comes to the Google listing, but the other aspects are discussed elsewhere (and are ever changing) and PageRank is interesting enough to deserve a paper of its own. PageRank is also displayed on the toolbar of your browser if you've installed the Google toolbar (<http://toolbar.google.com/>). But the Toolbar PageRank only goes from 0 – 10 and seems to be something like a logarithmic scale:

Toolbar (log base 10)	PageRank Real PageRank
0	0 - 10
1	100 - 1,000
2	1,000 - 10,000
3	10,000 - 100,000
4	and so on...

So what is PageRank?

In short PageRank is a “vote”, by all the other pages on the Web, about how important a page is. A link to a page counts as a vote of support. If there's no link there's no support (but it's an abstention from voting rather than a vote against the page).

Quoting from the original Google paper, PageRank is defined like this:

We assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:

$$PR(A) = (1-d) + d (PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$$

Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages' PageRanks will be one.

PageRank or PR(A) can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web.

but that's not too helpful so let's break it down into sections.

1. **PR(Tn)** - Each page has a notion of its own self-importance. That's “PR(T1)” for the first page in the web all the way up to “PR(Tn)” for the last page

2. **$C(T_n)$** - Each page spreads its vote out evenly amongst all of its outgoing links. The count, or number, of outgoing links for page 1 is " $C(T_1)$ ", " $C(T_n)$ " for page n, and so on for all pages.
3. **$PR(T_n)/C(T_n)$** - so if our page (page A) has a backlink from page "n" the share of the vote page A will get is " $PR(T_n)/C(T_n)$ "
4. **$d(\dots$** - All these fractions of votes are added together but, to stop the other pages having too much influence, this total vote is "damped down" by multiplying it by 0.85 (the factor "d")
5. **$(1 - d)$** - The $(1 - d)$ bit at the beginning is a bit of probability math magic so the "*sum of all web pages' PageRanks will be one*": it adds in the bit lost by the **$d(\dots$** . It also means that if a page has no links to it (no backlinks) even then it will still get a small PR of 0.15 (i.e. $1 - 0.85$). (Aside: the Google paper says "the sum of all pages" but they mean the "the normalised sum" – otherwise known as "the average" to you and me.

How is PageRank Calculated?

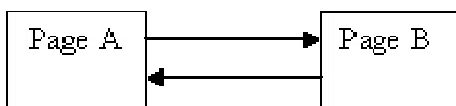
This is where it gets tricky. The PR of each page depends on the PR of the pages pointing to it. But we won't know what PR those pages have until the pages pointing to **them** have their PR calculated and so on... And when you consider that page links can form circles it seems impossible to do this calculation!

But actually it's not that bad. Remember this bit of the Google paper:

PageRank or $PR(A)$ can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web.

What that means to us is that we can just go ahead and calculate a page's PR without knowing the final value of the PR of the other pages. That seems strange but, basically, each time we run the calculation we're getting a closer estimate of the final value. So all we need to do is remember the each value we calculate and repeat the calculations lots of times until the numbers stop changing much.

Lets take the simplest example network: two pages, each pointing to the other:



Each page has one outgoing link (the outgoing count is 1, i.e. $C(A) = 1$ and $C(B) = 1$).

