db.books.save({ _id:5,Category:"Web Mining", Bookname:" Learning R ", Author:" Richard Cotton",qty:5, price:850,rol:10,pages:120});

Step 2: Confirm the presence of the above documents in the "books" collection.

Step 3: Write map and reduce functions to split the books into the following two categories:
(a) Big books
(b) Small books

Books which have more than 300 pages should be in the big book category. Books which have less than 300 pages should be in the small book category.

Step 4: Count the number of books in each category.

Step 5: Store the output as follows as documents in a new collection, called, "Book_Result".

Book Category	Count of the Books
(a) Big books	2
(b) Small books	3

ASSIGNMENT 2

Objective: To practice import, export, and aggregation in MongoDB.

Step 1: Pick any public dataset from the site www.kdnuggets.com. Convert it into CSV format. Make sure that you have at least two numeric columns.

Step 2: Use MongoImport to import data from the CSV format file into MongoDB collection, "MongoDBHandsOn" in test database.

Step 3: Identify a grouping column.

Step 4: Compute the sum of the values in the first numeric column.

Step 5: Compute the average of the values in the second numeric column.

ASSIGNMENT 3

Objective: To copy the JSON documents from one MongoDB collection to another MongoDB collection.

ASSIGNMENT 4

Objective: Write the insert method to store the following document in MongoDB.

Name: "Stephen More"

Address:

{ "City" : "Bangalore",
"Street" : "Electronics City",
"Affiliation" : "XYZ Ltd"
}

Hobbies: Chess, Lawn Tennis, Base ball

CHAPTER 7

Introduction to Cassandra

BRIEF CONTENTS

- What's in Store?
- Apache Cassandra – An Introduction
- Features of Cassandra
  - Peer-to-Peer Network
  - Gossip and Failure Detection
  - Partitioner
  - Replication Factor
  - Anti-Entropy and Read Repair
  - Writes in Cassandra
  - Hinted Handoffs
  - Tunable Consistency: Read Consistency and Write Consistency
- CQL Data Types
- CQLSH
- Keyspaces
- CRUD Operations
  - Insert
  - Update
  - Delete
  - Select
- Collections
  - Set Collection
  - List Collection
  - Map Collection
- Using a Counter
- Time To Live (TTL)
- Alter Commands
  - Alter Table to Change the Data Type of a Column
  - Alter Table to Delete a Column
  - Drop a Table
  - Drop a Database
- Import and Export
  - Export to CSV
  - Import from CSV
  - Import from STDIN
  - Export to STDOUT
- Querying System Tables
- Practice Examples

"Data is a precious thing and will last longer than the systems themselves."

– Tim Berners-Lee, inventor of the World Wide Web.

WHAT'S IN STORE?

This chapter will cover another NoSQL database called "Cassandra". We will explore the features of Cassandra that has made it so immensely popular. The chapter will cover the basic CRUD (Create, Read, Update, and Delete) operations using cqlsh.

Please attempt the Test Me exercises given at the end of the chapter to practice, learn, and comprehend Cassandra effectively.

7.1 APACHE CASSANDRA – AN INTRODUCTION

We shall start this chapter with few points that a reader should know about Cassandra.

1. Apache Cassandra was born at Facebook. After Facebook open sourced the code in 2008, Cassandra became an Apache Incubator project in 2009 and subsequently became a top-level Apache project in 2010.
2. It is built on Amazon's dynamo and Google's BigTable.
3. Cassandra does NOT compromise on availability. Since it does not have a master-slave architecture, there is no question of single point of failure. This proves beneficial for business critical applications that need to be up and running always and cannot afford to go down ever.
4. It is highly scalable (it scales out), high performance distributed database. It distributes and manages gigantic amount of data across commodity servers.

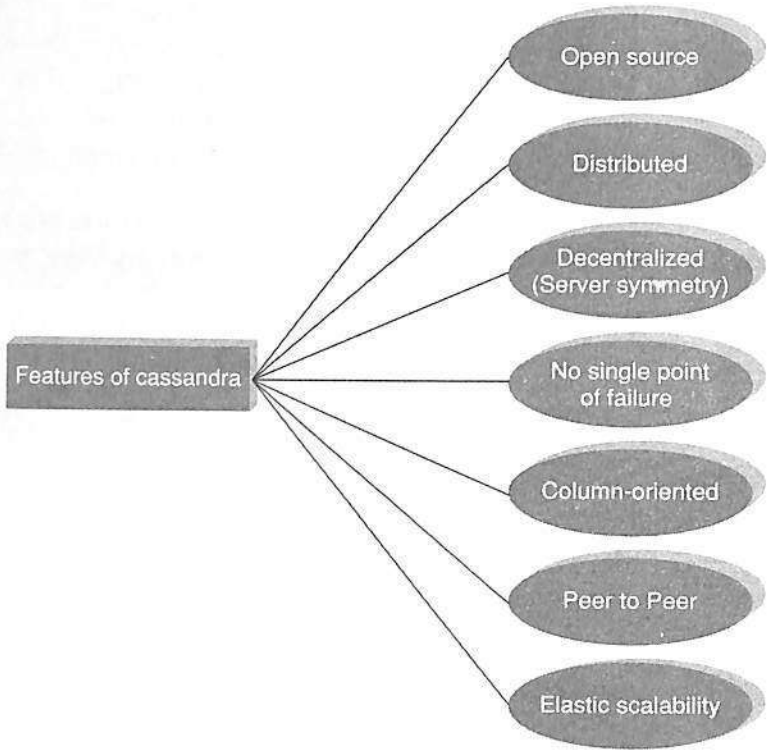


Figure 7.1 Features of Cassandra.

5. It is a column-oriented database designed to support peer-to-peer symmetric nodes instead of the master-slave architecture.
6. It has adherence to the Availability and Partition Tolerance properties of CAP theorem. It takes care of consistency using BASE (Basically Available Soft State Eventual Consistency) approach.

Refer Figure 7.1. Few companies that have successfully deployed Cassandra and have benefitted immensely from it are as follows:

1. Twitter
2. Netflix
3. Cisco
4. Adobe
5. eBay
6. Rackspace

7.2 FEATURES OF CASSANDRA

7.2.1 Peer-to-Peer Network

As with any other NoSQL database, Cassandra is designed to distribute and manage large data loads across multiple nodes in a cluster constituted of commodity hardware. Cassandra does NOT have a master-slave architecture which means that it does NOT have single point of failure. A node in Cassandra is structurally identical to any other node. Refer Figure 7.2. In case a node fails or is taken offline, it definitely impacts the throughput. However, it is a case of graceful degradation where everything does not come crashing at any given instant owing to a node failure. One can still go about business as usual. It tides over the problem of failure by employing a peer-to-peer distributed system across homogeneous nodes. It ensures that data is distributed across all nodes in the cluster. Each node exchanges information across the cluster every second.

Let us look at how a Cassandra node writes. Each write is written to the commit log sequentially. A write is taken to be successful only if it is written to the commit log. Data is then indexed and pushed to an in-memory structure called "Memtable". When the in-memory data structure, "the Memtable", is full, the contents are flushed to "SSTable" (Sorted String) data file on the disk. The SSTable is immutable and is

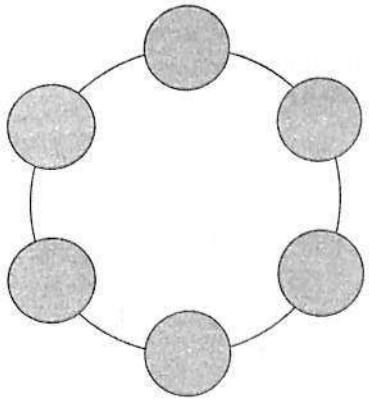


Figure 7.2 Sample Cassandra cluster.

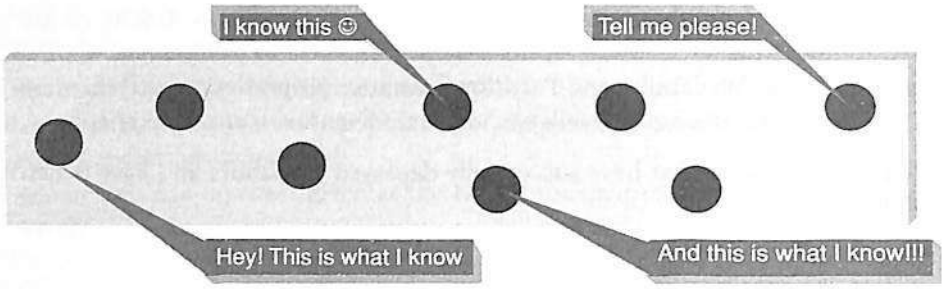


Figure 7.3 Gossip protocol.

append-only. It is stored on disk sequentially and is maintained for each Cassandra table. The partitioning and replication of all writes are performed automatically across the cluster.

7.2.2 Gossip and Failure Detection

Gossip protocol is used for intra-ring communication. It is a peer-to-peer communication protocol which eases the discovery and sharing of location and state information with other nodes in the cluster. Refer Figure 7.3. Although there are quite a few subtleties involved, but at its core it's a simple and robust system. A node only has to send out the communication to a subset of other nodes. For repairing unread data, Cassandra uses what's called an anti-entropy version of the gossip protocol.

7.2.3 Partitioner

A partitioner takes a call on how to distribute data on the various nodes in a cluster. It also determines the node on which to place the very first copy of the data. Basically a partitioner is a hash function to compute the token of the partition key. The partition key helps to identify a row uniquely.

7.2.4 Replication Factor

The replication factor determines the number of copies of data (replicas) that will be stored across nodes in a cluster. If one wishes to store only one copy of each row on one node, they should set the replication factor to one. However, if the need is for two copies of each row of data on two different nodes, one should go with a replication factor of two. The replication factor should ideally be more than one and not more than the number of nodes in the cluster. A replication strategy is employed to determine which nodes to place the data on. Two replication strategies are available:

1. SimpleStrategy.
2. NetworkToplogyStrategy.

The preferred one is NetworkTopologyStrategy as it is simple and supports easy expansion to multiple data centers, should there be a need.

7.2.5 Anti-Entropy and Read Repair

A cluster is made up of several nodes. Since the cluster is constituted of commodity hardware, it is prone to failure. In order to achieve fault tolerance, a given piece of data is replicated on one or more nodes. A client

can connect to any node in the cluster to read data. How many nodes will be read before responding to the client is based on the consistency level specified by the client. If the client-specified consistency is not met, the read operation blocks. There is a possibility that few of the nodes may respond with an out-of-date value. In such a case, Cassandra will initiate a read repair operation to bring the replicas with stale values up to date.

For repairing unread data, Cassandra uses an anti-entropy version of the gossip protocol. Anti-entropy implies comparing all the replicas of each piece of data and updating each replica to the newest version. The read repair operation is performed either before or after returning the value to the client as per the specified consistency level.

7.2.6 Writes in Cassandra

Let us look at behind the scene activities. Here is a client that initiates a write request. Where does his write get written to? It is first written to the commit log. A write is taken as successful only if it is written to the commit log. The next step is to push the write to a memory resident data structure called Memtable. A threshold value is defined in the Memtable. When the number of objects stored in the Memtable reaches a threshold, the contents of Memtable are flushed to the disk in a file called SSTable (Sorted String Table). Flushing is a non-blocking operation. It is possible to have multiple Memtables for a single column family. One out of them is current and the rest are waiting to be flushed.

7.2.7 Hinted Handoffs

The first question that arises is: Why Cassandra is all for availability? It works on the philosophy that it will always be available for writes.

Assume that we have a cluster of three nodes – Node A, Node B, and Node C. Node C is down for some reason. Refer Figure 7.4. We are maintaining a replication factor of 2 which implies that two copies of each row will be stored on two different nodes. The client makes a write request to Node A. Node A is the coordinator and serves as a proxy between the client and the nodes on which the replica is to be placed. The client

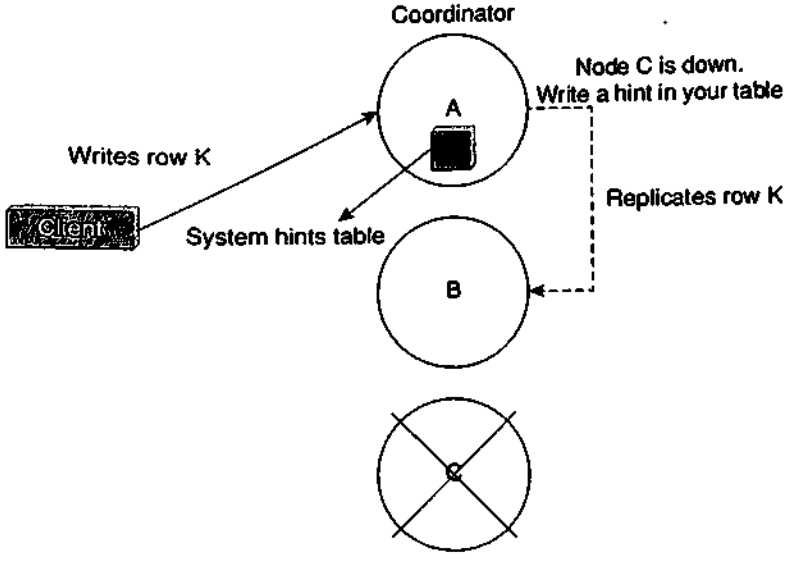


Figure 7.4 Depiction of hinted handoffs.

writes Row K to Node A. Node A then writes Row K to Node B and stores a hint for Node C. The hint will have the following information:

1. Location of the node on which the replica is to be placed.
2. Version metadata.
3. The actual data.

When Node C recovers and is back to the functional self, Node A reacts to the hint by forwarding the data to Node C.

7.2.8 Tunable Consistency

One of the features of Cassandra that has made it immensely popular is its ability to utilize tunable consistency. The database systems can go for either strong consistency or eventual consistency. Cassandra can cash in on either flavor of consistency depending on the requirements. In a distributed system, we work with several servers in the system. Few of these servers are in one data center and others in other data centers. Let us take a look at what it means by strong consistency and eventual consistency.

1. **Strong consistency:** If we work with strong consistency, it implies that each update propagates to all locations where that piece of data resides. Let us assume a single data center setup. Strong consistency will ensure that all of the servers that should have a copy of the data, will have it, before the client is acknowledged with a success. If we are wondering whether it will impact performance, yes it will. It will cost a few extra milliseconds to write to all servers.
2. **Eventual consistency:** If we work with eventual consistency, it implies that the client is acknowledged with a success as soon as a part of the cluster acknowledges the write. When should one go for eventual consistency? The choice is fairly obvious... when application performance matters the most. Example: A single server acknowledges the write and then begins propagating the data to other servers.

7.2.8.1 Read Consistency

Let us understand what the read consistency level means. It means how many replicas must respond before sending out the result to the client application. There are several read consistency levels as mentioned in Table 7.1.

7.2.8.2 Write Consistency

Let us understand what the write consistency level means. It means on how many replicas write must succeed before sending out an acknowledgement to the client application. There are several write consistency levels as mentioned in Table 7.2.

Table 7.1 Read consistency levels in Cassandra

ONE	Returns a response from the closest node (replica) holding the data.
QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data.
LOCAL_ QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data in the same data center as the coordinator node.
EACH_QUORUM	Returns a result from a quorum of servers with the most recent timestamp in all data centers.
ALL	This provides the highest level of consistency of all levels and the lowest level of availability of all levels. It responds to a read request from a client after all the replica nodes have responded.

Table 7.2 Write consistency levels in Cassandra

ALL	This is the highest level of consistency of all levels as it necessitates that a write must be written to the commit log and Memtable on all replica nodes in the cluster.
EACH_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in *all* data centers.
QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes.
LOCAL_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in the same data center as the coordinator node. This is to avoid latency of inter-data center communication.
ONE	A write must be written to the commit log and Memtable of at least one replica node.
TWO	A write must be written to the commit log and Memtable of at least two replica nodes.
THREE	A write must be written to the commit log and Memtable of at least three replica nodes.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local data center.

7.3 CQL DATA TYPES

Refer Table 7.3 for built-in data types for columns in CQL.

Table 7.3 Built-in data types in Cassandra

Int	32 bit signed integer
Bigint	64 bit signed long
Double	64-bit IEEE-754 floating point
Float	32-bit IEEE-754 floating point
Boolean	True or false
Blob	Arbitrary bytes, expressed in hexadecimal
Counter	Distributed counter value
Decimal	Variable – precision integer
List	A collection of one or more ordered elements
Map	A JSON style array of elements
Set	A collection of one or more elements
Timestamp	Date plus time
Varchar	UTF 8 encoded string
Varint	Arbitrary-precision integers
Text	UTF 8 encoded string

7.4 CQLSH

7.4.1 Logging into cqlsh

The below screenshot depicts the cqlsh command prompt after logging in, using cqlsh succeeds.

```
C:\windows\system32\cmd.exe - Python cqlsh

d:\apache-cassandra-2.0.0\apache-cassandra-2.0.0\apache-cassandra-2.0.0\bin>Python cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.0.0 | Cassandra 2.0.0 | CQL spec 3.1.0 | Thrift protocol 19.37.0]
Use HELP for help.
cqlsh>
```

The upcoming sections have been designed as follows:

Objective:	What is it that we are trying to achieve here?
Input (optional):	What is the input that has been given to us to act upon?
Act:	The actual statement /command to accomplish the task at hand.
Outcome:	The result/output as a consequence of executing the statement.

Objective: To get help with CQL.

Act:

```
Help
```

Outcome:

```
C:\windows\system32\cmd.exe - Python cqlsh

d:\apache-cassandra-2.0.0\apache-cassandra-2.0.0\apache-cassandra-2.0.0\bin>Python cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.0.0 | Cassandra 2.0.0 | CQL spec 3.1.0 | Thrift protocol 19.37.0]
Use HELP for help.
cqlsh> help

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

CQL help topics:
================
ALTER                          CREATE_TABLE_OPTIONS  REVOKE
ALTER_ADD                      CREATE_TABLE_TYPES    SELECT
ALTER_ALTER                    CREATE_USER           SELECT_COLUMNFAMILY
ALTER_DROP                     DELETE                SELECT_EXPR
ALTER_RENAME                   DELETE_COLUMNS        SELECT_LIMIT
ALTER_USER                     DELETE_USING          SELECT_TABLE
ALTER_WITH                     DELETE_WHERE          SELECT_WHERE
APPLY                          DROP                  TEXT_OUTPUT
ASCII_OUTPUT                   DROP_COLUMNFAMILY     TIMESTAMP_INPUT
BEGIN                          DROP_INDEX            TIMESTAMP_OUTPUT
BLOB_INPUT                     DROP_KEYSPACE         TRUNCATE
BOOLEAN_INPUT                  DROP_TABLE            TYPES
CREATE                         DROP_USER             UPDATE
CREATE_COLUMNFAMILY            GRANT                 UPDATE_COUNTERS
CREATE_COLUMNFAMILY_OPTIONS    INSERT                UPDATE_SET
CREATE_COLUMNFAMILY_TYPES      LIST                  UPDATE_USING
CREATE_INDEX                   LIST_PERMISSIONS      UPDATE_WHERE
CREATE_KEYSPACE                LIST_USERS            USE
CREATE_TABLE                   PERMISSIONS           UUID_INPUT

cqlsh>
```

7.5 KEYSPACES

What is a keyspace? A keyspace is a container to hold application data. It is comparable to a relational database. It is used to group column families together. Typically, a cluster has one keyspace per application. Replication is controlled on a per keyspace basis. Therefore, data that has different replication requirements should reside on different keyspaces.

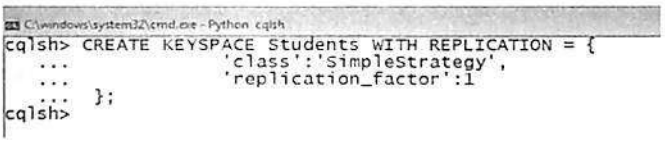
When one creates a keyspace, it is required to specify a strategy class. There are two choices available with us. Either we can specify a "SimpleStrategy" or a "NetworkTopologyStrategy" class. While using Cassandra for evaluation purpose, go with "SimpleStrategy" class and for production usage, work with the "NetworkTopologyStrategy" class.

Objective: To create a keyspace by the name "Students".

Act:

```
CREATE KEYSPACE Students WITH REPLICATION = {
        'class':'SimpleStrategy',
        'replication_factor':1
};
```

Outcome:



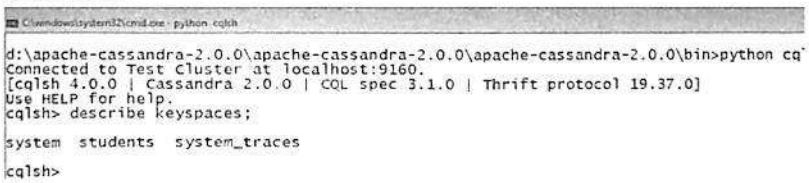
The replication factor stated above in the syntax for creating keyspace is related to the number of copies of keyspace data that is housed in a cluster.

Objective: To describe all the existing keyspaces.

Act:

```
DESCRIBE KEYSPACES;
```

Outcome:



Objective: To get more details on the existing keyspaces such as keyspace name, durable writes, strategy class, strategy options, etc.

Act:

```
SELECT *
        FROM system.schema_keyspaces;
```

Outcome:

```
cqlsh> SELECT * FROM system.schema_keyspaces;

 keyspace_name | durable_writes | strategy_class                                | strategy_options
---------------+----------------+-----------------------------------------------+----------------------------
      demo_con |           True | org.apache.cassandra.locator.SimpleStrategy | {"replication_factor":"3"}
        system |           True |  org.apache.cassandra.locator.LocalStrategy |                         {}
 system_traces |           True | org.apache.cassandra.locator.SimpleStrategy | {"replication_factor":"1"}
      students |           True | org.apache.cassandra.locator.SimpleStrategy | {"replication_factor":"1"}

(4 rows)
```

Note: Cassandra converted the Students keyspace to lowercase as quotation marks were not used.

Objective: To use the keyspace "Students", use the following command:

Use keyspace_name

Use connects the client session to the specified keyspace.

Act:

```
USE Students;
```

Outcome:

```
C:\windows\system32\cmd.exe - python cqlsh
cqlsh> use Students;
cqlsh:students>
```

Objective: To create a column family or table by the name "student_info".

Act:

```
CREATE TABLE Student_Info (
  RollNo int PRIMARY KEY,
  StudName text,
  DateofJoining timestamp,
  LastExamPercent double
  );
```

Outcome:

```
cqlsh> use Students;
cqlsh:students> CREATE TABLE Student_Info (
           ...   RollNo int PRIMARY Key,
           ...   StudName text,
           ...   DateofJoining timestamp,
           ...   LastExamPercent double
           ...   );
```

The table "student_info" gets created in the keyspace "students".

Note: Tables can have either a single or compound primary key. Always ensure that there is exactly one primary key definition. The primary key, however, can be simple (consisting of a single attribute) or composite (comprising two or more attributes).

Explanation about the composite PRIMARY KEY:

Primary key (column_name1, *column_name2, column_name3 ...*)
Primary key ((column_name4, *column_name5*), column_name6, *column_name7 ...*)

In the above syntax,
column_name1 is the partition key
column_name2 and column_name3 are the clustering columns.
column_name4 and column_name5 are the partitioning keys
column_name6 and column_name7 are the clustering columns.

The partition key is used to distribute the data in the table across various nodes that constitute the cluster. The clustering columns are used to store data in ascending order on the disk.

Objective: To lookup the names of all tables in the current keyspace, or in all the keyspaces if there is no current keyspace.

Act:

```
DESCRIBE TABLES;
```

Outcome:

```
cqlsh:students> describe tables;

student_info

cqlsh:students>
```

Objective: To describe the table "student_info" use the below command.

Act:

```
DESCRIBE TABLE student_info;
```

Note: The output is a list of CQL commands with the help of which the table "student_info" can be recreated.

Outcome:

```
C:\windows\system32\cmd.exe - python cqlsh
cqlsh:students> describe table student_info;

CREATE TABLE student_info (
  rollno int,
  dateofjoining timestamp,
  lastexampercent double,
  studname text,
  PRIMARY KEY (rollno)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

7.6 CRUD (CREATE, READ, UPDATE, AND DELETE) OPERATIONS

Objective: To insert data into the column family "student_info".

An insert writes one or more columns to a record in Cassandra table atomically. An insert statement does not return an output. One is not required to place values in all the columns; however, it is mandatory to specify all the columns that make up the primary key. The columns that are missing do not occupy any space on disk.

Internally insert and update operations are equal. However, insert does not support counters but update does. Counters will be discussed later in the chapter.

Act:

```
BEGIN BATCH
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (1,'Michael Storm','2012-03-29', 69.6)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (2,'Stephen Fox','2013-02-27', 72.5)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (3,'David Flemming','2014-04-12', 81.7)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (4,'Ian String','2012-05-11', 73.4)
APPLY BATCH;
```

Outcome:

```
cqlsh:students> BEGIN BATCH
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ... VALUES (1,'Michael Storm','2012-03-29',69.6)
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ... VALUES (2,'Stephen Fox','2013-02-27',72.5)
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ...  VALUES (3,'David Flemming','2014-04-12',81.7)
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ...  VALUES (4,'Ian String','2012-05-11',73.4)
           ... APPLY BATCH;
cqlsh:students>
```

Objective: To view the data from the table "student_info".

Act:

```
SELECT *
    FROM student_info;
```

The above select statement retrieves data from the "student_info" table.

Outcome:

```
cqlsh:students> select * from student_info;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+----------------
      1 | 2012-03-29 00:00:00India Standard Time  |            69.6 |  Michael Storm
      2 | 2013-02-27 00:00:00India Standard Time  |            72.5 |    Stephen Fox
      4 | 2012-05-11 00:00:00India Standard Time  |            73.4 |       Ian String
      3 | 2014-04-12 00:00:00India Standard Time  |            81.7 | David Flemming

(4 rows)
cqlsh:students>
```

Objective: To view only those records where the RollNo column either has a value 1 or 2 or 3.

Act:

```
SELECT *
    FROM student_info
        WHERE RollNo IN(1,2,3);
```

Note: For the above statement to execute successfully, ensure that the following criteria are satisfied:

1. Either the partition key definition includes the column that is used in the where clause i.e. search criteria.
2. OR the column being used in the where clause, that is, search criteria, has an index defined on it using the CREATE INDEX statement.

Outcome:

```
cqlsh:students> Select * from student_info where RollNo IN(1,2,3);

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+-----------------
      1 | 2012-03-29 00:00:00India Standard Time |            69.6 |   Michael Storm
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 |     Stephen Fox
      3 | 2014-04-12 00:00:00India Standard Time |            81.7 | David Flemming

(3 rows)

cqlsh:students>
```

Let us try running a query with "studname" in the where clause. Since "studname" is neither the primary key column nor a column in the primary key definition and also does not have an index defined on it, such a query will lead to error.

We set the stage to resolve the error by creating an index on the "studname" column of the "student_info" table and then subsequently executing the query.

To create an index on the "studname" column of the "student_info" column family use

```
CREATE INDEX ON student_info(studname)
```

To execute the query using the index defined on "studname" column use

```
SELECT *
    FROM student_info
        WHERE studname='Stephen Fox' ;
```

Outcome:

```
cqlsh:students> create index on student_info(studname);
cqlsh:students> select * from student_info where studname='Stephen Fox' ;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 | Stephen Fox

(1 rows)

cqlsh:students>
```

Objective: Let us create another index on the "LastExamPercent" column of the "student_info" column family.

Act:

```
CREATE INDEX ON student_info(LastExamPercent);
```

Outcome:

```
cqlsh:students> create index on student_info(Last
cqlsh:students> select * from student_info where LastExamPercent = 81.7;

 rollno | dateofjoining                          | lastexampercent | studname
--------+----------------------------------------+-----------------+-----------------
      3 | 2014-04-12 00:00:00India Standard Time |            81.7 | David Fleming

(1 rows)
```

Objective: To specify the number of rows returned in the output using limit.

Act:

```
SELECT rollno, hobbies, language, lastexampercent
      FROM student_info LIMIT 2;
```

Outcome:

```
cqlsh:students> select rollno, hobbies, language, lastexampercent from student_info limit 2;

 rollno | hobbies                                | language           | lastexampercent
--------+----------------------------------------+--------------------+-----------------
      1 |                ['Chess, Table Tennis'] | ['Hindi, English'] |            69.6
      4 | ['Lawn Tennis, Table Tennis, Golf'] | ['Hindi, English'] |            73.4

(2 rows)
```

Objective: To use column alias for the column "language" in the "student_info" table. We would like the column heading to be "knows language".

Act:

```
SELECT rollno, language AS "knows language"
      FROM student_info;
```

Outcome:

```
cqlsh:students> select rollno, language as "knows language" from student_info;

 rollno | knows language
--------+-----------------------
      1 |       ['Hindi, English']
      4 |       ['Hindi, English']
      3 | ['Hindi,English,French']

(3 rows)
```

Objective: To update the value held in the "StudName" column of the "student_info" column family to "David Sheen" for the record where the RollNo column has value = 2.

Note: An update updates one or more column values for a given row to the Cassandra table. It does not return anything.

Act:

UPDATE student_info SET StudName = 'David Sheen' WHERE RollNo = 2;

Outcome:

```
cqlsh:students> UPDATE student_info  SET StudName = 'David Sheen'  WHERE RollNo = 2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 | David Sheen

(1 rows)

cqlsh:students>
```

Objective: Let us try updating the value of a primary key column.

Act:

UPDATE student_info SET rollno=6 WHERE rollno=3;

Outcome:

```
cqlsh:students> update student_info set rollno=6 where rollno=3;
Bad Request: PRIMARY KEY part rollno found in SET part
cqlsh:students>
```

Note: It does not allow update to a primary key column.

Objective: Updating more than one column of a row of Cassandra table.

Act:

Step 1: Before the update

```
cqlsh:students> select rollno, studname, lastexampercent from student_info where rollno=3;

 rollno | studname        | lastexampercent
--------+-----------------+-----------------
      3 | David Flemming  |            81.7

(1 rows)
```

Step 2: Applying the update

```
cqlsh:students> select rollno, studname, lastexampercent from student_info where rollno=3;

 rollno | studname | lastexampercent
--------+----------+-----------------
      3 |  Samaira |              85

(1 rows)
```

Step 3: After the update

```
cqlsh:students> DELETE LastExamPercent FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            null | David Sheen

(1 rows)

cqlsh:students>
```

Objective: To delete the column "LastExamPercent" from the "student_info" table for the record where the RollNo = 2.

Note: Delete statement removes one or more columns from one or more rows of a Cassandra table or removes entire rows if no columns are specified.

Act:

```
DELETE LastExamPercent FROM student_info WHERE RollNo=2;
```

Outcome:

```
cqlsh:students> DELETE LastExamPercent FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                          | lastexampercent | studname
--------+----------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            null | David Sheen

(1 rows)

cqlsh:students>
```

Objective: To delete a row (where RollNo = 2) from the table "student_info".

Act:

```
DELETE FROM student_info WHERE RollNo=2;
```

Outcome:

```
cqlsh:students> DELETE  FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno=2;

(0 rows)

cqlsh:students>
```

Objective: To create a table "project _details" with primary key as (project_id, project_name).

Act:

```
CREATE TABLE project_details (
        project_id int,
        project_name text,
        stud_name text,
        rating double,
        duration int,
        PRIMARY KEY (project_id, project_name));
```

Outcome:

```
cqlsh:students> CREATE TABLE project_details (
            ...   project_id int,
            ...   project_name text,
            ...   stud_name text,
            ...   rating double,
            ...   duration int,
            ...   PRIMARY KEY (project_id, project_name));
cqlsh:students>
```

Objective: To insert data into the column family "project_details".

Act:

```
BEGIN BATCH
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (1,'MS data migration','David Sheen',3.5,720)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (1,'MS Data Warehouse','David Sheen',3.9,1440)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (2,'SAP Reporting','Stephen Fox',4.2,3000)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (2,'SAP BI DW','Stephen Fox',4,9000)
APPLY BATCH;
```

Outcome:

```
cqlsh:students> BEGIN BATCH
            ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
            ... VALUES (1,'MS data Migration','David Sheen',3.5,720)
            ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
            ... VALUES (1,'MS Data Warehouse','David Sheen',3.9,1440)
            ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
            ... VALUES (2,'SAP Reporting','Stephen Fox',4.2,3000)
            ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
            ... VALUES (2,'SAP BI DW','Stephen Fox',4,9000)
            ... APPLY BATCH;
```

Objective: To view all the rows of the "project_details" table.

Act:

```
SELECT *
      FROM project_details;
```

Outcome:

```
cqlsh:students> select * from project_details;

 project_id | project_name       | duration | rating | stud_name
------------+--------------------+----------+--------+-------------
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen
          1 | MS data Migration |      720 |    3.5 | David Sheen
          2 |        SAP BI DW |     9000 |      4 | Stephen Fox
          2 |     SAP Reporting |     3000 |    4.2 | Stephen Fox

(4 rows)
```

Objective: To view row/record from the "project_details" table wherein the project_id=1.

Act:

```
SELECT *
    FROM project_details
        WHERE project_id=1;
```

Outcome:

```
cqlsh:students> Select * from project_details where project_id=1;

 project_id | project_name      | duration | rating | stud_name
------------+-------------------+----------+--------+-------------
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen
          1 | MS data Migration |      720 |    3.5 | David Sheen

(2 rows)
```

Objective: To use "allow filtering" with the Select statement.

Note: When one attempts a potentially expensive query that might involve searching a range of rows, a prompt such as the one shown below appears:

> Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING.

Act:

```
SELECT *
    FROM project_details
        WHERE project_name='MS Data Warehouse' ALLOW FILTERING;
```

Outcome:

```
cqlsh:students> Select * from project_details where project_name='MS Data Warehouse' allow filtering;

 project_id | project_name      | duration | rating | stud_name
------------+-------------------+----------+--------+-------------
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen

(1 rows)
```

Objective: To sort or order the rows/records of the "project_details" column in ascending order of project_name.

Act:

```
SELECT *
    FROM project_details
        WHERE project_id IN (1,2)
            ORDER BY project_name DESC;
```

Outcome:

```
cqlsh:students> SELECT * FROM project_details WHERE project_id IN (1,2) ORDER BY project_name DESC;

 project_id | project_name       | duration | rating | stud_name
------------+--------------------+----------+--------+-------------
          2 |       SAP Reporting |     3000 |    4.2 | Stephen Fox
          2 |          SAP BI DW |     9000 |      4 | Stephen Fox
          1 | MS data Migration  |      720 |    3.5 | David Sheen
          1 | MS Data Warehouse  |     1440 |    3.9 | David Sheen

(4 rows)
```

Note: By default sorting or ordering is done in ascending order. The user can specify the order by using the keyword "ASC" for ascending or "DESC" for descending.

ORDER BY clause can select a single column only. This column is the second column of the compound primary key. This applies even when the compound primary key has more than two columns.

When specifying the ORDER BY clause, use only the column name and not the column alias.

7.7 COLLECTIONS

7.7.1 Set Collection

A column of type set consists of unordered unique values. However, when the column is queried, it returns the values in sorted order. For example, for text values, it sorts in alphabetical order.

PICTURE THIS...

You are required to store details about users of service "xyz". The details of the user include: User_ID, User_Name, User_Contact_Nos, User_Email_Ids. A user may have n number of Contact Nos and also may have n number of Email IDs. How do we accomplish this task in RDBMS?

We would create a table, let us say "Users", to store details such as "User_ID", "User_Name" and another table "UsersContactDetails". The relationship between "UsersContactDetails" and "Users" is many-to-one. Likewise, we would create a table "UsersEmailIDs" and establish a many-to-one relationship between "UserEmailIDs" and the "Users" table.

However, the multiple Contact Nos and multiple Email IDs problem can be solved by defining a column as a collection. The usage of collection types for columns is not only convenient but intuitive as well.

CQL makes use of the following collection types:

- Set
- List
- Map

When to use collection?

Use collection when it is required to store or denormalize a small amount of data.

What is the limit on the values of items in a collection?

The values of items in a collection are limited to 64K.

Where to use collections?

Collections can be used when you need to store the following:

1. Phone numbers of users.
2. Email ids of users.

When should one refrain from using a collection?

One should refrain from using a collection when the data has unbound growth potential such as all the messages posted by a user or all the event data as captured by a sensor. When faced with such a situation, use a table with compound primary key with data being held in clustering columns.

7.7.2 List Collection

When the order of elements matter, one should go for a list collection. For example, when you store the preferences of places to visit by a user, you would like to respect his preferences and retrieve the values in the order in which he has entered rather than in sorted order. A list also allows one to store the same value multiple times.

7.7.3 Map Collection

As the name implies, a map is used to map one thing to another. A map is a pair of typed values. It is used to store timestamp related information. Each element of the map is stored as a Cassandra column. Each element can be individually queried, modified, and deleted.

Objective: To alter the schema for the table "student_info" to add a column "hobbies".

Act:

ALTER TABLE student_info ADD hobbies set<text>;

Outcome:

```
cqlsh:students> ALTER TABLE student_info ADD hobbies set<text>;
```

Confirm the structure of the table after the change has been made:

```
cqlsh:students> describe table student_info;

CREATE TABLE student_info (
  rollno int,
  dateofjoining timestamp,
  hobbies set<text>,
  lastexampercent double,
  studname text,
  PRIMARY KEY (rollno)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

CREATE INDEX student_info_lastexampercent_idx ON student_info (lastexampercent);

CREATE INDEX student_info_studname_idx ON student_info (studname);
```

Objective: To alter the schema of the table "student_info" to add a list column "language".

Act:

ALTER TABLE student_info ADD language list<text>;

Outcome:

```
cqlsh:students> ALTER TABLE student_info ADD language list<text>;
cqlsh:students>
```

Confirm the structure of the table after the change has been made:

```
cqlsh:students> describe table student_info;

CREATE TABLE student_info (
  rollno int,
  dateofjoining timestamp,
  hobbies set<text>,
  language list<text>,
  lastexampercent double,
  studname text,
  PRIMARY KEY (rollno)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

CREATE INDEX student_info_lastexampercent_idx ON student_info (lastexampercent);

CREATE INDEX student_info_studname_idx ON student_info (studname);

cqlsh:students>
```

Objective: To update the table "student_info" to provide the values for "hobbies" for the student with Rollno =1.

Act:

```
UPDATE student_info
    SET hobbies = hobbies + {'Chess, Table Tennis'}
        WHERE RollNo=1;
```

Outcome:

```
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Chess, Table Tennis'} WHERE RollNo=1;
```

To confirm the values in the hobbies column, use the below command:

```
SELECT *
    FROM student_info
        WHERE RollNo=1;
```

```
cqlsh:students> select * from student_info where RollNo=1;

 rollno | dateofjoining                           | hobbies                | language | lastexampercent | studname
--------+-----------------------------------------+------------------------+----------+-----------------+---------------
      1 | 2012-03-29 00:00:00India Standard Time | {'Chess, Table Tennis'} |     null |            69.6 | Michael Storm

(1 rows)
```

Likewise update a few more records:

```
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Chess, Badminton'} WHERE RollNo=3;
cqlsh:students>
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Lawn Tennis, Table Tennis, Golf'} WHERE RollNo=4;
cqlsh:students>
```

Records after the updation:

```
cqlsh:students> select * from student_info;

 rollno | dateofjoining                          | hobbies                                  | language | lastexampercent | studna
--------+----------------------------------------+------------------------------------------+----------+-----------------+-------
      1 | 2012-03-29 00:00:00India Standard Time |                {'Chess, Table Tennis'} |     null |            69.6 |   Mich
      4 | 2012-05-11 00:00:00India Standard Time | {'Lawn Tennis, Table Tennis, Golf'} |     null |            73.4 |      I
      3 | 2014-04-12 00:00:00India Standard Time |                 {'Chess, Badminton'} |     null |            81.7 | David

(3 rows)
```

Objective: To update values in the list column, "language" of the table "student_info".

Act:

```
UPDATE student_info
    SET language = language + ['Hindi, English']
        WHERE RollNo=1;
```

Outcome:

```
cqlsh:students> UPDATE student_info
           ...    SET language = language + ['Hindi, English'] WHERE RollNo=1;
cqlsh:students>
```

Likewise update the remaining records.

```
cqlsh:students> UPDATE student_info
           ...    SET language = language + ['Hindi,English,French'] WHERE RollNo=3;
cqlsh:students>
cqlsh:students> UPDATE student_info
           ...    SET language = language + ['Hindi, English'] WHERE RollNo=4;
cqlsh:students>
```

To view the updates to the records, use the below statement:

```
cqlsh:students> select rollNo, studname, hobbies, language from student_info;

 rollno | studname        | hobbies                                  | language
--------+-----------------+------------------------------------------+-----------------------------
      1 |   Michael Storm |                {'Chess, Table Tennis'} |       ['Hindi, English']
      4 |      Ian String | {'Lawn Tennis, Table Tennis, Golf'} |       ['Hindi, English']
      3 | David Flemming |                 {'Chess, Badminton'} | ['Hindi,English,French']

(3 rows)
```

7.7.4 More Practice on Collections (SET and LIST)

Objective: To create a table "users" with an "emails" column. The type of this column "emails" is "set".

Act:

```
CREATE TABLE users (
        user_id text PRIMARY KEY,
        first_name text,
        last_name text,
        emails set<text>
);
```

Outcome:

```
cqlsh:students> CREATE TABLE users (
            ...   user_id text PRIMARY KEY,
            ...   first_name text,
            ...   last_name text,
            ...   emails set<text>
            ... );
cqlsh:students>
```

Objective: To insert values into the "emails" column of the "users" table.

Note: Set values must be unique.

Act:

```
INSERT INTO users
        (user_id, first_name, last_name, emails)
            VALUES('AB', 'Albert', 'Baggins', {'a@baggins.com', 'baggins@gmail.com'});
```

Outcome:

```
cqlsh:students> INSERT INTO users (user_id, first_name, last_name, emails)
            ...   VALUES('AB', 'Albert', 'Baggins', {'a@baggins.com', 'baggins@gmail.com'});
cqlsh:students>
```

Objective: Add an element to a set using the UPDATE command and the addition (+) operator.

Act:

```
UPDATE users
        SET emails = emails + {'ab@friendsofmordor.org'}
            WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> UPDATE users
            ...   SET emails = emails + {'ab@friendsofmordor.org'} WHERE user_id = 'AB';
cqlsh:students>
```

Objective: To retrieve email addresses for Albert from the set.

Act:

```
SELECT user_id, emails
    FROM users
        WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> SELECT user_id, emails FROM users WHERE user_id = 'AB';

 user_id | emails
---------+-----------------------------------------------------------------
      AB | {'a@baggins.com', 'ab@friendsofmordor.org', 'baggins@gmail.com'}

(1 rows)
```

Objective: To remove an element from a set using the subtraction (–) operator.

Act:

```
UPDATE users
    SET emails = emails - {'ab@friendsofmordor.org'}
        WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> UPDATE users
           ...   SET emails = emails - {'ab@friendsofmordor.org'} WHERE user_id = 'AB';
```

To view the records from the "users" table:

```
cqlsh:students> select * from users;

 user_id | emails                                  | first_name | last_name
---------+-----------------------------------------+------------+-----------
      AB | {'a@baggins.com', 'baggins@gmail.com'} |     Albert |   Baggins

(1 rows)
```

Objective: To remove all elements from a set by using the UPDATE or DELETE statement.

Act:

```
UPDATE users
    SET emails = {}
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users SET emails = {} WHERE user_id = 'AB';
```

Outcome: The above command removes the emails column. Here is the confirmation:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name
---------+--------+------------+-----------
      AB |   null |     Albert |   Baggins

(1 rows)
```

OR

```
DELETE emails
    FROM users
        WHERE user_id = 'AB';
```

```
cqlsh:students> DELETE emails FROM users WHERE user_id = 'AB';
```

The above command removes the emails column. Here is the confirmation:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name
---------+--------+------------+-----------
      AB |   null |     Albert |   Baggins

(1 rows)
```

Objective: To alter the "users" table to add a column, "top_places" of type list.

Act:

```
ALTER TABLE users ADD top_places list<text>;
```

```
cqlsh:students> ALTER TABLE users ADD top_places list<text>;
```

Outcome:

The above command alters the structure of the table, "users". Here is the confirmation.

```
cqlsh:students> describe table users;

CREATE TABLE users (
  user_id text,
  emails set<text>,
  first_name text,
  last_name text,
  top_places list<text>,
  PRIMARY KEY (user_id)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

Objective: To update the list column "top_places" in the "users" table for user_id = 'AB'.

Act:

```
UPDATE users
    SET top_places = [ 'Lonavla', 'Khandala' ]
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users
           ...   SET top_places = [ 'Lonavla', 'Khandala' ] WHERE user_id = 'AB';
cqlsh:students>
```

Outcome:

```
cqlsh:students> select * from users where user_id = 'AB';

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-------------------------
      AB |   null |     Albert |   Baggins | ['Lonavla', 'Khandala']

(1 rows)
```

Objective: Prepend an element to the list by enclosing it in square brackets and using the addition (+) operator.

Act:

```
UPDATE users
    SET top_places = [ 'Mahabaleshwar' ] + top_places
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users
           ...   SET top_places = [ 'Mahabaleshwar' ] + top_places WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)
```

Objective: To append an element to the list by switching the order of the new element data and the list name in the update command.