Input Code:

```
<?php
error_reporting(E_ERROR);
$links = array(
    1 => array(5),
    2 => array(4, 7, 8),
    3 => array(1, 3, 4, 7, 9),
    4 => array(1, 2, 4, 8),
    5 => array(1, 6, 7, 9),
    6 => array(1, 5, 8),
    8 => array(3, 4),
    9 => array(1, 4, 6, 8)
);

function calculatePageRank($linkGraph, $dampingFactor = 0.15) {
    $pageRank = array();
    $tempRank = array();
    $nodeCount = count($linkGraph);

    // initialise the PR as 1/n
    foreach($linkGraph as $node => $outbound) {
        $pageRank[$node] = 1/$nodeCount;
        $tempRank[$node] = 0;
    }

    $change = 1;
    $i = 0;
    while($change > 0.00005 && $i < 100) {
        $change = 0;
        $i++;

        // distribute the PR of each page
        foreach($linkGraph as $node => $outbound) {
            $outboundCount = count($outbound);
            foreach($outbound as $link) {
                $tempRank[$link] += $pageRank[$node] / $outboundCount;
            }
        }

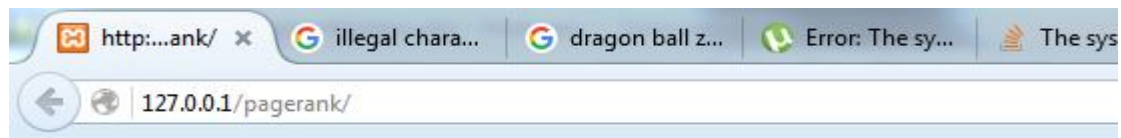
        $total = 0;
        // calculate the new PR using the damping factor
```

```

foreach($linkGraph as $node => $outbound) {
    $tempRank[$node] = ($dampingFactor / $nodeCount)
        + (1-$dampingFactor) * $tempRank[$node];
    $change += abs($pageRank[$node] - $tempRank[$node]);
    $pageRank[$node] = $tempRank[$node];
    $tempRank[$node] = 0;
    $total += $pageRank[$node];
}

// Normalise the page ranks so it's all a proportion 0-1
foreach($pageRank as $node => $score) {
    $pageRank[$node] /= $total;
}
}
$i=1;
foreach ($pageRank as $r)
{
    echo "Page Rank of $i :   $r <br/>";
    $i++;
}
}
echo "<h1>Page Rank Algorithm Demonstration</h1><br/>";
calculatePageRank($links);
?>

```



Page Rank Algorithm Demonstration

```

Page Rank of 1 : 0.17084756816354
Page Rank of 2 : 0.060138597093041
Page Rank of 3 : 0.095270626028832
Page Rank of 4 : 0.17308244947008
Page Rank of 5 : 0.20422266588834
Page Rank of 6 : 0.086831014221959
Page Rank of 7 : 0.12476184607085
Page Rank of 8 : 0.084845233063364

```

Output:

Conclusion:

PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents, such as theWorld Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element E is referred to as the *PageRank of E* and denoted by $PR(E)$. Other factors like Author Rank can contribute to the importance of an entity.

References:

[1] Internet: <https://en.wikipedia.org/wiki/PageRank>

[2] Internet: <http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm>

LAB 11: Implementation of Machine Learning Algorithm for Classification Task in Big Data Analytics.

Objective:

1. To learn and understand Database Modeling, Architectures.

Introduction:

k -means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K means clustering aims to [partition](#) n observations into k clusters in which each observation belongs to the cluster with the nearest [mean](#), serving as a [prototype](#) of the cluster. This results in a partitioning of the data space into [Voronoi cells](#). The problem is computationally difficult ([NP-hard](#)); however, there are efficient [heuristic algorithms](#) that are commonly employed and converge quickly to a [local optimum](#). These are usually similar to the expectation maximization [algorithm](#) for [mixtures of Gaussian distributions](#) via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, k -means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes. The algorithm has a loose relationship to the [\$k\$ -nearest neighbor classifier](#), a popular [machine learning](#) technique for classification that is often confused with k -means because of the k in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by k -means to classify new data into the existing clusters. This is known as [nearest centroid classifier](#) or Rocchio algorithm. Given a set of observations (x_1, x_2, \dots, x_n) , where each observation is a d -dimensional real vector, k -means clustering aims to partition the n observations into k ($\leq n$) sets $S = \{S_1, S_2, \dots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center). In other words, its objective is to find:

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

The Lloyd's algorithm, mostly known as k-means algorithm, is used to solve the k-means clustering problem and works as follows. First, decide the number of clusters k . Then further

1. Initialize the center of the clusters	$\mu_i = \text{some value}, i=1, \dots, k$
2. Attribute the closest cluster to each data point	$c_i = \{j: d(x_j, \mu_i) \leq d(x_j, \mu_l), l \neq i, j=1, \dots, n\}$
3. Set the position of each cluster to the mean of all data points belonging to that cluster	$\mu_i = \frac{1}{ c_i } \sum_{j \in c_i} x_j, \forall i$
4. Repeat steps 2-3 until convergence	
Notation	$ c $ = number of elements in c

steps in above table. The algorithm eventually converges to a point, although it is not necessarily the minimum of the sum of squares. That is because the problem is non-convex and the algorithm is just a heuristic, converging to a local minimum. The algorithm stops when the assignments do not change from one iteration to the next.