Act:

```
UPDATE users
    SET top_places = top_places + [ 'Tapola' ]
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users
           ...    SET top_places = top_places + [ 'Tapola' ] WHERE user_id = 'AB';
cqlsh:students>
```

Outcome:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Objective: To query the database for a list of top places.

Act:

```
SELECT user_id, top_places
        FROM users
                WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> SELECT user_id, top_places FROM users WHERE user_id = 'AB';

 user_id | top_places
---------+-----------------------------------------------------
      AB | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Objective: To remove an element from a list using the DELETE command and the list index position in square brackets.

The record as it exists prior to deletion is

```
cqlsh:students> SELECT user_id, top_places FROM users WHERE user_id = 'AB';

 user_id | top_places
---------+-----------------------------------------------------
      AB | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Act:

```
DELETE top_places[3]
        FROM users
                WHERE user_id = 'AB';
```

```
cqlsh:students> DELETE top_places[3] FROM users WHERE user_id = 'AB';
```

Outcome: The status after deletion is

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-------------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)
```

7.7.5 Using Map: Key, Value Pair

Objective: To alter the "users" table to add a map column "todo".

Act:

```
ALTER TABLE users
      ADD todo map<timestamp, text>;
```

```
cqlsh:students> ALTER TABLE users ADD todo map<timestamp, text>;
```

Outcome:

```
cqlsh:students> describe table users;

CREATE TABLE users (
  user_id text,
  emails set<text>,
  first_name text,
  last_name text,
  todo map<timestamp, text>,
  top_places list<text>,
  PRIMARY KEY (user_id)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

cqlsh:students>
```

Objective: To update the record for user (user_id = 'AB') in the "users" table.

Act: The record from user_id = 'AB' as it exists in the "users" table is

```
cqlsh:students> select * from users where user_id='AB';

 user_id | emails | first_name | last_name | todo | top_places
---------+--------+------------+-----------+------+-----------------------------------------
      AB |   null |     Albert |   Baggins | null | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)
```

```
cqlsh:students> UPDATE users
            ...   SET todo =
            ...   {'2014-9-24':'Cassandra Session',
            ...    '2014-10-2 12:00' :'MongoDB Session'}
            ...   WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select user_id, todo from users where user_id='AB';

r_id | todo
-----+-----------------------------------------------------------------------------------------------------------------
 AB | {'2014-09-24 00:00:00India Standard Time': 'Cassandra Session', '2014-10-02 12:00:00India Standard Time': 'MongoDB Se
```

Objective: To delete an element from the map using the DELETE command and enclosing the timestamp of the element in square brackets.

Act:

```
DELETE todo['2014-9-24']
        FROM users
              WHERE user_id = 'AB';
```

```
cqlsh:students> DELETE todo['2014-9-24'] FROM users WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select user_id, todo from users where user_id='AB';

 user_id | todo
---------+-----------------------------------------------------------------
      AB | {'2014-10-02 12:00:00India Standard Time': 'MongoDB Session'}

(1 rows)
```

7.8 USING A COUNTER

A counter is a special column that is changed in increments. For example, we may need a counter column to count the number of times a particular book is issued from the library by the student.

Step 1:

```
CREATE TABLE library_book (
        counter_value counter,
        book_name varchar,
        stud_name varchar,
        PRIMARY KEY (book_name, stud_name)
    );
```

```
cqlsh:students> CREATE TABLE library_book
           ...   (     counter_value counter,
           ...         book_name varchar,
           ...         stud_name varchar,
           ...   PRIMARY KEY (book_name, stud_name)
           ... );
```

Step 2: Load data into the counter column.

```
UPDATE library_book
      SET counter_value = counter_value + 1
            WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

```
cqlsh:students> UPDATE library_book
           ...  SET counter_value = counter_value + 1
           ... WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

Step 3: Take a look at the counter value.

```
SELECT *
    FROM library_book;
```

Output is:

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             1

(1 rows)
```

Step 4: Let us increase the value of the counter.

```
UPDATE library_book
    SET counter_value = counter_value + 1
        WHERE book_name='Fundamentals of Business Analytics' AND stud_name='shaan';
```

```
cqlsh:students> UPDATE library_book
           ...   SET counter_value = counter_value + 1
           ... WHERE book_name='Fundamentals of Business Analytics' AND stud_name='shaan';
```

Step 5: Again, take a look at the counter value.

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             1
 Fundamentals of Business Analytics |     shaan |             1

(2 rows)
```

Step 6: Update another record for Stud_name "Jeet".

```
UPDATE library_book
    SET counter_value = counter_value + 1
        WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

```
cqlsh:students> UPDATE library_book
           ...   SET counter_value = counter_value + 1
           ... WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

Step 7: Let us take a look at the counter value, one last time.

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             2
 Fundamentals of Business Analytics |     shaan |             1

(2 rows)
```

7.9 TIME TO LIVE (TTL)

Data in a column, other than a counter column, can have an optional expiration period called TTL (time to live). The client request may specify a TTL value for the data. The TTL is specified in seconds.

```
CREATE TABLE userlogin(
    userid int primary key, password text
);
```

```
cqlsh:students> CREATE TABLE  userlogin(
           ... userid int primary key, password text
           ... );
```

INSERT INTO userlogin (userid, password)

VALUES (1,'infy') USING TTL 30;

```
cqlsh:students> INSERT INTO userlogin (userid, password)
           ... VALUES (1,'infy') USING TTL 30;
```

SELECT TTL (password)

FROM userlogin

WHERE userid=1;

```
cqlsh:students> SELECT TTL (password)
           ... FROM userlogin
           ... WHERE userid=1;

 ttl(password)
---------------
            18

(1 rows)
```

7.10 ALTER COMMANDS

Let us look at a few Alter commands to bring about changes to the structure of the table/column family.

1. Create a table "sample" with columns "sample_id" and "sample_name".

CREATE TABLE sample(

sample_id text,

sample_name text,

PRIMARY KEY (sample_id)

);

```
cqlsh:students> Create table sample (sample_id text, sample_name text, primary key (sample_id));
cqlsh:students>
```

2. Insert a record into the table "sample".

INSERT INTO sample(

sample_id, sample_name)

VALUES ('S101', 'Big Data');

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 'S101', 'Big Data' );
```

3. View the records of the table "sample".

SELECT *

FROM sample;

```
cqlsh:students> select * from sample;

 sample_id | sample_name
-----------+-------------
      S101 |    Big Data

(1 rows)
```

7.10.1 Alter Table to Change the Data Type of a Column

1. Alter the schema of the table "sample". Change the data type of the column "sample_id" to integer from text.

ALTER TABLE sample
ALTER sample_id TYPE int;

```
cqlsh:students> alter table sample alter sample_id type int;
```

2. After the data type of the column "sample_id" is changed from text to integer, try inserting a record as follows and observe the error message:

INSERT INTO sample(sample_id, sample_name)
VALUES('S102', 'Big Data');

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 'S102', 'Big Data' );
Bad Request: Invalid STRING constant (S102) for sample_id of type int
cqlsh:students>
```

3. Try inserting a record as given below into the table "sample".

INSERT INTO sample(sample_id, sample_name)
VALUES(102, 'Big Data');

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 102, 'Big Data' );
cqlsh:students> select * from sample;

 sample_id  | sample_name
------------+-------------
 1395732529 |    Big Data
        102 |    Big Data

(2 rows)
```

4. Alter the data type of the "sample_id" column to varchar from integer.

ALTER TABLE sample
ALTER sample_id TYPE varchar;

```
cqlsh:students> alter table sample alter sample_id type varchar;
```

5. Check the records after the data type of "sample_id" has been changed to varchar from integer.

```
cqlsh:students> select * from sample;

 sample_id      | sample_name
----------------+-------------
           S101 |    Big Data
 \x00\x00\x00f |    Big Data

(2 rows)
```

7.10.2 Alter Table to Delete a Column

1. Drop the column "sample_id" from the table "sample".

ALTER TABLE sample
DROP sample_id;

```
cqlsh:students> alter table sample drop sample_id;
Bad Request: Cannot drop PRIMARY KEY part sample_id
```

Note: The request to drop the "sample_id" column from the table "sample" does not succeed as it is the primary key column.

2. Drop the column "sample_name" from the table "sample".

ALTER TABLE sample
DROP sample_name;

```
cqlsh:students> alter table sample drop sample_name;
```

Note: the above request to drop the column "sample_name" from table "sample" succeeds.

7.10.3 Drop a Table

1. Drop the column family/table "sample".

DROP columnfamily sample;

```
cqlsh:students> drop columnfamily sample;
```

The above request succeeds. The table/column family no longer exists in the keyspace.

2. Confirm the non-existence of the table "sample" in the keyspace by giving the following command:

```
cqlsh:students> describe table sample;

Column family 'sample' not found
```

7.10.4 Drop a Database

1. Drop the keyspace "students".

DROP keyspace students;

```
cqlsh:students> drop keyspace students;
```

2. Confirm the non-existence of the keyspace "students" by issuing the following command:

```
cqlsh:students> describe keyspace students;

Keyspace 'students' not found.
```

7.11 IMPORT AND EXPORT

7.11.1 Export to CSV

Objective: Export the contents of the table/column family "elearninglists" present in the "students" database to a CSV file (d:\elearninglists.csv).

Act:

Step 1: Check the records of the table "elearninglists" present in the "students" database.

SELECT *
FROM elearninglists;

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Step 2: Execute the below command at the cqlsh prompt:

COPY elearninglists (id, course_order, course_id, courseowner, title) TO 'd:\elearninglists.csv';

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) to 'd:\elearninglists.csv';
4 rows exported in 0.000 seconds.
cqlsh:students>
```

Step 3: Check the existence of the "elearninglists.csv" file in "D:\". Given below is the content of the "d:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

7.11.2 Import from CSV

Objective: To import data from "D:\elearninglists.csv" into the table "elearninglists" present in the "students" database.

Step 1: Check for the table "elearninglists" in the "students" database. If the table is already present, truncate the table. This will remove all records from the table but retain the structure of the table. In our case, the table "elearninglists" is already present in the "students" database. Let us take a look at the records of the "elearninglists" before we run the truncate command on it.

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Truncate the table using the below command:

TRUNCATE elearninglists;

```
cqlsh:students> Truncate  elearninglists;
cqlsh:students>
```

Note: No record is present in the table "elearninglists". The structure/schema is however preserved. We confirm it by executing the below command at the cqlsh prompt.

```
cqlsh:students> select * from elearninglists;

(0 rows)

cqlsh:students>
```

Step 2: Check for the content of the "D:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

Note: The content in the CSV agrees with the structure of the table "elearninglists" in the "students" database. The structure should be such that the content from the CSV can be housed within it without any issues.

Step 3: Execute the below command to import data from "d:\elearninglists.csv" into the table "elearninglists" in the database "students".

```
COPY elearninglists (id, course_order, course_id, courseowner, title) FROM 'd:\elearninglists.csv';
```

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) from 'd:\elearninglists.csv';
4 rows imported in 0.031 seconds.
cqlsh:students>
```

Step 4: Confirm that records have been imported into the table.

```
SELECT *
    FROM elearninglists;
```

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)

cqlsh:students>
```

7.11.3 Import from STDIN

Objective: To import data into an existing table "persons" present in the "students" database. The data is to be provided by the user using the standard input device.

Step 1: Ensure that the table "persons" exists in the database "students".

```
DESCRIBE TABLE persons;
```

```
cqlsh:students> describe table persons;

CREATE TABLE persons (
  id int,
  fname text,
  lname text,
  PRIMARY KEY (id)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

cqlsh:students>
```

Step 2:

COPY persons (id, fname, lname) FROM STDIN;

```
cqlsh:students> COPY persons (id, fname, lname) FROM STDIN;
[Use \. on a line by itself to end input]
[copy] 1,"Samuel","Jones"
[copy] 2,"Virat","Kumar"
[copy] 3,"Andrew","Simon"
[copy] 4,"Raul","A Simpson"
[copy] \.

4 rows imported in 1 minute and 24.336 seconds.
cqlsh:students>
```

Step 3: Confirm that the records from the standard input device are loaded into the "persons" table existing in the "students" database.

SELECT *
FROM persons;

```
cqlsh:students> select * from persons;

 id | fname  | lname
----+--------+-----------
  1 | Samuel |     Jones
  2 |  Virat |     Kumar
  4 |   Raul | A Simpson
  3 | Andrew |     Simon

(4 rows)

cqlsh:students>
```

7.11.4 Export to STDOUT

Objective: Export the contents of the table/column family "elearninglists" present in the "students" database to the standard output device (STDOUT).

Act:

Step 1: Check the records of the table "elearninglists" present in the "students" database.

SELECT *
FROM elearninglists;

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Step 2: Execute the below command at the cqlsh prompt.

COPY elearninglists (id, course_order, course_id, courseowner, title) TO STDOUT;

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) to STDOUT;
101,1,1001,Subhashini,NoSQL Cassandra
101,2,1002,Seema,NoSQL MongoDB
101,3,1003,Seema,Hadoop Sqoop
101,4,1004,Subhashini,Hadoop Flume
4 rows exported in 0.031 seconds.
cqlsh:students>
```

7.12 QUERYING SYSTEM TABLES

There are quite a few system tables such as schema_keyspaces, schema_columnfamilies, schema_columns, local, peers, etc. Let us look at what each of these system tables store in them.

SELECT *
FROM system.schema_keyspaces;

```
cqlsh:system> describe table system.schema_keyspaces;

CREATE TABLE schema_keyspaces (
  keyspace_name text,
  durable_writes boolean,
  strategy_class text,
  strategy_options text,
  PRIMARY KEY (keyspace_name)
) WITH COMPACT STORAGE AND
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='keyspace definitions' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=8640 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

SELECT *
FROM system.schema_columnfamilies;

```
CREATE TABLE schema_columnfamilies (
  keyspace_name text,
  columnfamily_name text,
  bloom_filter_fp_chance double,
  caching text,
  column_aliases text,
  comment text,
  compaction_strategy_class text,
  compaction_strategy_options text,
  comparator text,
  compression_parameters text,
  default_time_to_live int,
  default_validator text,
```

```
  dropped_columns map<text, bigint>,
  gc_grace_seconds int,
  index_interval int,
  key_aliases text,
  key_validator text,
  local_read_repair_chance double,
  max_compaction_threshold int,
  memtable_flush_period_in_ms int,
  min_compaction_threshold int,
  populate_io_cache_on_flush boolean,
  read_repair_chance double,
  replicate_on_write boolean,
  speculative_retry text,
  subcomparator text,
  type text,
  value_alias text,
  PRIMARY KEY (keyspace_name, columnfamily_name)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='ColumnFamily definitions' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=8640 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

SELECT *

FROM system.schema_columns;

```
cqlsh:system> describe table system.schema_columns;

CREATE TABLE schema_columns (
  keyspace_name text,
  columnfamily_name text,
  column_name text,
  component_index int,
  index_name text,
  index_options text,
  index_type text,
  type text,
  validator text,
  PRIMARY KEY (keyspace_name, columnfamily_name, column_name)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='ColumnFamily column attributes' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=8640 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

```
SELECT *
       FROM system.local;
CREATE TABLE local (
  key text,
  bootstrapped text,
  cluster_name text,
  cql_version text,
  data_center text,
  gossip_generation int,
  host_id uuid,
  native_protocol_version text,
  partitioner text,
  rack text,
  release_version text,
  schema_version uuid,
  thrift_version text,
  tokens set<text>,
  truncated_at map<uuid, blob>,
  PRIMARY KEY (key)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='information about the local node' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=0 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

SELECT *
       FROM system.peers;

CREATE TABLE peers (
  peer inet,
  data_center text,
  host_id uuid,
  preferred_ip inet,
  rack text,
  release_version text,
  rpc_address inet,
  schema_version uuid,
  tokens set<text>,
  PRIMARY KEY (peer)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='known peers in the cluster' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=0 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

7.13 PRACTICE EXAMPLES

Objective: To create table "elearninglist" with columns: id, course_order, course_id, title, courseowner.

Act:

```
CREATE TABLE elearninglists (
        id int,
        course_order int,
        course_id int,
        title text,
        courseowner text,
        PRIMARY KEY (id, course_order )
        );
```

Here, id ==> Partition Key, course_order ==> Clustering Column. The combination of the id and course_order in the elearninglists table uniquely identifies a row in the elearninglists table. You can have more than one row with the same id as long as the rows contain different course_order values.

Outcome:

```
cqlsh:students> CREATE TABLE elearninglists (
           ...   id int,
           ...   course_order int,
           ...   course_id int,
           ...   title text,
           ...   courseowner text,
           ...   PRIMARY KEY  (id, course_order ) );
cqlsh:students>
```

Objective: To insert rows into the table "elearninglists".

Act:

```
INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 1, 1001,'NoSQL Cassandra','Subhashini');

INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 2, 1002,'NoSQL MongoDB','Seema');

INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 3, 1003,'Hadoop Sqoop','Seema');

INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 4, 1004,'Hadoop Flume', 'Subhashini');
```

Outcome:

```
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101,1,1001,'NoSQL Cassandra','Subhashini');
cqlsh:students>
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101,2,1002,'NoSQL MongoDB','Seema');
cqlsh:students>
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101,3,1003,'Hadoop Sqoop','Seema',);
Bad Request: line 2:44 no viable alternative at input ')'
cqlsh:students>
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101, 4,1004,'Hadoop Flume', 'Subhashini');
cqlsh:students>
```

Objective: To query the table "elearninglists".

Act:

```
SELECT *
    FROM elearninglists;
```

Outcome:

```
cqlsh:students> SELECT * FROM elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(3 rows)
```

Objective: To query the "elearninglist" table on "courseowner" as a filter.

Act:

```
SELECT *
    FROM elearninglists
        WHERE courseowner = 'Seema';
```

Outcome: The query returns an error stating "No indexed columns present in by-columns clause with Equal operator".

```
cqlsh:students> SELECT * FROM elearninglists WHERE courseowner = 'Seema';
Bad Request: No indexed columns present in by-columns clause with Equal operator
cqlsh:students>
```

Solution: Create an index on courseowner column.

```
CREATE INDEX ON
    Elearninglists(courseowner);
```

```
cqlsh:students> CREATE INDEX ON elearninglists(courseowner);
cqlsh:students>
```

Executing the same query now shows the record:

```
cqlsh:students> SELECT * FROM elearninglists WHERE courseowner = 'Seema';

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            2 |      1002 |       Seema | NoSQL MongoDB

(1 rows)
cqlsh:students>
```

Objective: To order all the rows of elearninglists with id = 101 in descending order of "course_order". The maximum number of records to retrieve is 50.

Act:

```
SELECT *
    FROM elearninglists
        WHERE id = 101
            ORDER BY course_order DESC LIMIT 50;
```

Outcome:

```
cqlsh:students> SELECT * FROM elearninglists WHERE id = 101 ORDER BY course_order DESC LIMIT 50;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra

(3 rows)
```

REMIND ME

- Apache Cassandra was born at Facebook. After Facebook open sourced the code in 2008, Cassandra became an Apache Incubator project in 2009 and subsequently became a top-level Apache project in 2010.
- Cassandra does NOT compromise on availability. Since it does not have a master-slave architecture, there is no question of single point of failure. This proves beneficial for business critical applications that need to be up and running always and cannot afford to go down ever.
- A replication strategy is employed to determine which nodes to place the data on. Two replication strategies are available:
  - SimpleStrategy.
  - NetworkToplogyStrategy.
- One of the features of Cassandra that has made it immensely popular is its ability to utilize tunable consistency. The database systems can go for either strong consistency or eventual consistency.
- Read consistency means how many replicas must respond before sending out the result to the client application.
- Write consistency means on how many replicas write must succeed before sending out an acknowledgement to the client application.

POINT ME (BOOK)

- ***Cassandra:*** The Definitive Guide By Eben Hewitt, Publisher: O'Reilly Media.

CONNECT ME (INTERNET RESOURCES)

- http://www.datastax.com/documentation/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html
- http://www.datastax.com/documentation/cql/3.1/pdf/cql31.pdf
- http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html
- http://www.datastax.com/docs/1.0/cluster_architecture/about_client_requests
- http://www.datastax.com/docs/datastax_enterprise3.1/solutions/about_pig

TEST ME

A. Unsolved Exercises

1. What is Cassandra?
2. Comment on Cassandra writes.
3. What is your understanding of tunable consistency?
4. What are collections in CQLSH? Where are they used?
5. Explain hinted handoffs.
6. What is Cassandra – cli?
7. Explain Cassandra's data model.
8. Explain the replication strategy in Cassandra.
9. Cassandra adheres to the Availability and Partition tolerant traits as stated by the CAP theorem. Explain.

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1: COLLECTIONS

Objective: To learn about the various collection types: Set, List and Map.
Problem Description: Design a table/column family to support the following requirements.

- Store the basic information about students such as Student Roll No, Student Name, Student Date of Birth, and Student Address.
- Store the subject preferences of each student. There should be a minimum of two subject preferences and a maximum of four. The order of preferences as given by the student should be preserved.
- Store the hobbies of each student. There should be a minimum of two hobbies and a maximum of four. The hobbies as given by the student should be arranged in alphabetical order.

ASSIGNMENT 2: TIME TO LIVE

Objective: To learn about the TTL type (Time To Live).

Problem Description: Design a table/column family to support the following requirements.

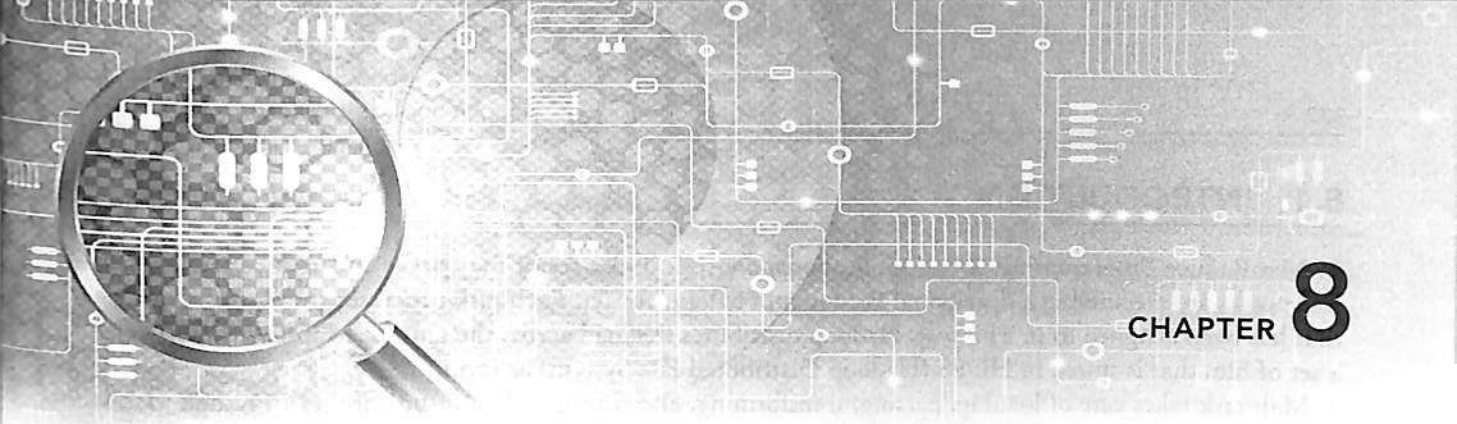
Store the login details of the user such as UserID and Password. The information stored should expire in a day's time.

ASSIGNMENT 3: IMPORT FROM CSV

Objective: To learn about the import from CSV to Cassandra table/column family.

Problem Description: Read a public dataset from the site www.kdnuggets.com. If not already in CSV format, first convert to CSV format and then import into a Cassandra table/column family by the name "PublicDataSet" in the "Sample" database.

Confirm the presence of data in the table "PublicDataSet" in the "Sample" database.



CHAPTER 8

Introduction to MAPREDUCE Programming

BRIEF CONTENTS

- What's in Store?
- Introduction
- Mapper
  - RecordReader
  - Map
  - Combiner
  - Partitioner
- Reducer
  - Shuffle
  - Sort
  - Reduce
  - Output Format
- Combiner
- Partitioner
- Searching
- Sorting
- Compression

"The alchemists in their search for gold discovered many other things of greater value."

– Arthur Schopenhauer, German Philosopher

WHAT'S IN STORE?

We assume that you are familiar with the basic concepts of HDFS and MapReduce Programming discussed in Chapters 4 and 5. The focus of this chapter will be to build on this knowledge to understand optimization techniques of MapReduce Programming such as combiner, partitioner, and compression. We will also discuss how to write MapReduce Programming for sorting and searching.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning and comprehension.

8.1 INTRODUCTION

In MapReduce Programming, Jobs (Applications) are split into a set of map tasks and reduce tasks. Then these tasks are executed in a distributed fashion on Hadoop cluster. Each task processes small subset of data that has been assigned to it. This way, Hadoop distributes the load across the cluster. MapReduce job takes a set of files that is stored in HDFS (Hadoop Distributed File System) as input.

Map task takes care of loading, parsing, transforming, and filtering. The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output. Each map task is broken into the following phases:

1. RecordReader.
2. Mapper.
3. Combiner.
4. Partitioner.

The output produced by map task is known as intermediate keys and values. These intermediate keys and values are sent to reducer. The reduce tasks are broken into the following phases:

1. Shuffle.
2. Sort.
3. Reducer.
4. Output Format.

Hadoop assigns map tasks to the DataNode where the actual data to be processed resides. This way, Hadoop ensures data locality. Data locality means that data is not moved over network; only computational code is moved to process data which saves network bandwidth.

8.2 MAPPER

A mapper maps the input key–value pairs into a set of intermediate key–value pairs. Maps are individual tasks that have the responsibility of transforming input records into intermediate key–value pairs.

1. **RecordReader:** RecordReader converts a byte-oriented view of the input (as generated by the InputSplit) into a record-oriented view and presents it to the Mapper tasks. It presents the tasks with keys and values. Generally the key is the positional information and value is a chunk of data that constitutes the record.
2. **Map:** Map function works on the key–value pair produced by RecordReader and generates zero or more intermediate key–value pairs. The MapReduce decides the key–value pair based on the context.
3. **Combiner:** It is an optional function but provides high performance in terms of network bandwidth and disk space. It takes intermediate key–value pair provided by mapper and applies user-specific aggregate function to only that mapper. It is also known as local reducer.
4. **Partitioner:** The partitioner takes the intermediate key–value pairs produced by the mapper, splits them into shard, and sends the shard to the particular reducer as per the user-specific code. Usually, the key with same values goes to the same reducer. The partitioned data of each map task is written to the local disk of that machine and pulled by the respective reducer.

8.3 REDUCER

The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values. The Reducer has three primary phases: Shuffle and Sort, Reduce, and Output Format.

1. **Shuffle and Sort:** This phase takes the output of all the partitioners and downloads them into the local machine where the reducer is running. Then these individual data pipes are sorted by keys which produce larger data list. The main purpose of this sort is grouping similar words so that their values can be easily iterated over by the reduce task.
2. **Reduce:** The reducer takes the grouped data produced by the shuffle and sort phase, applies reduce function, and processes one group at a time. The reduce function iterates all the values associated with that key. Reducer function provides various operations such as aggregation, filtering, and combining data. Once it is done, the output (zero or more key–value pairs) of reducer is sent to the output format.
3. **Output Format:** The output format separates key–value pair with tab (default) and writes it out to a file using record writer.

Figure 8.1 describes the chores of Mapper, Combiner, Partitioner, and Reducer for the word count problem. The Word Count problem has been discussed under "Combiner" and "Partitioner".

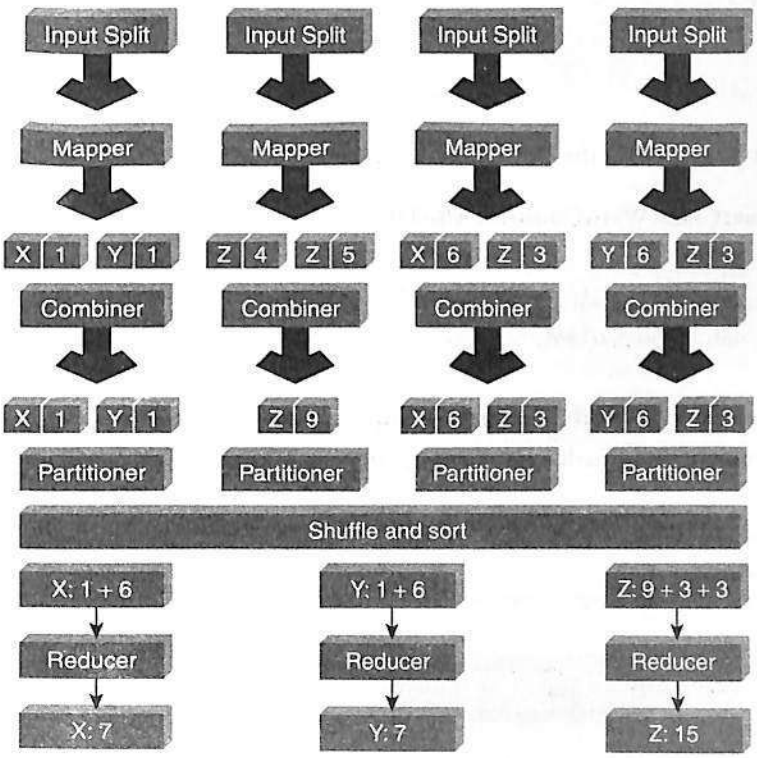


Figure 8.1 The chores of Mapper, Combiner, Partitioner, and Reducer.

8.4 COMBINER

It is an optimization technique for MapReduce Job. Generally, the reducer class is set to be the combiner class. The difference between combiner class and reducer class is as follows:

1. Output generated by combiner is intermediate data and it is passed to the reducer.
2. Output of the reducer is passed to the output file on disk.

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?
Input Data: What is the input that has been given to us to act upon?
Act: The actual statement/command to accomplish the task at hand.
Output: The result/output as a consequence of executing the statement.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use combiner for optimization.

Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act: In the driver program, set the combiner class as shown below.

```
job.setCombinerClass(WordCounterRed.class);

// Input and Output Path
FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new Path("/mapreducedemos/output/wordcount/"));
```

hadoop jar <<jar name>> <<driver class>> <<input path>> <<output path>>

Here driver class name, input path, and output path are optional arguments.

Output:

```
[root@volgalnx010 mapreducedemos]# hadoop jar wordcount.jar
```

Contents of directory /mapreducedemos

Goto : /mapreducedemos go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
lines.txt	file	91 B	3	128 MB	2015-03-01 21:05	rw-r--r--	root	supergroup
output	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Contents of directory /mapreducedemos/output

Goto : /mapreducedemos/output go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
wordcount	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

The reducer output will be stored in part-r-00000 file by default.

Contents of directory /mapreducedemos/output/wordcount

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup
part-r-00000	file	76 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

File: /mapreducedemos/output/wordcount/part-r-00000

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
Introducing     1
Introduction    1
Pig     1
Session 3
Welcome 1
to      2
```

8.5 PARTITIONER

The partitioning phase happens after map phase and before reduce phase. Usually the number of partitions are equal to the number of reducers. The default partitioner is hash partitioner.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use partitioner to partition key based on alphabets.
Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act:

WordCountPartitioner.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

public class WordCountPartitioner extends Partitioner<Text, IntWritable> {
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
                String word = key.toString();
                char alphabet = word.toUpperCase().charAt(0);
                int partitionNumber = 0;
                switch(alphabet) {
                case 'A': partitionNumber = 1; break;
                case 'B': partitionNumber = 2; break;
                case 'C': partitionNumber = 3; break;
                case 'D': partitionNumber = 4; break;
                case 'E': partitionNumber = 5; break;
                case 'F': partitionNumber = 6; break;
                case 'G': partitionNumber = 7; break;
                case 'H': partitionNumber = 8; break;
                case 'I': partitionNumber = 9; break;
                case 'J': partitionNumber = 10; break;
                case 'K': partitionNumber = 11; break;
                case 'L': partitionNumber = 12; break;
                case 'M': partitionNumber = 13; break;
                case 'N': partitionNumber = 14; break;
                case 'O': partitionNumber = 15; break;
                case 'P': partitionNumber = 16; break;
                case 'Q': partitionNumber = 17; break;
                case 'R': partitionNumber = 18; break;
                case 'S': partitionNumber = 19; break;
                case 'T': partitionNumber = 20; break;
                case 'U': partitionNumber = 21; break;
                case 'V': partitionNumber = 22; break;
                case 'W': partitionNumber = 23; break;
                case 'X': partitionNumber = 24; break;
                case 'Y': partitionNumber = 25; break;
                case 'Z': partitionNumber = 26; break;
                default: partitionNumber = 0; break;
                }
                return partitionNumber;
        }
}
```

In the driver program, set the partitioner class as shown below:

```
job.setNumReduceTasks(27);
job.setPartitionerClass(WordCountPartitioner.class);

// Input and Output Path
FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new Path("/mapreducedemos/output/
wordcountpartitioner/"));
```

Output:

You can see 27 partitions in the below output.

Contents of directory /mapreducedemos/output/wordcountpartitioner

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00000	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00001	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00002	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00003	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00004	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00005	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00006	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00007	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00008	file	16 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00009	file	29 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00010	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00011	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00012	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00013	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00014	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00015	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00016	file	6 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00017	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00018	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00019	file	10 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00020	file	5 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00021	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00022	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00023	file	10 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00024	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00025	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00026	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup

The output file part-r-00008 is associated with alphabet **'H'**.

File: /mapreducedemos/output/wordcountpartitioner/part-r-00008

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
```

8.6 SEARCHING

Objective: To write a MapReduce program to search for a specific keyword in a file.

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

Act:

WordSearcher.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordSearcher {

        public static void main(String[] args) throws IOException,
                        InterruptedException, ClassNotFoundException {
                Configuration conf = new Configuration();
                Job job = new Job(conf);
                job.setJarByClass(WordSearcher.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                job.setMapperClass(WordSearchMapper.class);
                job.setReducerClass(WordSearchReducer.class);
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                job.setNumReduceTasks(1);
                job.getConfiguration().set("keyword", "Jack");
                FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
```

```
                FileOutputFormat.setOutputPath(job, new Path("/mapreduce/output/search"));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

WordSearchMapper.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
public class WordSearchMapper extends Mapper<LongWritable, Text, Text, Text> {
        static String keyword;
        static int pos = 0;
        protected void setup(Context context) throws IOException,
                        InterruptedException {
                Configuration configuration = context.getConfiguration();
                keyword = configuration.get("keyword");
        }
        protected void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                InputSplit i = context.getInputSplit(); // Get the input split for this map.
                FileSplit f = (FileSplit) i;
                String fileName = f.getPath().getName();
                Integer wordPos;
                pos++;
                if (value.toString().contains(keyword)) {
                        wordPos = value.find(keyword);
                        context.write(value, new Text(fileName + ","+ new IntWritable(pos).
toString() + ", "+ wordPos.toString()));
                }
        }
}
```

WordSearchReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordSearchReducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Text value, Context context)
                        throws IOException, InterruptedException {
                context.write(key, value);
        }
}
```

Output:

File: /mapreduce/output/search/part-r-00000

Goto : /mapreduce/output/search go

Go back to dir listing
Advanced view/download options

```
1002,Jack,39    student.csv,2, 5
```

8.7 SORTING

Objective: To write a MapReduce program to sort data by student name (value).

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

Act:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SortStudNames {

        public static class SortMapper extends
                        Mapper<LongWritable, Text, Text, Text> {

                protected void map(LongWritable key, Text value, Context context)
                                throws IOException, InterruptedException {

                        String[ ] token = value.toString().split(",");
                        context.write(new Text(token[1]), new Text(token[0]+" - "+token[1]));
                }
        }
    // Here, value is sorted...
        public static class SortReducer extends
                        Reducer<Text, Text, NullWritable, Text> {

                public void reduce(Text key, Iterable<Text> values, Context context)
                                throws IOException, InterruptedException {

                        for (Text details : values) {
                                context.write(NullWritable.get(), details);
                        }
                }
        }

        public static void main(String[ ] args) throws IOException,
                        InterruptedException, ClassNotFoundException {

                Configuration conf = new Configuration();
                Job job = new Job(conf);
                job.setJarByClass(SortEmpNames.class);
                job.setMapperClass(SortMapper.class);
                job.setReducerClass(SortReducer.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
                FileOutputFormat.setOutputPath(job, new
Path("/mapreduce/output/sorted/"));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }

}
```

Output:

File: /mapreduce/output/search/part-r-00000

Goto : /mapreduce/output/search go

Go back to dir listing
Advanced view/download options

```
1002,Jack,39    student.csv,2, 5
```

8.8 COMPRESSION

In MapReduce programming, you can compress the MapReduce output file. Compression provides two benefits as follows:

1. Reduces the space to store files.
2. Speeds up data transfer across the network.

You can specify compression format in the Driver Program as shown below:

```
conf.setBoolean("mapred.output.compress", true);
conf.setClass("mapred.output.compression.codec", GzipCodec.class,CompressionCodec.class);
```

Here, codec is the implementation of a compression and decompression algorithm. GzipCodec is the compression algorithm for gzip. This compresses the output file.

REMIND ME

- Mapper maps the input key–value pairs to intermediate key–value pairs.
- Reducer then reduces the set of key–value pairs that share a common key to a smaller set of values.
- The Reducer has three primary phases:
  - Shuffle and Sort
  - Reduce
  - Output Format
- Combiner and Partitioner are optimization techniques.

POINT ME (BOOK)

- MapReduce Design Patterns, O'REILLY, Donald Miner and Adam Shook.

CONNECT ME (INTERNET RESOURCES)

- http://hadooptutorial.wikispaces.com/MapReduce
- http://bigdataanalyticsnews.com/anatomy-mapreduce-job/
- http://bigdataconsultants.blogspot.in/2013/11/secondary-sort-in-hadoop-actor.html

TEST ME

A. Fill Me

1. Partitioner phase belongs to _____ task.
2. Combiner is also known as _____.
3. RecordReader converts byte-oriented view into _____ view.
4. MapReduce sorts the intermediate value based on _____.
5. In MapReduce Programming, reduce function is applied _____ group at a time.

Answers:

1. map
2. local reducer
3. record-oriented
4. keys
5. one

ASSIGNMENT FOR HANDS-ON PRACTICE

ASSIGNMENT 1

Objective: To learn about MapReduce Programming using Java.
Problem Description: Write a MapReduce Program to arrange the data on user id, then within the user id sort them in increasing order of the page count.

Input:

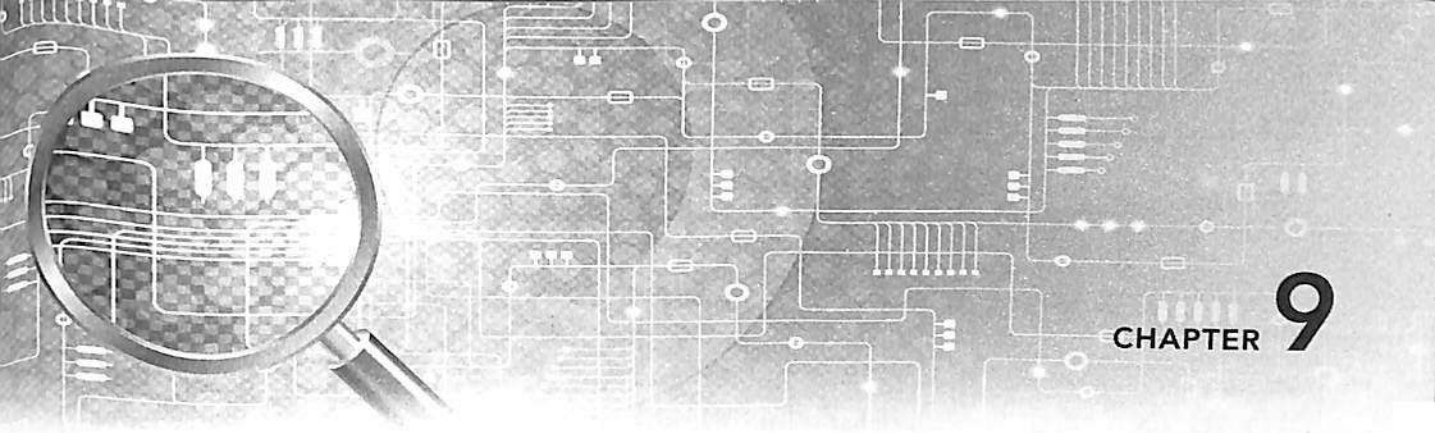
User_id	count	URL
12398	5	http://www.cbtnuggets.com/
23487	9	http://www.xda-developers.com/
34576	3	http://www.w3schools.com/
45665	6	https://www.google.co.in/
56754	4	http://www.encyclopedia.com/
67843	6	http://tutorialspoint.com/
78932	7	http://stackoverflow.com/
89021	3	http://www.wikipedia.org/
91210	2	http://www.cisce.org/results
82391	4	http://www.slideshare.net/

ASSIGNMENT 2

Objective: To learn about MapReduce Programming using Java.
Problem Description: Write a MapReduce Program to find unitwise salary.

Input:

Empno	Empname	Unit	Designation	Salary	Location
1001	John	IMST	TA	30000	Trivandrum
1002	Jack	CLOUD	PM	80000	Bangalore
1003	Joshi	FNPR	TA	35000	Trivandrum
1004	Josh	ECSSAP	PM	75000	Bangalore
1005	Jim	FSADM	SPM	60000	Bangalore
1006	Smith	ICS	TA	24000	Chandigarh
1007	Tiger	IMST	SPM	56000	Trivandrum
1008	Kate	FNPR	PM	76000	Chennai
1009	Cassy	MFGADM	TA	40000	Bangalore
1010	Ronald	ECSSAP	SPM	65000	Chennai



CHAPTER 9

Introduction to Hive

BRIEF CONTENTS

- What's in Store?
- What is Hive?
  - History of Hive and Recent Releases of Hive
  - Hive Features
  - Hive Integration and Work Flow
  - Hive Data Units
- Hive Architecture
- Hive Data Types
  - Primitive Data Types
  - Collection Data Types
- Hive File Format
  - Text File
  - Sequential File
  - RCFile (Record Columnar File)
- Hive Query Language (HQL)
  - DDL (Data Definition Language) Statements
  - DML (Data Manipulation Language) Statements
  - Starting Hive Shell
  - Database
  - Tables
  - Partitions
  - Bucketing
  - Views
  - Sub-Query
  - Joins
  - Aggregation
  - Group By and Having
- RCFILE Implementation
- SERDE
- User-Defined Function (UDF)

"Information is the oil of the 21st century, and analytics is the combustion engine."

– Peter Sondergaard, Gartner Research

WHAT'S IN STORE?

We assume that you are already familiar with commercial database systems. In this chapter, we will try to use that knowledge as our base to build a structure on Hadoop for effective analysis. We will discuss the importance of Hive with the help of use cases. We will also enrich your knowledge by working with Hive Query Language.

We suggest you refer to some of the learning resources suggested at the end of this chapter and also complete the "Test Me" exercises.

CASE STUDY: RETAIL LOG PROCESSING

About the Company

TENTOTEN is a Retail Store which has a chain of hypermarkets in India. They have 250+ stores across 95 cities and towns. About 45,000+ people are working in **TENTOTEN**. **TENTOTEN** deals in a wide range of products including fashion apparels, food products, books, furniture, etc. Around 1500+ customers visit and/or purchase products every day from each of these stores.

Problem Scenario

The approximate size of **TENTOTEN** log datasets is 12 TB. Information about the various stores is stored in the form of semi-structured data. Traditional Business Intelligence (BI) tools are good when data is present in pre-defined schema and datasets are just several hundreds of gigabytes. But the **TENTOTEN** dataset is mostly log dataset, which does not conform to any particular schema. Querying such large dataset is difficult and immensely time consuming.

The challenges are:

1. Moving the log dataset to HDFS (Hadoop Distributed File System).
2. Performing analysis on HDFS data.

Hadoop MapReduce can be used to resolve these issues. However we will still have to deal with the below constraints:

1. Writing complex MapReduce jobs in Java can be tedious and error prone.
2. Joining across large datasets is quite tricky.

Enter Hive to counter the above challenges.

9.1 WHAT IS HIVE?

Hive is a Data Warehousing tool that sits on top of Hadoop. Refer Figure 9.1. Hive is used to process structured data in Hadoop. The three main tasks performed by Apache Hive are:

1. Summarization
2. Querying
3. Analysis

Facebook initially created Hive component to manage their ever-growing volumes of log data. Later Apache software foundation developed it as open-source and it came to be known as Apache Hive.

Hive makes use of the following:

1. HDFS for Storage.
2. MapReduce for execution.
3. Stores metadata/schemas in an RDBMS.

Hive provides HQL (Hive Query Language) or HiveQL which is similar to SQL. Hive compiles SQL queries into MapReduce jobs and then runs the job in the Hadoop Cluster. It is designed to support

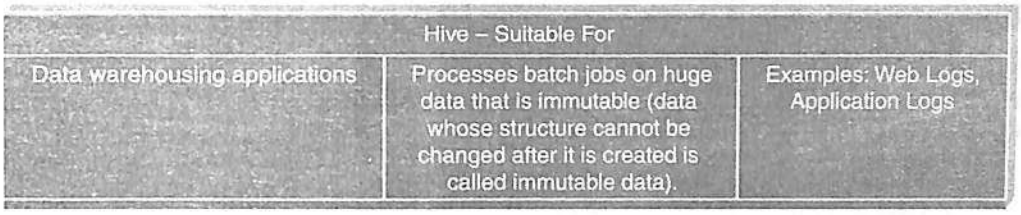


Figure 9.1 Hive – a data warehousing tool.

OLAP (Online Analytical Processing). Hive provides extensive data type functions and formats for data summarization and analysis.

Note:

1. Hive is not RDBMS.
2. It is not designed to support OLTP (Online Transaction Processing).
3. It is not designed for real-time queries.
4. It is not designed to support row-level updates.

9.1.1 History of Hive and Recent Releases of Hive

The history of Hive and recent releases of Hive are illustrated pictorially in Figures 9.2 and 9.3, respectively.

Figure 9.2 History of Hive.

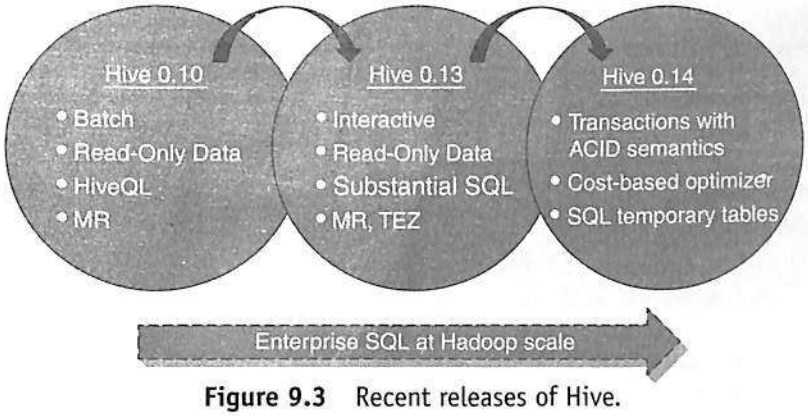


Figure 9.3 Recent releases of Hive.

9.1.2 Hive Features

1. It is similar to SQL.
2. HQL is easy to code.
3. Hive supports rich data types such as structs, lists and maps.
4. Hive supports SQL filters, group-by and order-by clauses.
5. Custom Types, Custom Functions can be defined.

9.1.3 Hive Integration and Work Flow

Figure 9.4 depicts the flow of log file analysis.

Explanation of the workflow. Hourly Log Data can be stored directly into HDFS and then data cleansing is performed on the log file. Finally, Hive table(s) can be created to query the log file.

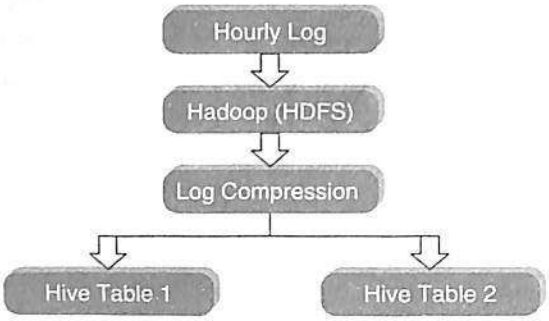


Figure 9.4 Flow of log analysis file.

9.1.4 Hive Data Units

1. **Databases:** The namespace for tables.
2. **Tables:** Set of records that have similar schema.
3. **Partitions:** Logical separations of data based on classification of given information as per specific attributes. Once hive has partitioned the data based on a specified key, it starts to assemble the records into specific folders as and when the records are inserted.
4. **Buckets (or Clusters):** Similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.

Let us take an example to understand partitioning and bucketing.

PICTURE THIS...

"XYZ Corp" has their customer base spread across 190+ countries. There are 5 million records/entities available. If it is required to fetch the entities pertaining to a particular country, in the absence of partitioning, there is no choice but to go through all of the 5 million entities. This despite the fact our query will eventually result in few thousand entities of the particular country. However, creating partitions based on country will greatly help to alleviate the performance issue by checking the data belonging to the partition for the country in question.

Partitioning tables changes how Hive structures the data storage. Hive will create subdirectories reflecting the partitioning structure like

.../customers/*country=ABC*

Although partitioning helps in enhancing performance and is recommended, having too many partitions may prove detrimental for few queries.

Bucketing is another technique of managing large datasets. If we partition the dataset based on customer_ID, we would end up with far too many partitions. Instead, if we bucket the customer table and use customer_id as the bucketing column, the value of this column will be hashed by a user-defined number

into buckets. Records with the same customer_id will always be placed in the same bucket. Assuming we have far more customer_ids than the number of buckets, each bucket will house many customer_ids. While creating the table you can specify the number of buckets that you would like your data to be distributed in using the syntax "CLUSTERED BY (customer_id) INTO XX BUCKETS"; here XX is the number of buckets.

When to Use Partitioning/Bucketing?

Bucketing works well when the field has high cardinality (cardinality is the number of values a column or field can have) and data is evenly distributed among buckets. Partitioning works best when the cardinality of the partitioning field is not too high. Partitioning can be done on multiple fields with an order (Year/Month/Day) whereas bucketing can be done on only one field.

Figure 9.5 shows how these data units are arranged in a Hive Cluster. Figure 9.6 describes the semblance of Hive structure with database.

A database contains several tables. Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as a sub-directory. Bucketed tables are stored as a file.

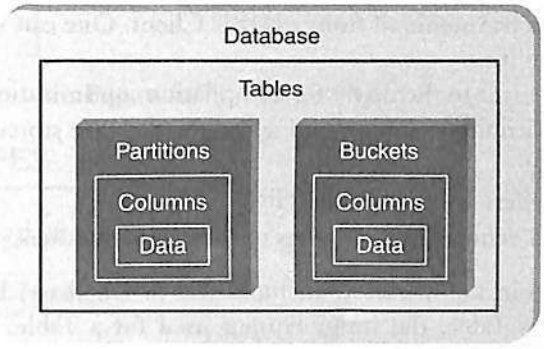


Figure 9.5 Data units as arranged in a Hive.

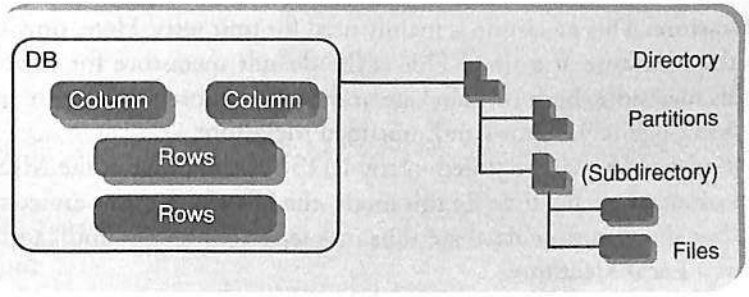


Figure 9.6 Semblance of Hive structure with database.

9.2 HIVE ARCHITECTURE

Hive Architecture is depicted in Figure 9.7. The various parts are as follows:

1. **Hive Command-Line Interface (Hive CLI):** The most commonly used interface to interact with Hive.
2. **Hive Web Interface:** It is a simple Graphic User Interface to interact with Hive and to execute query.
3. **Hive Server:** This is an optional server. This can be used to submit Hive Jobs from a remote client.

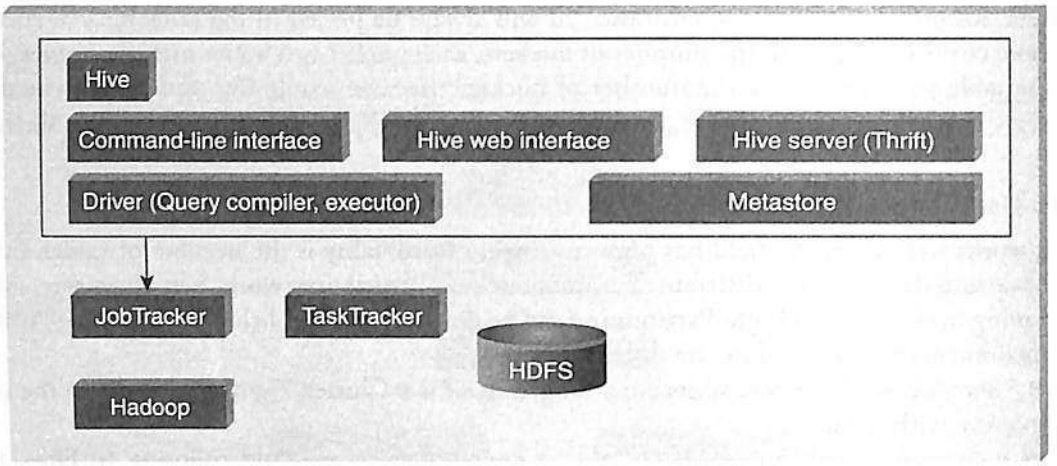


Figure 9.7 Hive architecture.

4. **JDBC/ODBC:** Jobs can be submitted from a JDBC Client. One can write a Java code to connect to Hive and submit jobs on it.
5. **Driver:** Hive queries are sent to the driver for compilation, optimization and execution.
6. **Metastore:** Hive table definitions and mappings to the data are stored in a Metastore. A Metastore consists of the following:
   - **Metastore service:** Offers interface to the Hive.
   - **Database:** Stores data definitions, mappings to the data and others.

The metadata which is stored in the metastore includes IDs of Database, IDs of Tables, IDs of Indexes, etc., the time of creation of a Table, the Input Format used for a Table, the Output Format used for a Table, etc. The metastore is updated whenever a table is created or deleted from Hive. There are three kinds of metastore.

1. **Embedded Metastore:** This metastore is mainly used for unit tests. Here, only one process is allowed to connect to the metastore at a time. This is the default metastore for Hive. It is Apache Derby Database. In this metastore, both the database and the metastore service run embedded in the main Hive Server process. Figure 9.8 shows an Embedded Metastore.
2. **Local Metastore:** Metadata can be stored in any RDBMS component like MySQL. Local metastore allows multiple connections at a time. In this mode, the Hive metastore service runs in the main Hive Server process, but the metastore database runs in a separate process, and can be on a separate host. Figure 9.9 shows a Local Metastore.
3. **Remote Metastore:** In this, the Hive driver and the metastore interface run on different JVMs (which can run on different machines as well) as in Figure 9.10. This way the database can be fire-walled from the Hive user and also database credentials are completely isolated from the users of Hive.

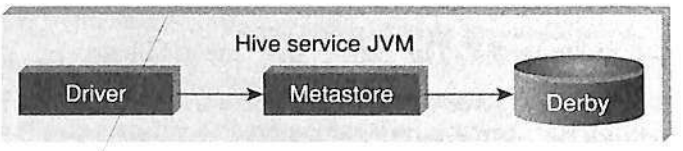


Figure 9.8 Embedded Metastore.

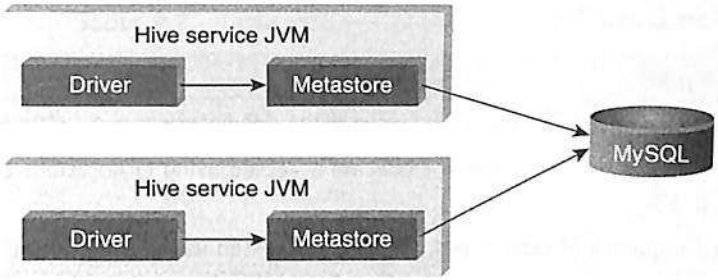


Figure 9.9 Local Metastore.

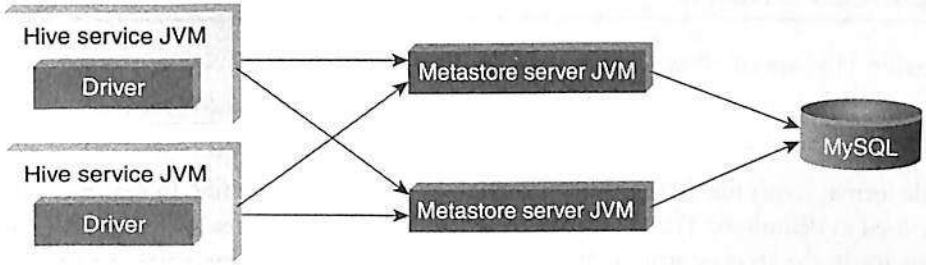


Figure 9.10 Remote Metastore.

9.3 HIVE DATA TYPES

9.3.1 Primitive Data Types

Numeric Data Type	
TINYINT	1-byte signed integer
SMALLINT	2-byte signed integer
INT	4-byte signed integer
BIGINT	8-byte signed integer
FLOAT	4-byte single-precision floating-point
DOUBLE	8-byte double-precision floating-point number
String Types	
STRING	
VARCHAR	Only available starting with Hive 0.12.0
CHAR	Only available starting with Hive 0.13.0
Strings can be expressed in either single quotes (') or double quotes (")	
Miscellaneous Types	
BOOLEAN	
BINARY	Only available starting with Hive

9.3.2 Collection Data Types

Collection Data Types	
STRUCT	Similar to 'C' struct. Fields are accessed using dot notation. E.g.: struct('John', 'Doe')
MAP	A collection of key–value pairs. Fields are accessed using [] notation. E.g.: map('first', 'John', 'last', 'Doe')
ARRAY	Ordered sequence of same types. Fields are accessed using array index. E.g.: array('John', 'Doe')

9.4 HIVE FILE FORMAT

The file formats in Hive specify how records are encoded in a file.

9.4.1 Text File

The default file format is text file. In this format, each record is a line in the file. In text file, different control characters are used as delimiters. The delimiters are ^A (octal 001, separates all fields), ^B (octal 002, separates the elements in the array or struct), ^C (octal 003, separates key–value pair), and \n. The term **field** is used when overriding the default delimiter. The supported text files are CSV and TSV. JSON or XML documents too can be specified as text file.

9.4.2 Sequential File

Sequential files are flat files that store binary key–value pairs. It includes compression support which reduces the CPU, I/O requirement.

9.4.3 RCFile (Record Columnar File)

RCFile stores the data in **Column Oriented Manner** which ensures that **Aggregation** operation is not an expensive operation. For example, consider a table which contains four columns as shown in Table 9.1.

Instead of only partitioning the table horizontally like the row-oriented DBMS (row-store), RCFile partitions this table first horizontally and then vertically to serialize the data. Based on the user-specified value, first the table is partitioned into multiple row groups horizontally. Depicted in Table 9.2, Table 9.1 is partitioned into two row groups by considering three rows as the size of each row group.

Next, in every row group RCFile partitions the data vertically like column-store. So the table will be serialized as shown in Table 9.3.

Table 9.1 A table with four columns

C1	C2	C3	C4
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

Table 9.2 Table with two row groups

Row Group 1				Row Group 2			
C1	C2	C3	C4	C1	C2	C3	C4
11	12	13	14	41	42	43	44
21	22	23	24	51	52	53	54
31	32	33	34				

Table 9.3 Table in RCFile Format

Row Group 1	Row Group 2
11, 21, 31;	41, 51;
12, 22, 32;	42, 52;
13, 23, 33;	43, 53;
14, 24, 34;	44, 54;

9.5 HIVE QUERY LANGUAGE (HQL)

Hive query language provides basic SQL like operations. Here are few of the tasks which HQL can do easily.

1. Create and manage tables and partitions.
2. Support various Relational, Arithmetic, and Logical Operators.
3. Evaluate functions.
4. Download the contents of a table to a local directory or result of queries to HDFS directory.

9.5.1 DDL (Data Definition Language) Statements

These statements are used to build and modify the tables and other objects in the database. The DDL commands are as follows:

1. Create/Drop/Alter Database
2. Create/Drop/Truncate Table
3. Alter Table/Partition/Column
4. Create/Drop/Alter View
5. Create/Drop/Alter Index
6. Show
7. Describe

9.5.2 DML (Data Manipulation Language) Statements

These statements are used to retrieve, store, modify, delete, and update data in database. The DML commands are as follows:

1. Loading files into table.
2. Inserting data into Hive Tables from queries.

Note: Hive 0.14 supports update, delete, and transaction operations.

9.5.3 Starting Hive Shell

To start Hive, go to the installation path of Hive and type as below:

```
[root@volgalnx005 ~]# hive

Logging initialized using configuration in jar:file:/root/Desktop/VMDATA/Hive/hi
ve/lib/hive-common-0.14.0.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/root/Desktop/VMDATA/Hadoop/hadoop/share/hadoo
p/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/root/Desktop/VMDATA/Hive/hive/lib/hive-jdbc-0
.14.0-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>
```

The sections have been designed as follows:

Objective:	What is it that we are trying to achieve here?
Input (optional):	What is the input that has been given to us to act upon?
Act:	The actual statement/command to accomplish the task at hand.
Outcome:	The result/output as a consequence of executing the statement.

9.5.4 Database

A database is like a container for data. It has a collection of tables which houses the data.

Objective: To create a database named "STUDENTS" with comments and database properties.

Act:

```
CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT 'STUDENT Details'
WITH DBPROPERTIES ('creator' = 'JOHN');
```

Outcome:

```
hive> CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT 'STUDENT Details' WITH DBPROPERTIES ('creato
r' = 'JOHN');
OK
Time taken: 0.536 seconds
hive>
```

Explanation of the syntax:

IF NOT EXIST: It is an optional clause. The create database statement with "IF Not EXISTS" clause creates a database if it does not exist. However, if the database already exists then it will notify the user that a database with the same name already exists and will not show any error message.

COMMENT: This is to provide short description about the database.

WITH DBPROPERTIES: It is an optional clause. It is used to specify any properties of database in the form of (key, value) separated pairs. In the above example, "Creator" is the "Key" and "JOHN" is the value.

We can use "**SCHEMA**" in place of "**DATABASE**" in this command.

Note: We have not specified the location where the Hive database will be created. By default all the Hive databases will be created under default warehouse directory (set by the property hive.metastore.warehouse.dir) as /user/hive/warehouse/database_name.db. But if we want to specify our own location, then the LOCATION clause can be specified. This clause is optional.

Objective: To display a list of all databases.

Act:

```
SHOW DATABASES;
```

Outcome:

```
hive> SHOW DATABASES;
OK
students
Time taken: 0.082 seconds, Fetched: 22 row(s)
hive>
```

By default, **SHOW DATABASES** lists all the databases available in the metastore. We can use "**SCHEMAS**" in place of "**DATABASES**" in this command. The command has an optional "Like" clause. It can be used to filter the database names using regular expressions such as "*" , "?", etc.

SHOW DATABASES LIKE "Stu*"

SHOW DATABASES like "Stud???s"

Objective: To describe a database.

Act:

```
DESCRIBE DATABASE STUDENTS;
```

Note: Shows only DB name, comment, and DB directory.

Outcome:

```
hive> DESCRIBE DATABASE STUDENTS;
OK
students        STUDENT Details hdfs://volgalnx010.ad.infosys.com:9000/user/hive/warehouse/studen
ts.db   root    USER
Time taken: 0.03 seconds, Fetched: 1 row(s)
hive>
```

Objective: To describe the extended database.

Act:

```
DESCRIBE DATABASE EXTENDED STUDENTS;
```

Note: Shows DB properties ...

Outcome:

```
hive> DESCRIBE DATABASE EXTENDED STUDENTS;
OK
students        STUDENT Details hdfs://volgalnx010.ad.infosys.com:9000/user/hive/warehouse/studen
ts.db   root    USER    {creator=JOHN}
Time taken: 0.027 seconds, Fetched: 1 row(s)
hive>
```

DESCRIBE DATABASE EXTENDED shows database's properties given under DBPROPERTIES argument at the time of creation.

We can use "**SCHEMA**" in place of "**DATABASE**", "**DESC**" in place of "**DESCRIBE**" in this command.

Objective: To alter the database properties.

Act:

```
ALTER DATABASE STUDENTS SET DBPROPERTIES ('edited-by' = 'JAMES');
```

Note: We can use the "ALTER DATABASE" command to

- Assign any new (key, value) pairs into DBPROPERTIES.
- Set owner user or role to the Database.

In Hive, it is not possible to unset the DB properties.

Outcome:

```
hive> ALTER DATABASE STUDENTS SET DBPROPERTIES ('edited-by' = 'JAMES');
OK
Time taken: 0.086 seconds
hive>
```

In the above example, the ALTER DATABASE command is used to assign new ('edited-by' = 'JAMES') pair into DBPROPERTIES. This can be verified by using the 'describe *extended*'.

Hive> DESCRIBE DATABASE Student EXTENDED

Objective: To make the database as current working database.

Act:

```
USE STUDENTS;
```

Outcome:

```
hive> USE STUDENTS;
OK
Time taken: 0.02 seconds
hive>
```

There is no command to show the current database, but use the below command statement to keep printing the current database name as suffix in the command line prompt.

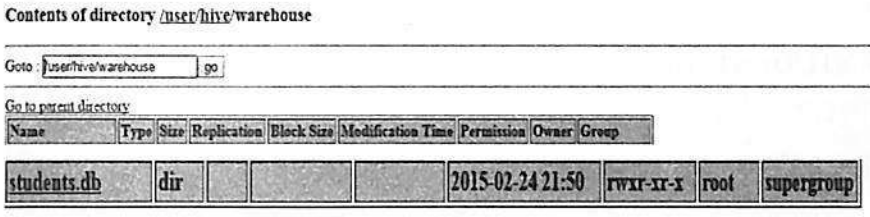
set hive.cli.print.current.db=true;

Objective: To drop database.

Act:

```
DROP DATABASE STUDENTS;
```

Note: Hive creates database in the warehouse directory of Hive as shown below:


Contents of directory /user/hive/warehouse

Goto : /user/hive/warehouse go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
students.db	dir				2015-02-24 21:50	rwxr-xr-x	root	supergroup

Now assume that the database "STUDENTS" has 10 tables within it. How do we delete the complete database along with the tables contained therein?

Use the command:

DROP DATABASE STUDENTS CASCADE;

By default the mode is RESTRICT which implies that the database will NOT be dropped if it contains tables.

Note: The complete syntax is as follows:

DROP DATABASE [IF EXISTS] database_name [RESTRICT | CASCADE]

9.5.5 Tables

Hive provides two kinds of table:

1. Internal or Managed Table
2. External Table

9.5.5.1 Managed Table

1. Hive stores the Managed tables under the warehouse folder under Hive.
2. The complete life cycle of table and data is managed by Hive.
3. When the internal table is dropped, it drops the data as well as the metadata.

When you create a table in Hive, by default it is internal or managed table. If one needs to create an external table, one will have to use the keyword "EXTERNAL".

Objective: To create managed table named 'STUDENT'.

Act:

```
CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW
FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT DELIMITED F
IELDS TERMINATED BY '\t';
OK
Time taken: 0.355 seconds
hive>
```

Objective: To describe the "STUDENT" table.

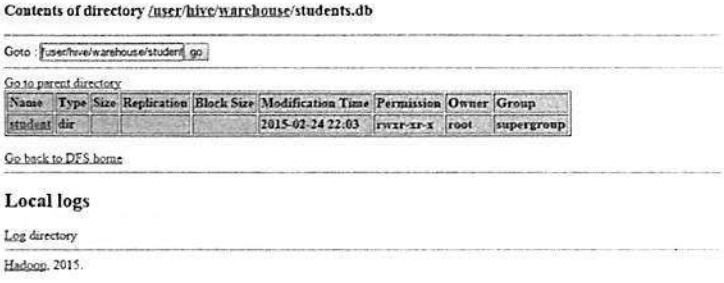
Act:

```
DESCRIBE STUDENT;
```

Outcome:

```
hive> DESCRIBE STUDENT;
OK
rollno                  int
name                    string
gpa                     float
Time taken: 0.163 seconds, Fetched: 3 row(s)
hive>
```

Note: Hive creates managed table in the warehouse directory of Hive as shown below:


Contents of directory /user/hive/warehouse/students.db

Goto : /user/hive/warehouse/student go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
student	dir				2015-02-24 22:03	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2015.

To check whether an existing table is managed or external, use the below syntax:

DESCRIBE FORMATTED tablename;

It displays complete metadata of a table. You will see one row called table type which will display either MANAGED_TABLE OR EXTERNAL_TABLE.

DESCRIBE FORMATTED STUDENT;

9.5.5.2 External or Self-Managed Table

1. When the table is dropped, it retains the data in the underlying location.
2. **External** keyword is used to create an external table.
3. **Location** needs to be specified to store the dataset in that particular location.

Objective: To create external table named 'EXT_STUDENT'.

Act:

```
CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/STUDENT_INFO;
```

Outcome:

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMA
T DELIMITED FIELDS TERMINATED BY '\t'  LOCATION '/STUDENT_INFO';
OK
Time taken: 0.123 seconds
hive>
```

Note: Hive creates the external table in the specified location.