Deciding the number of clusters

The number of clusters should match the data. An incorrect choice of the number of clusters will invalidate the whole process. An empirical way to find the best number of clusters is to try K-means clustering with different number of clusters and measure the resulting sum of squares.

The most curious can look at this paper for a benchmarking of 30 procedures for estimating the number of clusters.

Initializing the position of the clusters

It is really up to you! Here are some common methods:

- **Forgy:** set the positions of the k clusters to k observations chosen randomly from the dataset.
- **Random partition:** assign a cluster randomly to each observation and compute means as in step 3.

Since the algorithm stops in a local minimum, the initial position of the clusters is very important.

Advantages

- 1) Fast, robust and easier to understand.
- 2) Relatively efficient: $O(tknd)$, where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, $k, t, d \ll n$.
- 3) Gives best result when data set are distinct or well separated from each other.

Disadvantages

- 1) The learning algorithm requires apriori specification of the number of cluster centers.
- 2) The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.
- 3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
- 4) Euclidean distance measures can unequally weight underlying factors.
- 5) The learning algorithm provides the local optima of the squared error function.
- 6) Randomly choosing of the cluster center cannot lead us to the fruitful result. Pl. refer Fig.
- 7) Applicable only when mean is defined i.e. fails for categorical data.
- 8) Unable to handle noisy data and outliers.
- 9) Algorithm fails for non-linear data set.

Input Code:

```
import sys
import math
import random
import subprocess
def main():
    # How many points are in our dataset?
    num_points = 10
    # For each of those points how many dimensions do they have?
    dimensions = 2
    # Bounds for the values of those points in each dimension
    lower = 0
    upper = 200
    # The K in k-means. How many clusters do we assume exist?
    num_clusters = 3
```

```

# When do we say the optimization has 'converged' and stop updating clusters
opt_cutoff = 0.5
# Generate some points
points = [makeRandomPoint(dimensions, lower, upper) for i in xrange(num_points)]
# Cluster those data!
clusters = kmeans(points, num_clusters, opt_cutoff)
# Print our clusters
for i,c in enumerate(clusters):
    for p in c.points:
        print " Cluster: ", i, "\t Point :", p
class Point:
    """
    An point in n dimensional space
    """
    def __init__(self, coords):
        """
        coords - A list of values, one per dimension
        """
        self.coords = coords
        self.n = len(coords)
    def __repr__(self):
        return str(self.coords)
class Cluster:
    """
    A set of points and their centroid
    """
    def __init__(self, points):
        """
        points - A list of point objects
        """
        if len(points) == 0: raise Exception("ILLEGAL: empty cluster")
        # The points that belong to this cluster
        self.points = points
        # The dimensionality of the points in this cluster
        self.n = points[0].n
        # Assert that all points are of the same dimensionality
        for p in points:
            if p.n != self.n: raise Exception("ILLEGAL: wrong dimensions")

```

```

    # Set up the initial centroid (this is usually based off one point)
    self.centroid = self.calculateCentroid()
def __repr__(self):
    """
    String representation of this object
    """
    return str(self.points)
def update(self, points):
    """
    Returns the distance between the previous centroid and the new after
    recalculating and storing the new centroid.
    """
    old_centroid = self.centroid
    self.points = points
    self.centroid = self.calculateCentroid()
    shift = getDistance(old_centroid, self.centroid)
    return shift
def calculateCentroid(self):
    """
    Finds a virtual center point for a group of n-dimensional points
    """
    numPoints = len(self.points)
    # Get a list of all coordinates in this cluster
    coords = [p.coords for p in self.points]
    # Reformat that so all x's are together, all y's etc.
    unzipped = zip(*coords)
    # Calculate the mean for each dimension
    centroid_coords = [math.fsum(dList)/numPoints for dList in unzipped]
    return Point(centroid_coords)
def kmeans(points, k, cutoff):
    # Pick out k random points to use as our initial centroids
    initial = random.sample(points, k)
    # Create k clusters using those centroids
    clusters = [Cluster([p]) for p in initial]
    # Loop through the dataset until the clusters stabilize
    loopCounter = 0
    while True:
        # Create a list of lists to hold the points in each cluster