9.5.5.3 *Loading Data into Table from File*

Objective: To load data into the table from file named student.tsv.

Act:

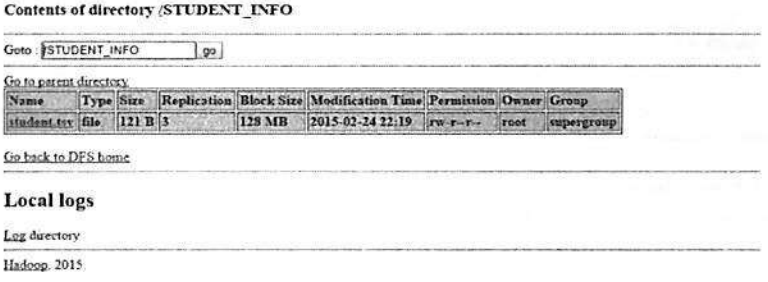
```
LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE EXT_STUDENT;
```

Note: Local keyword is used to load the data from the local file system. In this case, the file is copied. To load the data from HDFS, remove local key word from the statement. In this case, the file is moved from the original location.

Outcome:

```
hive> LOAD DATA LOCAL INPATH  '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE EXT_STUDENT;
Loading data to table students.ext_student
Table students.ext_student stats: [numFiles=0, numRows=0, totalSize=0, rawDataSize=0]
OK
Time taken: 5.034 seconds
hive>
```

Hive loads the file in the specified location as shown below:



Contents of directory /STUDENT_INFO

Goto : /STUDENT_INFO go

Go to parent directory

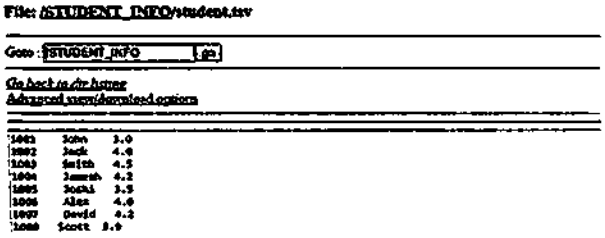
Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
student.tsv	file	121 B	3	128 MB	2015-02-24 22:19	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2015.



Let us understand the difference between INTO TABLE and OVERWRITE TABLE with an example:

Assume the "EXT_STUDENT" table already had 100 records and the "student.tsv" file has 10 records. After issuing the LOAD DATA statement with the INTO TABLE clause, the table "EXT_STUDENT" will contain 110 records; however, the same LOAD DATA statement with the OVERWRITE clause will wipe out all the former content from the table and then load the 10 records from the data file.

9.5.5.4 Collection Data Types

Objective: To work with collection data types.

Input:

1001,John,Smith:Jones,Mark1!45:Mark2!46:Mark3!43
1002,Jack,Smith:Jones,Mark1!46:Mark2!47:Mark3!42

Act:

```
CREATE TABLE STUDENT_INFO (rollno INT,name String, sub ARRAY<STRING>,marks
MAP<STRING,INT>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY ':'
MAP KEYS TERMINATED BY '!';
LOAD DATA LOCAL INPATH '/root/hivedemos/studentinfo.csv' INTO TABLE
STUDENT_INFO;
```

Outcome:

```
hive> CREATE TABLE STUDENT_INFO (rollno INT,name String, sub ARRAY<STRING>,marks MAP<STRING,FLOAT>)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > COLLECTION ITEMS TERMINATED BY ':'
    > MAP KEYS TERMINATED BY '!';
OK
Time taken: 0.112 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/studentinfo.csv' INTO TABLE STUDENT_INFO;
Loading data to table students.student_info
Table students.student_info stats: [numFiles=1, totalSize=109]
OK
Time taken: 0.397 seconds
hive>
```

9.5.5.5 Querying Table

Objective: To retrieve the student details from "EXT_STUDENT" table.

Act:

```
SELECT * from EXT_STUDENT;
```

Outcome:

```
hive> select * from EXT_STUDENT;
OK
1001    John    3.0
1002    Jack    4.0
1003    Smith   4.5
1004    Scott   4.2
1005    Joshi   3.5
1006    Alex    4.5
1007    David   4.2
1008    James   4.0
1009    John    3.0
1010    Joshi   3.5
Time taken: 0.054 seconds, Fetched: 10 row(s)
hive>
```

Objective: Querying Collection Data Types.

Act:

```
SELECT * from STUDENT_INFO;
SELECT NAME,SUB FROM STUDENT_INFO;
// To retrieve value of Mark1
SELECT NAME, MARKS['Mark1'] from STUDENT_INFO;
// To retrieve subordinate (array) value
SELECT NAME,SUB[0] FROM STUDENT_INFO;
```

Outcome:

```
hive> SELECT * from STUDENT_INFO;
OK
1001    John    ["Smith","Jones"]       {"Mark1":45,"Mark2":46,"Mark3":43}
1002    Jack    ["Smith","Jones"]       {"Mark1":46,"Mark2":47,"Mark3":42}
Time taken: 0.044 seconds, Fetched: 2 row(s)
```

```
hive> SELECT NAME,SUB FROM STUDENT_INFO;
OK
John    ["Smith","Jones"]
Jack    ["Smith","Jones"]
Time taken: 0.061 seconds, Fetched: 2 row(s)
hive>
```

```
hive> SELECT NAME, MARKS['Mark1']  from STUDENT_INFO;
OK
John    45
Jack    46
Time taken: 0.06 seconds, Fetched: 2 row(s)
hive>
```

```
hive> SELECT NAME,SUB[0] FROM STUDENT_INFO;
OK
John    Smith
Jack    Smith
Time taken: 0.071 seconds, Fetched: 2 row(s)
hive>
```

9.5.6 Partitions

In Hive, the query reads the entire dataset even though a where clause filter is specified on a particular column. This becomes a bottleneck in most of the MapReduce jobs as it involves huge degree of I/O. So it is necessary to reduce I/O required by the MapReduce job to improve the performance of the query. A very common method to reduce I/O is data partitioning.

Partitions split the larger dataset into more meaningful chunks.

We will try to understand partitioning with the help of a simple example.

PICTURE THIS...

"XYZ enterprise" has a wide customer base spread across several states in the US. The data has been fed to the central system. The senior leadership team would like to get a report providing the statewise Sales %.

The IT team has proposed two options to help service this request:

1. Run the query with the where clause of each state name (such as where StateName = "A"). This will mean that the entire dataset is scanned/read through for each and every state. If we have data for 25 states, the dataset is read 25 times (once for each state). Assuming we have 5 million records, it would mean that 5 million records are read 25 times. This can be a major performance bottle neck and will worsen as the dataset grows.
2. The second option stated here can help alleviate this problem. The IT team can start off with creating 25 folders (one for each state) and instruct to have the data pertaining to states place into the folder of the respective states. This arrangement will greatly help at the time of querying the data. To get the Sales % of a particular state, the folder belonging to that state only has to be scanned/read through.

This intelligent way of grouping data during data load is termed as PARTITIONING in Hive.

This brings us to the next question.

If you are looking through the folder for state = "A", is there any possibility of you coming across data for state = "B" or state = "C"? Think through! I am sure your answer will be No! And we know the reason.

A point to note here is that as we create the partitions, there is no need to include the partitioned column along with the other columns of the dataset as this is something that is automatically taken care of "BY PARTITIONED" clause.

In our example above, we will refrain from adding the partitioned column along with other columns of the dataset and trust Hive to automatically manage this.

Partition is of two types:

1. **STATIC PARTITION:** It is upon the user to mention the partition (the segregation unit) where the data from the file is to be loaded.
2. **DYNAMIC PARTITION:** The user is required to simply state the column, basis which the partitioning will take place. Hive will then create partitions basis the unique values in the column on which partition is to be carried out.

Points to consider as you create partitions:

1. STATIC PARTITIONING implies that the user controls everything from defining the PARTITION column to loading data into the various partitioned folders.
2. As in our example above, if STATIC partition is done over the STATE column and assume by mistake the data for state "B" is placed inside the partition for state "A", our query for data for state "B" is

bound to return zilch records. The reason is obvious. A Select fired on STATIC partition just takes into consideration the partition name, and does not consider the data held inside the partition.

3. DYNAMIC PARTITIONING means Hive will intelligently get the distinct values for partitioned column and segregate data into respective partitions. There is no manual intervention.

By default, dynamic partitioning is enabled in Hive. Also by default it is strictly implying that one is required to do one level of STATIC partitioning before Hive can perform DYNAMIC partitioning inside this STATIC segregation unit.

In order to go with full dynamic partitioning, we have to set below property to non-strict in Hive.

```
hive> set hive.exec.dynamic.partition.mode=nonstrict
```

9.5.6.1 Static Partition

Static partitions comprise columns whose values are known at compile time.

Objective: To create static partition based on "gpa" column.

Act:

```
CREATE TABLE IF NOT EXISTS STATIC_PART_STUDENT (rollno INT, name STRING)
PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED
BY '\t';
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STATIC_PART_STUDENT(rollno INT,name STRING) PARTITIONED BY (gpa FLOAT) RO
W FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.105 seconds
hive>
```

Objective: Load data into partition table from table.

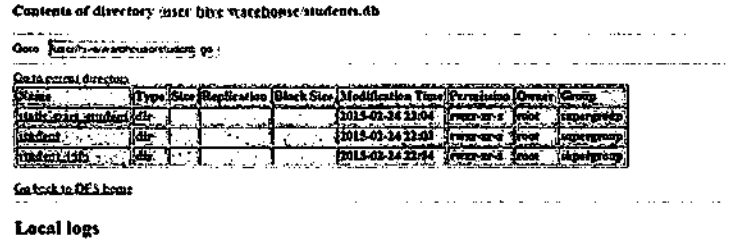
Act:

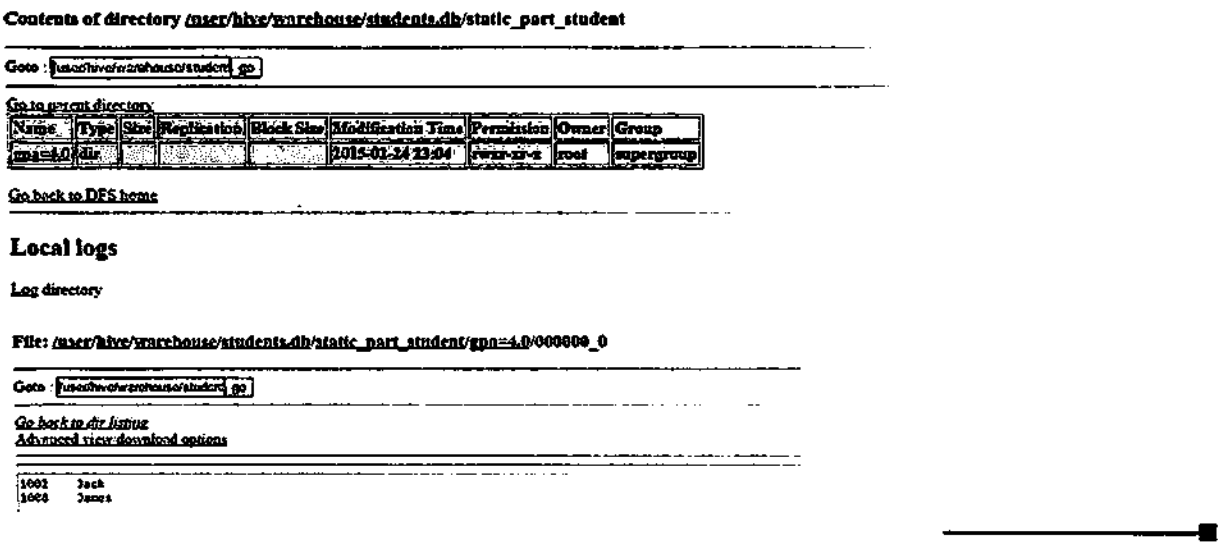
```
INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0)
SELECT rollno, name from EXT_STUDENT where gpa=4.0;
```

Outcome:

```
hive> INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0) SELECT rollno,name from EXT_STUDENT
where gpa=4.0;
Query ID = root_20150224230404_4500d58a-cb21-4912-ba40-788e5cf8f9da
Total jobs = 1
```

Hive creates the folder for the value specified in the partition.



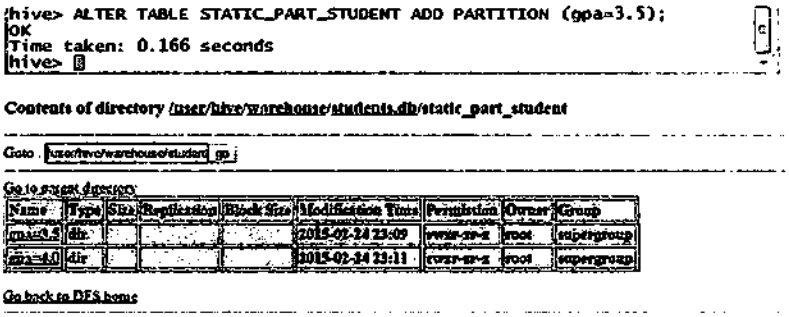


Objective: To add one more static partition based on "gpa" column using the "alter" statement.

Act:

```
ALTER TABLE STATIC_PART_STUDENT ADD PARTITION (gpa=3.5);
INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0) SELECT
rollno,name from EXT_STUDENT where gpa=4.0;
```

Outcome:



9.5.6.2 *Dynamic Partition*

Dynamic partition have columns whose values are known only at Execution Time.

Objective: To create dynamic partition on column date.

Act:

```
CREATE TABLE IF NOT EXISTS DYNAMIC_PART_STUDENT(rollno INT,name STRING)
PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED
BY '\t';
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS DYNAMIC_PART_STUDENT(rollno INT,name STRING) PARTITIONED BY (gpa FLOAT) R
OW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.166 seconds
hive>
```

Objective: To load data into a dynamic partition table from table.

Act:

SET hive.exec.dynamic.partition = true;

SET hive.exec.dynamic.partition.mode = nonstrict;

Note: The dynamic partition strict mode requires at least one static partition column. To turn this off, set hive.exec.dynamic.partition.mode=nonstrict

INSERT OVERWRITE TABLE DYNAMIC_PART_STUDENT PARTITION (gpa) SELECT rollno,name,gpa from EXT_STUDENT;

Outcome:

Contents of directory /user/hive/warehouse/students.db/dynamic_part_student

Goto [/user/hive/warehouse/st.udent] go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
gpa=2.0	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=3.5	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=4.0	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=4.3	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=4.5	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup

Go back to DFS home

Note: Create partition for all values.

9.5.7 Bucketing

Bucketing is similar to partition. However, there is a subtle difference between partition and bucketing. In a partition, you need to create partition for each unique value of the column. This may lead to situations where you may end up with thousands of partitions. This can be avoided by using Bucketing in which you can limit the number of buckets that will be created. A bucket is a file whereas a partition is a directory.

Objective: To learn the concept of bucket in hive.

Act:

CREATE TABLE IF NOT EXISTS STUDENT (rollno INT,name STRING,grade FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' INTO TABLE STUDENT;

Set below property to enable bucketing.

set hive.enforce.bucketing=true;

```
// To create a bucketed table having 3 buckets
CREATE TABLE IF NOT EXISTS STUDENT_BUCKET (rollno INT,name STRING,grade
FLOAT)
CLUSTERED BY (grade) into 3 buckets;
// Load data to bucketed table
FROM STUDENT
INSERT OVERWRITE TABLE STUDENT_BUCKET
SELECT rollno,name,grade;
// To display content of first bucket
SELECT DISTINCT GRADE FROM STUDENT_BUCKET
TABLESAMPLE(BUCKET 1 OUT OF 3 ON GRADE);
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STUDENT (rollno INT,name STRING,grade FLOAT)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.101 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' INTO TABLE STUDENT;
Loading data to table book.student
Table book.student stats: [numFiles=1, totalSize=145]
OK
Time taken: 0.536 seconds
hive>
```

```
hive> set hive.enforce.bucketing=true;
hive>
```

```
hive> CREATE TABLE IF NOT EXISTS STUDENT_BUCKET (rollno INT,name STRING,grade FLOAT)
    > CLUSTERED BY (grade) into 3 buckets;
OK
Time taken: 0.101 seconds
hive>
```

```
hive> FROM STUDENT
    > INSERT OVERWRITE TABLE STUDENT_BUCKET
    > SELECT rollno,name,grade;
```

3 buckets have been created as shown below:

Contents of directory /user/hive/warehouse/book.db/student_bucket

Goto /user/hive/warehouse/book.db/ go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
000000_0	file	59 B	3	128 MB	2015-03-10 22:29	rw-r--r--	root	supergroup
000001_0	file	59 B	3	128 MB	2015-03-10 22:29	rw-r--r--	root	supergroup
000002_0	file	28 B	3	128 MB	2015-03-10 22:29	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2015

```
hive>
    > SELECT DISTINCT GRADE FROM STUDENT_BUCKET
    > TABLESAMPLE(BUCKET 1 OUT OF 3 ON GRADE);
OK
4.0
4.2
Time taken: 21.117 seconds, Fetched: 2 row(s)
hive>
```

9.5.8 Views

In Hive, view support is available only in version starting from 0.6. Views are purely logical object.

Objective: To create a view table named "STUDENT_VIEW".

Act:

```
CREATE VIEW STUDENT_VIEW AS SELECT rollno, name FROM EXT_STUDENT;
```

Outcome:

```
hive> CREATE VIEW STUDENT_VIEW AS SELECT rollno,name FROM EXT_STUDENT;
OK
Time taken: 0.606 seconds
hive>
```

Objective: Querying the view "STUDENT_VIEW".

Act:

```
SELECT * FROM STUDENT_VIEW LIMIT 4;
```

Outcome:

```
hive> SELECT * FROM STUDENT_VIEW LIMIT 4;
OK
1001    John
1002    Jack
1003    Smith
1004    Scott
Time taken: 0.279 seconds, Fetched: 4 row(s)
hive>
```

Objective: To drop the view "STUDENT_VIEW".

Act:

```
DROP VIEW STUDENT_VIEW;
```

Outcome:

```
hive> DROP VIEW STUDENT_VIEW;
OK
Time taken: 0.452 seconds
hive>
```

9.5.9 Sub-Query

In Hive, sub-queries are supported only in the FROM clause (Hive 0.12). You need to specify name for sub-query because every table in a FROM clause has a name. The columns in the sub-query select list should have unique names. The columns in the subquery select list are available to the outer query just like columns of a table.

Objective: Write a sub-query to count occurrence of similar words in the file.

Act:

```
CREATE TABLE docs (line STRING);
LOAD DATA LOCAL INPATH '/root/hivedemos/lines.txt' OVERWRITE INTO TABLE docs;
CREATE TABLE word_count AS
SELECT word, count(1) AS count FROM
(SELECT explode (split (line, ' ')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
SELECT * FROM word_count;
```

Outcome:

```
hive> CREATE TABLE docs (line STRING);
OK
Time taken: 0.118 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/lines.txt' OVERWRITE INTO TABLE docs;
Loading data to table students.docs
Table students.docs stats: [numFiles=1, numRows=0, totalSize=91, rawDataSize=0]
OK
Time taken: 2.697 seconds
hive>
```

```
hive> CREATE TABLE word_count AS
    > SELECT word, count(1) AS count FROM
    > (SELECT explode (split (line, ' ')) AS word FROM docs) w
    > GROUP BY word
    > ORDER BY word;

hive> SELECT * FROM word_count;
OK
Hadoop  2
Hive    2
Introducing     1
Introduction    1
Pig     1
Session 3
Welcome 1
to      2
Time taken: 0.062 seconds, Fetched: 8 row(s)
hive>
```

Note: The explode() function takes an array as input and outputs the elements of the array as separate rows.

In Hive 0.13, sub-queries are supported in the where clause as well.

9.5.10 Joins

Joins in Hive is similar to the SQL Join.

Objective: To create JOIN between Student and Department tables where we use RollNo from both the tables as the join key.

Act:

```
CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW
FORMAT DELIMITED FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE
STUDENT;

CREATE TABLE IF NOT EXISTS DEPARTMENT(rollno INT,deptno int,name STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/root/hivedemos/department.tsv' OVERWRITE INTO
TABLE DEPARTMENT;

SELECT a.rollno, a.name, a.gpa, b.deptno FROM STUDENT a JOIN DEPARTMENT b ON
a.rollno = b.rollno;
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT D
ELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.115 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE STUDENT
;
Loading data to table students.student
Table students.student stats: [numFiles=1, numRows=0, totalSize=145, rawDataSize=0]
OK
Time taken: 0.723 seconds
hive>
```

```
hive> CREATE TABLE IF NOT EXISTS DEPARTMENT(rollno INT,deptno int,name STRING) ROW FORM
AT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.099 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/department.tsv' OVERWRITE INTO TABLE DEPA
RTMENT;
Loading data to table students.department
Table students.department stats: [numFiles=1, numRows=0, totalSize=120, rawDataSize=0]
OK
Time taken: 0.442 seconds
hive>
```

```
hive> SELECT a.rollno, a.name, a.gpa, b.deptno  FROM STUDENT  a JOIN DEPARTMENT b ON a.
rollno = b.rollno;
OK
1001    John    3.0     101
1002    Jack    4.0     102
1003    Smith   4.5     103
1004    Scott   4.2     104
1005    Joshi   3.5     105
1006    Alex    4.5     101
1007    David   4.2     104
1008    James   4.0     102
Time taken: 115.282 seconds, Fetched: 8 row(s)
hive>
```

9.5.11 Aggregation

Hive supports aggregation functions like avg, count, etc.

Objective: To write the average and count aggregation functions.

Act:

```
SELECT avg(gpa) FROM STUDENT;
SELECT count(*) FROM STUDENT;
```

Outcome:

```
hive> SELECT avg(gpa) FROM STUDENT;

OK
3.839999961853027
Time taken: 28.639 seconds, Fetched: 1 row(s)
hive>

hive> SELECT count(*) FROM STUDENT;

OK
10
Time taken: 26.218 seconds, Fetched: 1 row(s)
hive>
```

9.5.12 Group By and Having

Data in a column or columns can be grouped on the basis of values contained therein by using "Group By". "Having" clause is used to filter out groups NOT meeting the specified condition.

Objective: To write group by and having function.

Act:

```
SELECT rollno, name,gpa FROM STUDENT GROUP BY rollno,name,gpa HAVING gpa >
4.0;
```

Outcome:

```
1003    Smith   4.5
1004    Scott   4.2
1006    Alex    4.5
1007    David   4.2
Time taken: 78.972 seconds, Fetched: 4 row(s)
hive>
```

9.6 RCFILE IMPLEMENTATION

RCFile (Record Columnar File) is a data placement structure that determines how to store relational tables on computer clusters.

Objective: To work with RCFILE Format.

Act:

```
CREATE TABLE STUDENT_RC( rollno int, name string,gpa float ) STORED AS RCFILE;
INSERT OVERWRITE table STUDENT_RC SELECT * FROM STUDENT;
SELECT SUM(gpa) FROM STUDENT_RC;
```

Outcome:

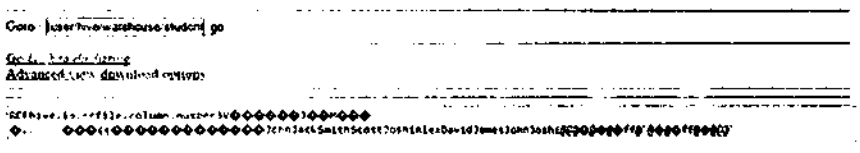
```
hive> CREATE TABLE STUDENT_RC( rollno int, name string,gpa float ) STORED AS RCFILE;
OK
Time taken: 0.093 seconds
hive>
```

```
hive> INSERT OVERWRITE table STUDENT_RC SELECT * from STUDENT;

hive> SELECT SUM(gpa) from STUDENT_RC;

OK
38.39999961853027
Time taken: 25.41 seconds, Fetched: 1 row(s)
hive>
```

Note: Stores the data in column oriented manner.



9.7 SERDE

SerDe stands for Serializer/Deserializer.

1. Contains the logic to convert unstructured data into records.
2. Implemented using Java.
3. Serializers are used at the time of writing.
4. Deserializers are used at query time (SELECT Statement).

Deserializer interface takes a binary representation or string of a record, converts it into a java object that Hive can then manipulate. Serializer takes a java object that Hive has been working with and translates it into something that Hive can write to HDFS.

Objective: To manipulate the XML data.

Input:

<employee> <empid>1001</empid> <name>John</name> <designation>Team Lead</designation> </employee>
<employee> <empid>1002</empid> <name>Smith</name> <designation>Analyst</designation> </employee>

Act:

```
CREATE TABLE XMLSAMPLE(xmldata string);
LOAD DATA LOCAL INPATH '/root/hivedemos/input.xml' INTO TABLE XMLSAMPLE;

CREATE TABLE xpath_table AS
SELECT xpath_int(xmldata,'employee/empid'),
xpath_string(xmldata,'employee/name'),
xpath_string(xmldata,'employee/designation')
FROM xmlsample;

SELECT * FROM xpath_table;
```

Outcome:

```
hive> CREATE TABLE XMLSAMPLE(xmldata string);
OK
Time taken: 0.244 seconds
hive>

hive> LOAD DATA LOCAL INPATH '/root/hivedemos/input.xml' INTO TABLE XMLSAMPLE;
Loading data to table students.xmlsample
Table students.xmlsample stats: [numFiles=1, totalSize=194]
OK
Time taken: 0.889 seconds
hive>

hive> CREATE TABLE xpath_table AS
    > SELECT xpath_int(xmldata,'employee/empid'),
    > xpath_string(xmldata,'employee/name'),
    > xpath_string(xmldata,'employee/designation')
    > FROM xmlsample;

hive> SELECT * FROM xpath_table;
OK
1001    John    Team Lead
1002    Smith   Analyst
Time taken: 0.064 seconds, Fetched: 2 row(s)
hive>
```

9.8 USER-DEFINED FUNCTION (UDF)

In Hive, you can use custom functions by defining the User-Defined Function (UDF).

Objective: Write a Hive function to convert the values of a field to uppercase.

Act:

```
package com.example.hive.udf;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
@Description(
 name="SimpleUDFExample")
```

```
public final class MyLowerCase extends UDF {
  public String evaluate(final String word) {
   return word.toLowerCase();
  }
}
```

Note: Convert this Java Program into Jar.

```
ADD JAR /root/hivedemos/UpperCase.jar;
CREATE TEMPORARY FUNCTION touppercase AS 'com.example.hive.udf.MyUpperCase';
SELECT TOUPPERCASE(name) FROM STUDENT;
```

Outcome:

```
hive> ADD JAR /root/hivedemos/UpperCase.jar;
Added [/root/hivedemos/UpperCase.jar] to class path
Added resources: [/root/hivedemos/UpperCase.jar]
hive> CREATE TEMPORARY FUNCTION touppercase AS 'com.example.hive.udf.MyUpperCase';
OK
Time taken: 0.014 seconds
hive>
```

```
hive> Select touppercase (name) from STUDENT;
OK
JOHN
JACK
SMITH
SCOTT
JOSHI
ALEX
DAVID
JAMES
JOHN
JOSHI
Time taken: 0.061 seconds, Fetched: 10 row(s)
hive>
```

REMIND ME

- Hive is a Data Warehousing tool.
- Hive is used to query structured data built on top of Hadoop.
- Hive provides HQL (Hive Query Language) which is similar to SQL.
- A Hive database contains several tables. Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as a sub-directory.
- Bucketed tables are stored as a file.

POINT ME (BOOKS)

- Programming Hive, Jason Rutherglen, O'Reilly Publication.

CONNECT ME (INTERNET RESOURCES)

- **http://en.wikipedia.org/wiki/RCFile**
- **https://cwiki.apache.org/confluence/display/Hive/DynamicPartitions**
- **https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL**
- **https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML**

TEST ME

A. Match Me

Column A	Column B
HQL	Web Logs
Database	struct, map
Complex Data Types	Set of records
Hive Application	Hive Query Language
Table	Namespace

Answers:

Column A	Column B
HQL	Hive Query Language
Database	Namespace
Complex Data Types	struct, map
Hive Application	Web Logs
Table	Set of records

B. Fill Me

1. The metastore consists of _____ and a _____.
2. The most commonly used interface to interact with Hive is _____.
3. The default metastore for Hive is _____.
4. Metastore contains _____ of Hive tables.
5. _____ is responsible for compilation, optimization, and execution of Hive queries.

Answers:

1. Metaservices, database
2. Command Line Interface
3. Derby
4. System Catalog
5. Driver

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1: PARTITION

Objective: To learn about partitions in hive.

Problem Description:

Create a partition table for customer schema to reward the customers based on their life time values.

Input:

Customer ID	Customers	Life Time Value
1001	Jack	25000
1002	Smith	8000
1003	David	12000
1004	John	15000
1005	Scott	12000
1006	Joshi	28000
1007	Ajay	12000
1008	Vinay	30000
1009	Joseph	21000

- Create a partition table if life time value is 12000.
- Create a partition table for all life time values.

ASSIGNMENT 2: PARTITION

1. Create Table:

```
CREATE EXTERNAL TABLE Emp_Proj (
   EmpID INT,
   TechnologyID INT,
   StartData date,
     EndDate date
) PARTITIONED BY (ProjectID INT)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '#'
  STORED AS TEXTFILE;
```

2. Load partitioned data:

```
LOAD DATA INPATH '/hive/data/Employee' INTO TABLE Emp_Proj PARTITION (ProjectID=1)
```

3. Verify data load:

```
SELECT * from Emp_Proj where ProjectID = 1;
```

Did you notice, it prints all data, that is, data with ProjectID = 2,3,4,5 as well.
Why did this happen? Was partitioning NOT performed?

4. Verify file location:

hadoop fs -ls /user/hive/warehouse/Employee.db/Emp_Proj/

It should have folder named ProjectID=1.

DYNAMIC PARTITIONING:

1. Create Table:

```
CREATE EXTERNAL TABLE Emp_Proj_DP (
    EmpID INT,
   TechnologyID INT,
   StartData date,
    EndDate date
) PARTITIONED BY (ProjectID INT)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '#'
 STORED AS TEXTFILE;
```

2. Create TEMP table:

```
CREATE EXTERNAL TABLE Temp_ProjectID (
    EmpID INT,
   TechnologyID INT,
   StartData date,
    EndDate date,
    ProjectID INT
)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '#'
 STORED AS TEXTFILE;
```

3. Load data in temp table from file:

LOAD DATA LOCAL INPATH '/home/Seema/bigdata/hivedata/Employee' INTO TABLE Temp_ProjectID;

4. Load data in dynamic partitioned table:

INSERT INTO TABLE Emp_Proj_DP PARTITION (ProjectID) SELECT EmpID, TechnologyID , StartDate , EndDate FROM Temp_ProjectID;

Note: The column order while select should be maintained except partitioned column, which should be selected last and if there are multiple partitioned columns then they should be cited in the order of creation.

5. Verify data load:

```
SELECT * from Emp_Proj_DP where ProjectID = 1;
SELECT * from Emp_Proj_DP where ProjectID = 2;
SELECT * from Emp_Proj_DP where ProjectID = 3;
SELECT * from Emp_Proj_DP where ProjectID = 4;
```

6. Verify file location:

```
hadoop fs -ls /user/hive/warehouse/Employee.db/Emp_Proj_DP/
```

It should have folder named ProjectID=1, ProjectID=2, ProjectID=3, etc..

ASSIGNMENT 3: HIVEQL

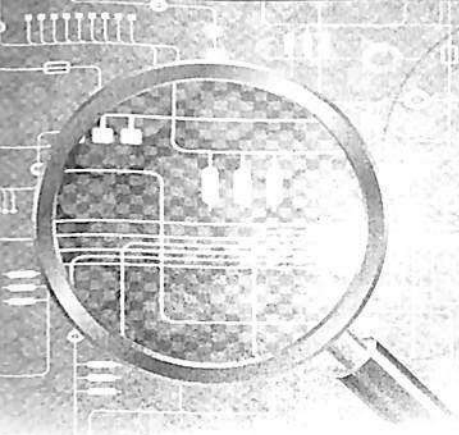
Objective: To learn about HiveQL statement.

Problem Description:

Create a data file for below schemas:

- **Order:** CustomerId, ItemId, ItemName, OrderDate, DeliveryDate
- **Customer:** CustomerId, CustomerName, Address, City, State, Country

1. Create a table for Order and Customer Data.
2. Write a HiveQL to find number of items bought by each customer.



CHAPTER 10

Introduction to Pig

BRIEF CONTENTS

- What's in Store?
- What is Pig?
  - Key Features of Pig
- The Anatomy of Pig
- Pig on Hadoop
- Pig Philosophy
- Use Case for Pig: ETL Processing
- Pig Latin Overview
  - Pig Latin Statements
  - Pig Latin: Keywords
  - Pig Latin: Identifiers
  - Pig Latin: Comments
  - Pig Latin: Case Sensitivity
  - Operators in Pig Latin
- Data Types in Pig
  - Simple Data Types
  - Complex Data Types
- Running Pig
  - Interactive Mode
  - Batch Mode
- Execution Modes of Pig
  - Local Mode
  - MapReduce Mode
- HDFS Commands
- Relational Operators
- EVAL Function
- Complex Data Types
  - Tuple
  - Map
- Piggy Bank
- User-Defined Functions (UDF)
- Parameter Substitution
- Diagnostic Operator
- Word Count Example using Pig
- When to use Pig?
- When NOT to use Pig?
- Pig at Yahoo!
- Pig versus Hive

"If you can't explain it simply, you don't understand it well enough."

– Albert Einstein, Physicist

WHAT'S IN STORE?

We assume that by now you would have become familiar with the basic concepts of HDFS and MapReduce Programming. The focus of this chapter will be to build on this knowledge to perform analysis using Pig. We will discuss few relational and eval operators of Pig. We will also discuss Complex Data Types, Piggy Bank, and UDF (User Defined Functions) of Pig.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning. We also suggest you to practice "Test Me" exercises.

10.1 WHAT IS PIG?

Apache Pig is a platform for data analysis. It is an alternative to MapReduce Programming. Pig was developed as a research project at Yahoo.

10.1.1 Key Features of Pig

1. It provides an **engine** for executing **data flows** (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
2. It provides a language called "**Pig Latin**" to express data flows.
3. Pig Latin contains operators for many of the traditional data operations such as join, filter, sort, etc.
4. It allows users to develop their own functions (User Defined Functions) for reading, processing, and writing data.

10.2 THE ANATOMY OF PIG

The main components of Pig are as follows:

1. Data flow language (**Pig Latin**).
2. Interactive shell where you can type Pig Latin statements (**Grunt**).
3. Pig interpreter and execution engine.

Refer Figure 10.1.

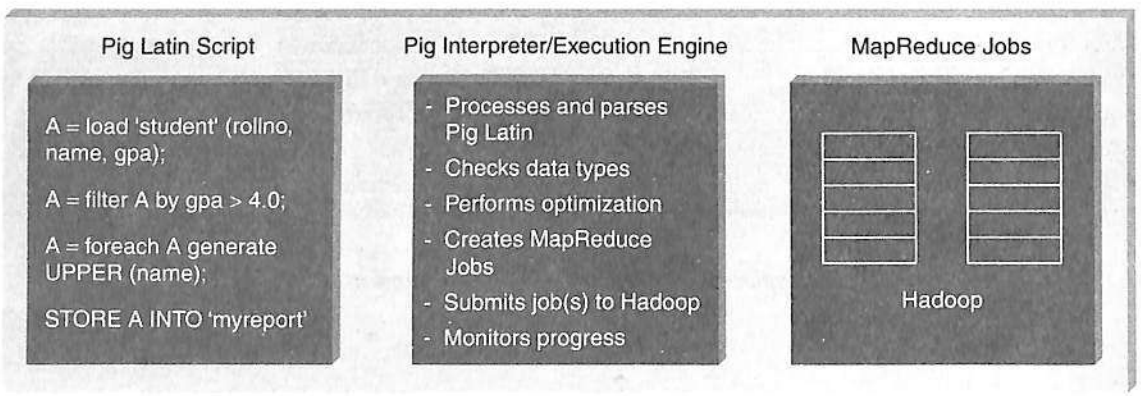


Figure 10.1 The anatomy of Pig.

10.3 PIG ON HADOOP

Pig runs on Hadoop. Pig uses both Hadoop Distributed File System and MapReduce Programming. By default, Pig reads input files from HDFS. Pig stores the intermediate data (data produced by MapReduce jobs) and the output in HDFS. However, Pig can also read input from and place output to other sources.

Pig supports the following:

1. HDFS commands.
2. UNIX shell commands.
3. Relational operators.
4. Positional parameters.
5. Common mathematical functions.
6. Custom functions.
7. Complex data structures.

10.4 PIG PHILOSOPHY

Figure 10.2 describes the Pig philosophy.

1. **Pigs Eat Anything:** Pig can process different kinds of data such as structured and unstructured data.
2. **Pigs Live Anywhere:** Pig not only processes files in HDFS, it also processes files in other sources such as files in the local file system.
3. **Pigs are Domestic Animals:** Pig allows you to develop user-defined functions and the same can be included in the script for complex operations.
4. **Pigs Fly:** Pig processes data quickly.

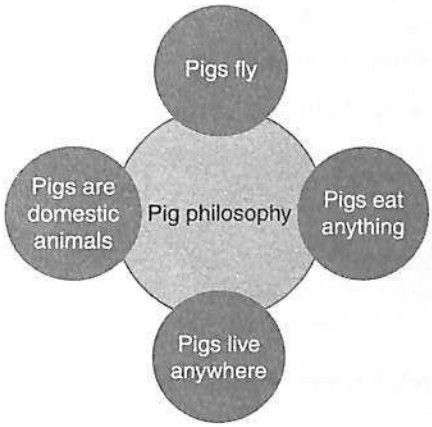


Figure 10.2 Pig philosophy.

10.5 USE CASE FOR PIG: ETL PROCESSING

Pig is widely used for "ETL" (**E**xtract, **T**ransform, and **L**oad). Pig can extract data from different sources such as ERP, Accounting, Flat Files, etc. Pig then makes use of various operators to perform transformation on the data and subsequently loads it into the data warehouse. Refer Figure 10.3.

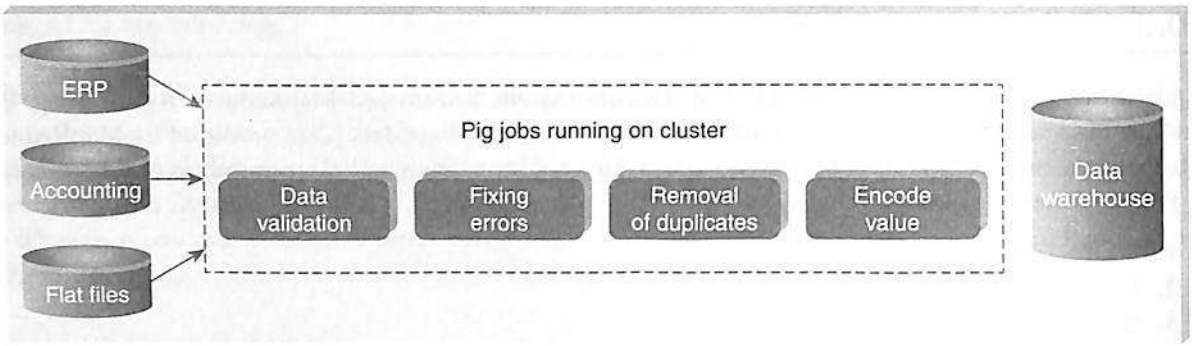


Figure 10.3 Pig: ETL Processing.

10.6 PIG LATIN OVERVIEW

10.6.1 Pig Latin Statements

1. Pig Latin statements are basic constructs to process data using Pig.
2. Pig Latin statement is an operator.
3. An operator in Pig Latin takes a relation as input and yields another relation as output.
4. Pig Latin statements include schemas and expressions to process data.
5. Pig Latin statements should end with a semi-colon.

Pig Latin Statements are generally ordered as follows:

1. **LOAD** statement that reads data from the file system.
2. Series of statements to perform transformations.
3. **DUMP** or **STORE** to display/store result.

The following is a simple Pig Latin script to load, filter, and store "student" data.

```
A = load 'student' (rollno, name, gpa);
A = filter A by gpa > 4.0;
A = foreach A generate UPPER (name);
STORE A INTO 'myreport'
```

Note: In the above example **A** is a relation and NOT a variable.

10.6.2 Pig Latin: Keywords

Keywords are reserved. It cannot be used to name things.

10.6.3 Pig Latin: Identifiers

1. Identifiers are names assigned to fields or other data structures.
2. It should begin with a letter and should be followed only by letters, numbers, and underscores.

Table 10.1 Valid and invalid identifiers

Valid Identifier	Y	A1	A1_2014	Sample
Invalid Identifier	5	Sales$	Sales%	_Sales

Table 10.1 describes valid and invalid identifiers.

10.6.4 Pig Latin: Comments

In Pig Latin two types of comments are supported:

1. Single line comments that begin with "--".
2. Multiline comments that begin with "/* and end with */".

10.6.5 Pig Latin: Case Sensitivity

1. Keywords are *not* case sensitive such as LOAD, STORE, GROUP, FOREACH, DUMP, etc.
2. Relations and paths are case-sensitive.
3. Function names are case sensitive such as PigStorage, COUNT.

10.6.6 Operators in Pig Latin

Table 10.2 describes operators in Pig Latin.

Table 10.2 Operators in Pig Latin

Arithmetic	Comparison	Null	Boolean
+	==	IS NULL	AND
–	!=	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

10.7 DATA TYPES IN PIG

10.7.1 Simple Data Types

Table 10.3 describes simple data types supported in Pig. In Pig, fields of unspecified types are considered as an array of bytes which is known as bytearray.

Null: In Pig Latin, **NULL** denotes a value that is unknown or is non-existent.

10.7.2 Complex Data Types

Table 10.4 describes complex data types in Pig.

Table 10.3 Simple data types supported in Pig

Name	Description
Int	Whole numbers
Long	Large whole numbers
Float	Decimals
Double	Very precise decimals
Chararray	Text strings
Bytearray	Raw bytes
Datetime	Datetime
Boolean	true or false

Table 10.4 Complex data types in Pig

Name	Description
Tuple	An ordered set of fields. **Example: (2,3)**
Bag	A collection of tuples. **Example: {(2,3),(7,5)}**
map	key, value pair **(open # Apache)**

10.8 RUNNING PIG

You can run Pig in two ways:

1. Interactive Mode.
2. Batch Mode.

10.8.1 Interactive Mode

You can run Pig in interactive mode by invoking **grunt** shell. Type pig to get grunt shell as shown below.

```
root@volgalnx010:~
[root@volgalnx010 ~]# pig
2015-02-23 21:07:38,916 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.
1.3 (rexported) compiled Sep 16 2014, 20:39:43
2015-02-23 21:07:38,917 [main] INFO  org.apache.pig.Main - Logging error messages to: /roo
t/pig_1424705858915.log
2015-02-23 21:07:38,934 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file
/root/.pigbootup not found
2015-02-23 21:07:39,313 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - ma
pred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2015-02-23 21:07:39,313 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs
.default.name is deprecated. Instead, use fs.defaultFS
2015-02-23 21:07:39,313 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecut
ionEngine - Connecting to hadoop file system at: hdfs://volgalnx010.ad.infosys.com:9000
2015-02-23 21:07:39,800 [main] WARN  org.apache.hadoop.util.NativeCodeLoader - Unable to l
oad native-hadoop library for your platform... using builtin-java classes where applicable
2015-02-23 21:07:40,234 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs
.default.name is deprecated. Instead, use fs.defaultFS
grunt>
```

Once you get the grunt prompt, you can type the Pig Latin statement as shown below.

```
grunt> A = load '/pigdemo/student.tsv' as (rollno, name, gpa);
grunt> DUMP A;
```

Here, the path refers to HDFS path and DUMP displays the result on the console as shown below.