```

```

lists = [ [] for c in clusters]
clusterCount = len(clusters)
# Start counting loops
loopCounter += 1
# For every point in the dataset ...
for p in points:
    # Get the distance between that point and the centroid of the first
    # cluster.
    smallest_distance = getDistance(p, clusters[0].centroid)
    # Set the cluster this point belongs to
    clusterIndex = 0
    # For the remainder of the clusters ...
    for i in range(clusterCount - 1):
        # calculate the distance of that point to each other cluster's
        # centroid.
        distance = getDistance(p, clusters[i+1].centroid)
        # If it's closer to that cluster's centroid update what we
        # think the smallest distance is, and set the point to belong
        # to that cluster
        if distance < smallest_distance:
            smallest_distance = distance
            clusterIndex = i+1
    lists[clusterIndex].append(p)
# Set our biggest_shift to zero for this iteration
biggest_shift = 0.0
# As many times as there are clusters ...
for i in range(clusterCount):
    # Calculate how far the centroid moved in this iteration
    shift = clusters[i].update(lists[i])
    # Keep track of the largest move from all cluster centroid updates
    biggest_shift = max(biggest_shift, shift)
# If the centroids have stopped moving much, say we're done!
if biggest_shift < cutoff:
    print "Converged after %s iterations" % loopCounter
    break
return clusters

def getDistance(a, b):
    """ Euclidean distance between two n-dimensional points.

```

Note: This can be very slow and does not scale well

'''

```
if a.n != b.n:
```

```
    raise Exception("ILLEGAL: non comparable points")
```

```
ret = reduce(lambda x,y: x + pow((a.coords[y]-b.coords[y]), 2),range(a.n),0.0)
```

```
return math.sqrt(ret)
```

```
def makeRandomPoint(n, lower, upper):
```

```
    '''Returns a Point object with n dimensions and values between lower and upper in each  
    of those dimensions'''
```

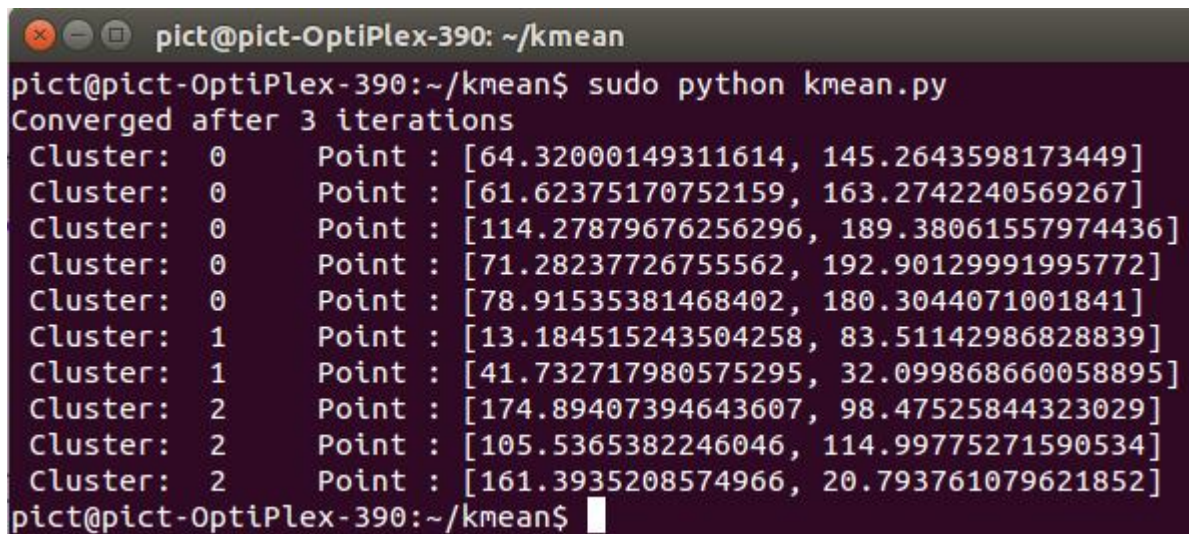
```
    p = Point([random.uniform(lower, upper) for i in range(n)])
```

```
    return p
```

```
if __name__ == "__main__":
```

```
    main()
```

Output:

A terminal window titled 'pict@pict-OptiPlex-390: ~/kmean' showing the execution of 'python kmean.py'. The output indicates convergence after 3 iterations and lists 9 points assigned to 3 clusters. The first 5 points are assigned to Cluster 0, the next 2 to Cluster 1, and the last 2 to Cluster 2. Each point is represented as a list of 10 coordinates.

```
pict@pict-OptiPlex-390: ~/kmean
pict@pict-OptiPlex-390:~/kmean$ sudo python kmean.py
Converged after 3 iterations
Cluster: 0      Point : [64.32000149311614, 145.2643598173449]
Cluster: 0      Point : [61.62375170752159, 163.2742240569267]
Cluster: 0      Point : [114.27879676256296, 189.38061557974436]
Cluster: 0      Point : [71.28237726755562, 192.90129991995772]
Cluster: 0      Point : [78.91535381468402, 180.3044071001841]
Cluster: 1      Point : [13.184515243504258, 83.51142986828839]
Cluster: 1      Point : [41.732717980575295, 32.099868660058895]
Cluster: 2      Point : [174.89407394643607, 98.47525844323029]
Cluster: 2      Point : [105.5365382246046, 114.99775271590534]
Cluster: 2      Point : [161.3935208574966, 20.793761079621852]
pict@pict-OptiPlex-390:~/kmean$
```

Conclusion:

K-means (MacQueen,1967) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori.

References:

[1] Internet: https://en.wikipedia.org/wiki/K-means_clustering

[2] Internet: <http://www.onmyphd.com/?p=k-means.clustering&ckattempt=1>

LAB 12: Implementation of Social Web Mining Application Using Python, Google Developer Console, Youtube API.

Objective:

1. To learn and understand Database Modeling, Architectures.
2. To learn and understand Advanced Database Programming Frameworks.

Introduction:

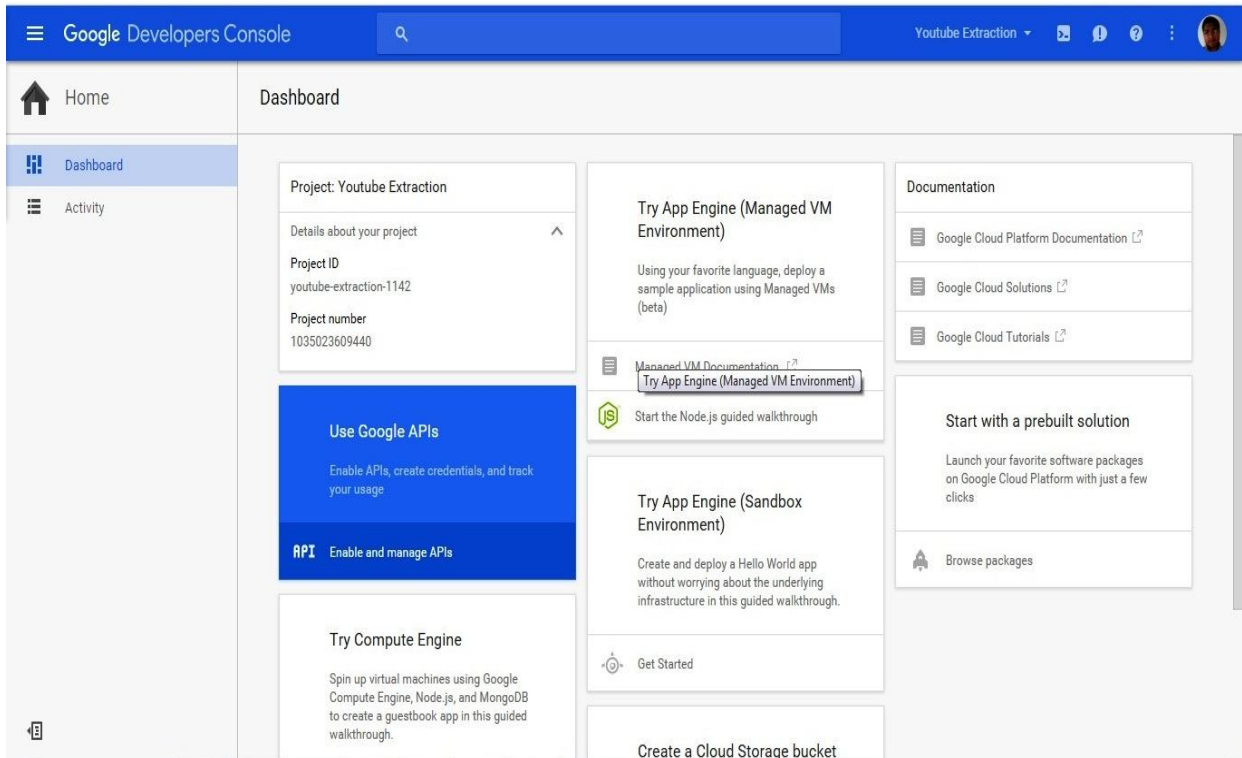
Social Media Mining is the process of representing, analyzing, and extracting actionable patterns from social media data. Social Media Mining, introduces basic concepts and principal algorithms suitable for investigating massive social media data; it discusses theories and methodologies from different disciplines such as computer science, data mining, machine learning, social network analysis, network science, sociology, ethnography, statistics, optimization, and mathematics. It encompasses the tools to formally represent, measure, model, and mine meaningful patterns from large-scale social media data.

Google Developer Console

Google Developers (previously Google Code) is [Google's](#) site for [software development](#) tools, [application programming interfaces](#)(APIs), and technical resources. The site contains documentation on using Google developer tools and APIs—including discussion groups and blogs for developers using Google's developer products. There are APIs offered for almost all of Google's popular consumer products, like [Google Maps](#), [YouTube](#), [Google Apps](#), and others. The site also features a variety of developer products and tools built specifically for developers. [Google App Engine](#) is a hosting service for web apps. Project Hosting gives users version control for [open source](#) code. [Google Web Toolkit](#) (GWT) allows developers to create [Ajax](#) applications in the [Java programming language](#). The site contains reference information for community based developer products that Google is involved with like [Android](#) from the [Open Handset Alliance](#) and [Open Social](#) from the Open Social Foundation

YouTube API

The [YouTube](#) Application Programming Interface, or the YouTube API, allows developers to access video statistics and YouTube channels' data via two types of calls, [REST](#) and [XML-RPC](#). Google describe the YouTube API Resources as 'APIs and Tools that let you bring the YouTube experience to your webpage, application or device.' ^[1] This is one of the [Google Developers](#)



Input Code:

```
from apiclient.discovery import build #pip install google-api-python-client
from apiclient.errors import HttpError #pip install google-api-python-client
from oauth2client.tools import argparser #pip install oauth2client
import pandas as pd #pip install pandas
#import matplotlib as
```

```
DEVELOPER_KEY = "AlzaSyBgKRPbxETHlXCruUNzaKyhSokzFlhDVT0"
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"
```

```
argparser.add_argument("--q", help="Search term", default="Movie Trailer")
#change the default to the search term you want to search
argparser.add_argument("--max-results", help="Max results", default=25)
#default number of results which are returned. It can vary from 0 - 100
args = argparser.parse_args()
options = args
youtube = build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION,
               developerKey=DEVELOPER_KEY)
# Call the search.list method to retrieve results matching the specified
# query term.
search_response = youtube.search().list(
    q=options.q,
```

```

type="video",
part="id,snippet",
maxResults=options.max_results
).execute()

```

```

videos={}
for search_result in search_response.get("items", []):
    if search_result["id"]["kind"] == "youtube#video":
        #videos.append("%s" % (search_result["id"]["videoid"]))
        videos[search_result["id"]["videoid"]] = search_result["snippet"]["title"]
        #print "Videos:\n", "\n".join(videos), "\n"
        s = ','.join(videos.keys())

```

```

videos_list_response = youtube.videos().list(id=s,part='id,statistics').execute()
res = []
for i in videos_list_response['items']:
    temp_res = dict(v_id = i['id'], v_title = videos[i['id']])
    temp_res.update(i['statistics'])
    res.append(temp_res)

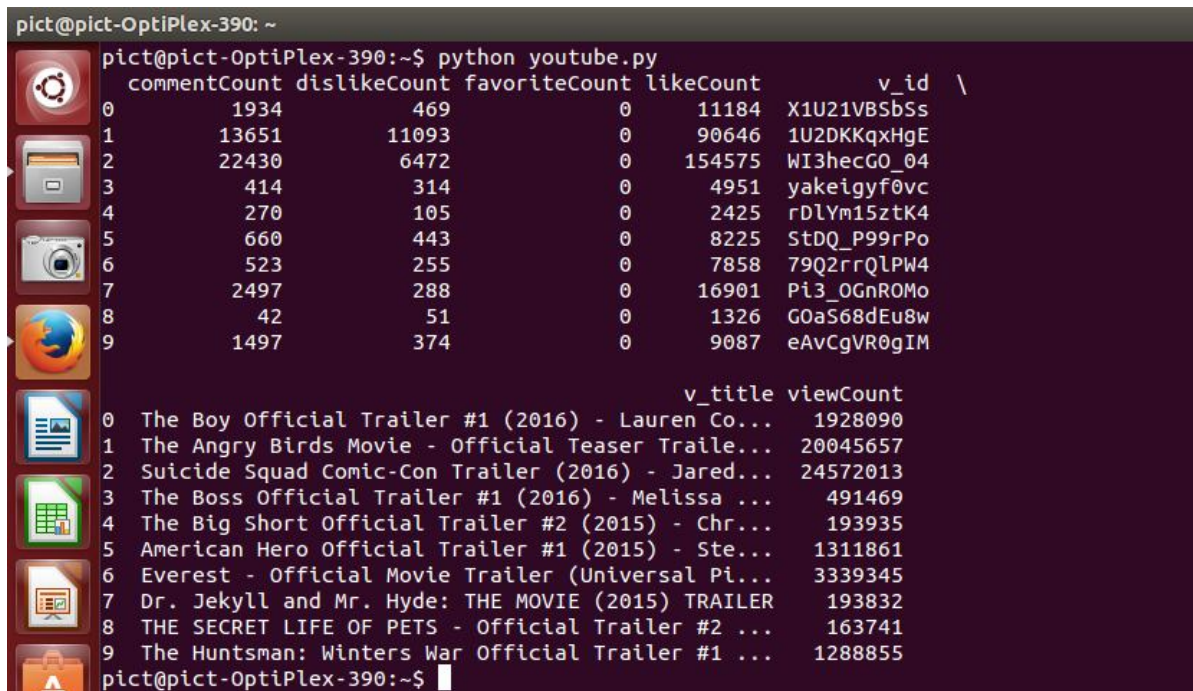
```

```

data=pd.DataFrame.from_dict(res)
print data.head(10)

```

Output:



```

pict@pict-OptiPlex-390: ~
pict@pict-OptiPlex-390:~$ python youtube.py
commentCount  dislikeCount  favoriteCount  likeCount      v_id \
0           1934           469              0       11184  X1U21VBSbSs
1          13651          11093              0       90646  1U2DKKqxHgE
2          22430           6472              0      154575  WI3hecGO_04
3           414           314              0       4951  yakeigyf0vc
4           270           105              0       2425  rDLYm15ztK4
5           660           443              0       8225  StDQ_P99rPo
6           523           255              0       7858  79Q2rrQLPW4
7          2497           288              0      16901  Pi3_OGnROMo
8            42            51              0       1326  GOaS68dEu8w
9          1497           374              0       9087  eAvCgVR0gIM

v_title  viewCount
0  The Boy Official Trailer #1 (2016) - Lauren Co...  1928090
1  The Angry Birds Movie - Official Teaser Traile...  20045657
2  Suicide Squad Comic-Con Trailer (2016) - Jared...  24572013
3  The Boss Official Trailer #1 (2016) - Melissa ...   491469
4  The Big Short Official Trailer #2 (2015) - Chr...  193935
5  American Hero Official Trailer #1 (2015) - Ste...  1311861
6  Everest - Official Movie Trailer (Universal Pi...  3339345
7  Dr. Jekyll and Mr. Hyde: THE MOVIE (2015) TRAILER  193832
8  THE SECRET LIFE OF PETS - Official Trailer #2 ...   163741
9  The Huntsman: Winters War Official Trailer #1 ...  1288855
pict@pict-OptiPlex-390:~$

```

References:

[1] Internet : https://en.wikipedia.org/wiki/Google_Developers

[2] Internet: https://en.wikipedia.org/wiki/YouTube_API

—