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
grunt>
```

10.8.2 Batch Mode

You need to create **"Pig Script"** to run pig in batch mode. Write Pig Latin statements in a file and save it with **.pig** extension.

10.9 EXECUTION MODES OF PIG

You can execute pig in two modes:

1. Local Mode.
2. MapReduce Mode.

10.9.1 Local Mode

To run pig in local mode, you need to have your files in the local file system.

Syntax:

pig –x local filename

10.9.2 MapReduce Mode

To run pig in MapReduce mode, you need to have access to a Hadoop Cluster to read /write file. This is the default mode of Pig.

Syntax:

pig filename

10.10 HDFS COMMANDS

You can work with all HDFS commands in Grunt shell. For example, you can create a directory as shown below.

```
grunt> fs -mkdir /piglatindemos;
grunt>
```

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?
Input: What is the input that has been given to us to act upon?
Act: The actual statement/command to accomplish the task at hand.
Outcome: The result/output as a consequence of executing the statement.

10.11 RELATIONAL OPERATORS

10.11.1 FILTER

FILTER operator is used to select tuples from a relation based on specified conditions.

Objective: Find the tuples of those student where the GPA is greater than 4.0.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = filter A by gpa > 4.0;
DUMP B;
```

Output:

```
(1003,Smith,4.5)
(1004,Scott,4.2)
[root@volgalnx010 pigdemos]#
```

10.11.2 FOREACH

Use ***FOREACH*** when you want to do data transformation based on columns of data.

Objective: Display the name of all students in uppercase.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = foreach A generate UPPER (name);
DUMP B;
```

Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(JOSHI)
[root@volgalnx010 pigdemos]#
```

10.11.3 GROUP

GROUP operator is used to group data.

Objective: Group tuples of students based on their GPA.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = GROUP A BY gpa;
DUMP B;
```

Output:

```
(3.0,{(1001,John,3.0),(1001,John,3.0)})
(3.5,{(1005,Joshi,3.5),(1005,Joshi,3.5)})
(4.0,{(1008,James,4.0),(1002,Jack,4.0)})
(4.2,{(1007,David,4.2),(1004,Scott,4.2)})
(4.5,{(1006,Alex,4.5),(1003,Smith,4.5)})
[root@volgalnx010 pigdemos]#
```

10.11.4 DISTINCT

DISTINCT operator is used to remove duplicate tuples. In Pig, DISTINCT operator works on the entire tuple and NOT on individual fields.

Objective: To remove duplicate tuples of students.

Input:

Student (rollno:int,name:chararray,gpa:float)

Input:

1001	**John**	**3.0**
1002	**Jack**	**4.0**
1003	**Smith**	**4.5**
1004	**Scott**	**4.2**
1005	**Joshi**	**3.5**
1006	**Alex**	**4.5**
1007	**David**	**4.2**
1008	**James**	**4.0**
1001	**John**	**3.0**
1005	**Joshi**	**3.5**

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = DISTINCT A;
DUMP B;
```

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
(1006,Alex,4.5)
(1007,David,4.2)
(1008,James,4.0)
[root@volgalnx010 pigdemos]#
```

10.11.5 LIMIT

LIMIT operator is used to limit the number of output tuples.

Objective: Display the first 3 tuples from the "student" relation.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = LIMIT A 3;
DUMP B;
```

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
[root@volgalnx010 pigdemos]#
```

10.11.6 ORDER BY

ORDER BY is used to sort a relation based on specific value.

Objective: Display the names of the students in Ascending Order.
Input:
Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = ORDER A BY name;
DUMP B;
```

Output:

```
(1006,Alex,4.5)
(1007,David,4.2)
(1002,Jack,4.0)
(1008,James,4.0)
(1001,John,3.0)
(1001,John,3.0)
(1005,Joshi,3.5)
(1005,Joshi,3.5)
(1004,Scott,4.2)
(1003,Smith,4.5)
[root@volgalnx010 pigdemos]#
```

10.11.7 JOIN

It is used to join two or more relations based on values in the common field. It always performs inner Join.

Objective: To join two relations namely, "student" and "department" based on the values contained in the "rollno" column.

Input:

Student (rollno:int,name:chararray,gpa:float)

Department(rollno:int,deptno:int,deptname:chararray)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int,deptname:chararray);
C = JOIN A BY rollno, B BY rollno;
DUMP C;
DUMP B;
```

Output:

```
(1001,John,3.0,1001,101,B.E.)
(1001,John,3.0,1001,101,B.E.)
(1002,Jack,4.0,1002,102,B.Tech)
(1003,Smith,4.5,1003,103,M.Tech)
(1004,Scott,4.2,1004,104,MCA)
(1005,Joshi,3.5,1005,105,MBA)
(1005,Joshi,3.5,1005,105,MBA)
(1006,Alex,4.5,1006,101,B.E)
(1007,David,4.2,1007,104,MCA)
(1008,James,4.0,1008,102,B.Tech)
[root@volgalnx010 pigdemos]#
```

10.11.8 UNION

It is used to merge the contents of two relations.

Objective: To merge the contents of two relations "student" and "department".
Input:
Student (rollno:int,name:chararray,gpa:float)
Department(rollno:int,deptno:int,deptname:chararray)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno, name, gp);
B = load '/pigdemo/department.tsv' as (rollno, deptno,deptname);
C = UNION A,B;
STORE C INTO '/pigdemo/uniondemo';
DUMP B;
```

Output:
"Store" is used to save the output to a specified path. The output is stored in two files: part-m-00000 contains "student" content and part-m-00001 contains "department" content.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup
part-m-00000	file	146 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup
part-m-00001	file	114 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup

File: /pigdemo/uniondemo/part-m-00000

Goto : /pigdemo/un-ondemo go

Go back to dir listing
Advanced view download options

```
1001    John    3.0
1002    Jack    4.0
1003    Smith   4.5
1004    Scott   4.2
1005    Joshi   3.5
1006    Alex    4.5
1007    David   4.2
1008    James   4.0
1001    John    3.0
1005    Joshi   3.5
```

File: /pigdemo/uniondemo/part-m-00001

Goto : /pigdemo/uniondemo go

Go back to dir listing
Advanced view download options

```
1001    101     B.E.
1002    102     B.Tech
1003    103     M.Tech
1004    104     MCA
1005    105     MBA
1006    101     B.E
1007    104     MCA
1008    102     B.Tech
```

10.11.9 SPLIT

It is used to partition a relation into two or more relations.

Objective: To partition a relation based on the GPAs acquired by the students.

- GPA = 4.0, place it into relation X.
- GPA is < 4.0, place it into relation Y.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
SPLIT A INTO X IF gpa==4.0, Y IF gpa<=4.0;
DUMP X;
```

Output: Relation X

```
(1002,Jack,4.0)
(1008,James,4.0)
[root@volgalnx010 pigdemos]#
```

Output: Relation Y

```
(1001,John,3.0)
(1002,Jack,4.0)
(1005,Joshi,3.5)
(1008,James,4.0)
(1001,John,3.0)
(1005,Joshi,3.5)
[root@volgalnx010 pigdemos]#
```

10.11.10 SAMPLE

It is used to select random sample of data based on the specified sample size.

Objective: To depict the use of ***SAMPLE.***

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = SAMPLE A 0.01;
DUMP B;
```

10.12 EVAL FUNCTION

10.12.1 AVG

AVG is used to compute the average of numeric values in a single column bag.

Objective: To calculate the average marks for each student.
Input:
Student (studname:chararray,marks:int)
Act:

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as (studname:chararray,marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname, AVG(A.marks);
DUMP C;
```

Output:

```
({(Jack),(Jack),(Jack),(Jack)},39.75)
({(John),(John),(John),(John)},39.0)
[root@volgalnx010 pigdemos]#
```

Note: You need to use PigStorage function if you wish to manipulate files other than .csv.

10.12.2 MAX

MAX is used to compute the maximum of numeric values in a single column bag.

Objective: To calculate the maximum marks for each student.
Input:
Student (studname:chararray,marks:int)
Act:

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname, MAX(A.marks);
DUMP C;
```

Output:

```
({(Jack),(Jack),(Jack),(Jack)},46)
({(John),(John),(John),(John)},45)
[root@volgalnx010 pigdemos]#
```

Note: Similarly, you can try the MIN and the SUM functions as well.

10.12.3 COUNT

- *COUNT* is used to count the number of elements in a bag.

Objective: To count the number of tuples in a bag.
Input:
Student (studname:chararray,marks:int)
Act:

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE  A. studname,COUNT(A);
DUMP C;
```

Output:

```
({(Jack),(Jack),(Jack),(Jack)},4)
({(John),(John),(John),(John)},4)
[root@volgalnx010 pigdemos]#
```

Note: The default file format of Pig is .tsv file. Use PigStorage() to manipulate files other than .tsv file.

10.13 COMPLEX DATA TYPES

10.13.1 TUPLE

A *TUPLE* is an ordered collection of fields.

Objective: To use the complex data type "Tuple" to load data.
Input:

(John,12)	(Jack,13)
(James,7)	(Joseph,5)
(Smith,8)	(Scott,12)

Act:

```
A = LOAD '/root/pigdemos/studentdata.tsv' AS (t1:tuple(t1a:chararray,
t1b:int),t2:tuple(t2a:chararray,t2b:int));
B = FOREACH A GENERATE t1.t1a, t1.t1b,t2.$0,t2.$1;
DUMP B;
```

Output:

```
(John,12,Jack,13)
(James,7,Joseph,5)
(Smith,8,Scott,12)
[root@volgalnx010 pigdemos]#
```

Note: You can refer to the field using Positional Notation as shown above. The Positional Notation is denoted by $ sign and the position starts with 0 (e.g., $0).

10.13.2 MAP

MAP represents a key/value pair.

Objective: To depict the complex data type "map".

Input:

John	**[city#Bangalore]**
Jack	**[city#Pune]**
James	**[city#Chennai]**

Act:

```
A = load '/root/pigdemos/studentcity.tsv' Using PigStorage as
(studname:chararray,m:map[chararray]);
B = foreach A generate m#'city' as CityName:chararray;
DUMP B
```

Output:

```
(Bangalore)
(Pune)
(Chennai)
[root@volgalnx010 pigdemos]#
```

10.14 PIGGY BANK

Pig user can use Piggy Bank functions in Pig Latin script and they can also share their functions in Piggy Bank.

Objective: To use Piggy Bank string UPPER function.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
register '/root/pigdemos/piggybank-0.12.0.jar';
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
upper = foreach A generate
org.apache.pig.piggybank.evaluation.string.UPPER(name);
DUMP upper;
```

Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(JOSHI)
(ALEX)
(DAVID)
(JAMES)
(JOHN)
(JOSHI)
[root@volgalnx010 pigdemos]#
```

Note: You need to use the "**register**" keyword to use Piggy Bank jar function in your pig script.

10.15 USER-DEFINED FUNCTIONS (UDF)

Pig allows you to create your own function for complex analysis.

Objective: To depict user-defined function.

Java Code to convert name into uppercase:

```
package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;
public class UPPER extends EvalFunc<String>
  {
    public String exec(Tuple input) throws IOException {
      if (input == null || input.size() == 0)
        return null;
        try{
          String str = (String)input.get(0);
          return str.toUpperCase();
        }catch(Exception e){
          throw WrappedIOException.wrap("Caught exception processing input row ", e);
      }
    }
  }
```

Note: Convert above java class into jar to include this function into your code.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
register /root/pigdemos/myudfs.jar;
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = FOREACH A GENERATE myudfs.UPPER(name);
DUMP B;
```

Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(JOSHI)
(ALEX)
(DAVID)
(JAMES)
(JOHN)
(JOSHI)
[root@volgalnx010 pigdemos]#
```

10.16 PARAMETER SUBSTITUTION

Pig allows you to pass parameters at runtime.

Objective: To depict parameter substitution.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '$student' as (rollno:int, name:chararray, gpa:float);
DUMP A;
```

Execute:

```
pig –param student=/pigdemo/student.tsv parameterdemo.pig
```

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
(1006,Alex,4.5)
(1007,David,4.2)
(1008,James,4.0)
(1001,John,3.0)
(1005,Joshi,3.5)
[root@volgalnx010 pigdemos]#
```

10.17 DIAGNOSTIC OPERATOR

It returns the schema of a relation.

Objective: To depict the use of ***DESCRIBE.***

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
DESCRIBE A;
```

Output:

```
A: {rollno: int,name: chararray,gpa: float}
[root@volgalnx010 pigdemos]#
```

10.18 WORD COUNT EXAMPLE USING PIG

Objective: To count the occurrence of similar words in a file.

Input:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act:

```
lines = LOAD '/root/pigdemos/lines.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

Output:

```
(to,2)
(Pig,1)
(Hive,2)
(Hadoop,2)
(Session,3)
(Welcome,1)
(Introducing,1)
(Introduction,1)
[root@volgalnx010 pigdemos]#
```

Note:

TOKENIZE splits the line into a field for each word.

FLATTEN will take the collection of records returned by TOKENIZE and produce a separate record for each one, calling the single field in the record word.

10.19 WHEN TO USE PIG?

Pig can be used in the following situations:

1. When your data loads are time sensitive.
2. When you want to process various data sources.
3. When you want to get analytical insights through sampling.

10.20 WHEN NOT TO USE PIG?

Pig should not be used in the following situations:

1. When your data is completely in the unstructured form such as video, text, and audio.
2. When there is a time constraint because Pig is slower than MapReduce jobs.

10.21 PIG AT YAHOO!

Yahoo uses Pig for two things:

1. **In Pipelines**, to fetch log data from its web servers and to perform cleansing to remove companies interval views and clicks.
2. **In Research**, script is used to test a theory. Pig provides facility to integrate Perl or Python script which can be executed on a huge dataset.

10.22 PIG versus HIVE

Features	Pig	Hive
Used By	Programmers and Researchers	Analyst
Used For	Programming	Reporting
Language	Procedural data flow language	SQL Like
Suitable For	Semi - Structured	Structured
Schema/Types	Explicit	Implicit
UDF Support	YES	YES
Join/Order/Sort	YES	YES
DFS Direct Access	YES (Implicit)	YES (Explicit)
Web Interface	YES	NO
Partitions	YES	NO
Shell	YES	YES

REMIND ME

- Apache Pig is a platform for data analysis. It is an alternative to MapReduce Programming.
- It provides an **engine** for executing **data flows** (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
- It provides a language called "**Pig Latin**" to express data flows.
- The main components of Pig are as follows:
  - Data flow language (**Pig Latin**).
  - Interactive shell where you can type Pig Latin statements (**Grunt**).
  - Pig interpreter and execution engine.
- You can run Pig in two ways:
  - Interactive Mode.
  - Batch Mode.

POINT ME (BOOK)

- Programming Pig, Alan Gates, O'REILY.

CONNECT ME (INTERNET RESOURCES)

- http://pig.apache.org/docs/r0.12.0/index.html
- http://www.edureka.co/blog/introduction-to-pig/
- http://www.edureka.co/blog/pig-vs-hive/

TEST ME

A. Fill Me

1. Pig is a _____ language.
2. In Pig, _____ is used to specify data flow.
3. Pig provides an _____ to execute data flow.
4. _____, _____ are execution modes of Pig.
5. The interactive mode of Pig is _____.
6. _____ and _____ are case sensitive in Pig.
7. _____, _____, _____ are Complex Data Types of Pig.
8. Pig is used in _____ process.

Answers:

1. Scripting
2. Pig Latin
3. Pig Engine
4. Local Mode, MapReduce Mode
5. Grunt
6. Fields and Aliases
7. Bag, Tuple, Map
8. ETL

B. Match Me

Column A	Column B
Map	Hadoop Cluster
Bag	An Ordered Collection of Fields
Local Mode	Collection of Tuples
Tuple	Key/Value Pair
MapReduce Mode	Local File System

Answers:

Column A	Column B
Map	Key/Value Pair
Bag	Collection of Tuples
Local Mode	Local File System
Tuple	An Ordered Collection of Fields
MapReduce Mode	Hadoop Cluster

C. True or False

1. PigStorage() function is case sensitive.
2. Local Mode is the default mode of Pig.
3. DISTINCT Keyword removes duplicate fields.
4. LIMIT keyword is used to display limited number of tuples in Pig.
5. ORDER BY is used for sorting.

Answers:

1. True
2. False
3. False
4. True
5. True

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1: SPLIT

Objective: To learn about SPLIT relational operator.

Problem Description:
Write a Pig Script to split customers for reward program based on their life time values.

Input:

Customers	Life Time Value
Jack	25000
Smith	8000
David	35000
John	15000
Scott	10000
Joshi	28000
Ajay	12000
Vinay	30000
Joseph	21000

- If Life Time Value is >1000 and <= 2000 → Sliver Program.
- If Life Time Value is >20000 → Gold Program.

ASSIGNMENT 2: GROUP

Objective: To learn about GROUP relational operator.

Problem Description:
Create a data file for below schemas:

- **Order:** CustomerId, ItemId, ItemName, OrderDate, DeliveryDate
- **Customer:** CustomerId, CustomerName, Address, City, State, Country

1. Load Order and Customer Data.
2. Write a Pig Latin Script to determine number of items bought by each customer.

ASSIGNMENT 3: COMPLEX DATA TYPE – BAG

Objective: To learn complex data type – bag in Pig.

Problem Description:

1. Create a file which contains bag dataset as shown below.

User ID	From	To
user1001	user1001@sample.com	{(user003@sample.com),(user004@sample.com), (user006@sample.com)}
user1002	user1002@sample.com	{(user005@sample.com), (user006@sample.com)}
user1003	user1003@sample.com	{(user001@sample.com),(user005@sample.com)}

2. Write a Pig Latin statement to display the names of all users who have sent emails and also a list of all the people that they have sent the email to.
3. Store the result in a file.

CHAPTER 11

JasperReport using Jaspersoft

BRIEF CONTENTS

- What's in Store?
- Introduction to JasperReports
  - JasperReports
  - Jaspersoft Studio
- Connecting to MongoDB NoSQL Database
  - Syntax of Few MongoDB Query Language
  - Elements and Attributes
  - Creating Variables
  - Creating Report Parameters
- Connecting to Cassandra NoSQL Database

"The greatest value of a picture is when it forces us to notice what we never expected to see."

– John Tukey, American Mathematician

WHAT'S IN STORE?

We assume that you are familiar with the MongoDB and Cassandra NoSQL databases. The focus of this chapter will be to build on this knowledge to create JasperReports using Jaspersoft Studio. We will discuss few MongoDB Queries to retrieve data from MongoDB and place it on the report.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning.

11.1 INTRODUCTION TO JASPERREPORTS

11.1.1 JasperReports

1. Open-source reporting engine.
2. Helps to create page-oriented reports.

3. Completely written in Java.
4. Reports can be embedded in any Java Application.
5. Represents reports in various formats such as HTML, PDF, CSV, etc.

11.1.2 Jaspersoft Studio

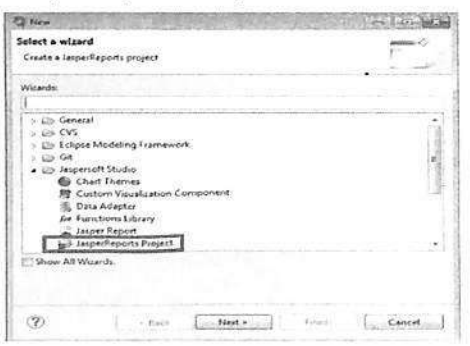
1. Eclipse based report designer.
2. Available in two flavors: Eclipse plugin and as a standalone application.
3. Data can be accessed from multiple sources such as JDBC, XML, CSV, etc.
4. Supports big data components such as MongoDB, Cassandra, Hive, etc.
5. Provides sophisticated layouts such as charts, crosstabs, images, subreports, etc.

11.2 CONNECTING TO MONGODB NoSQL DATABASE

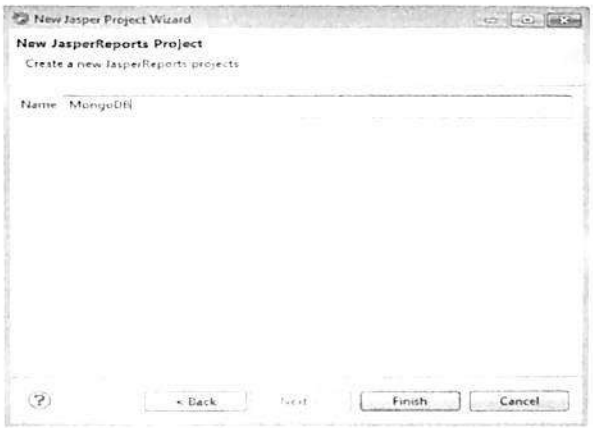
Jaspersoft uses **Jaspersoft MongoDB Query Language** to query the MongoDB databases. Jaspersoft MongoDB Query Language is a declarative language. It is used for stating which data to retrieve from which database. Connector converts this MongoDB query into the appropriate API calls. Then it makes use of MongoDB Java connector to query the MongoDB instance.

1. Double click on Jaspersoft Studio icon. You can see Jaspersoft Studio IDE as shown below.

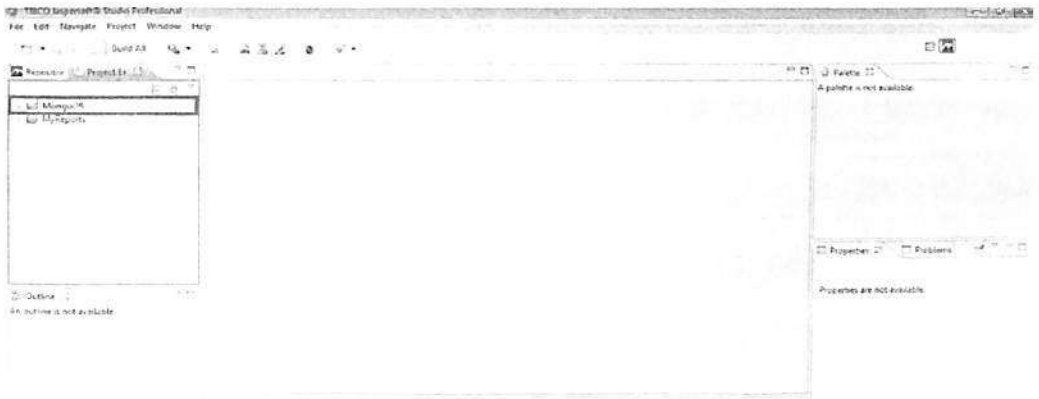
2. Select **File** → **New** → **Other** to create JasperReports Project. From the **New** wizard select **"JasperReports Project"** and click **"Next"**.



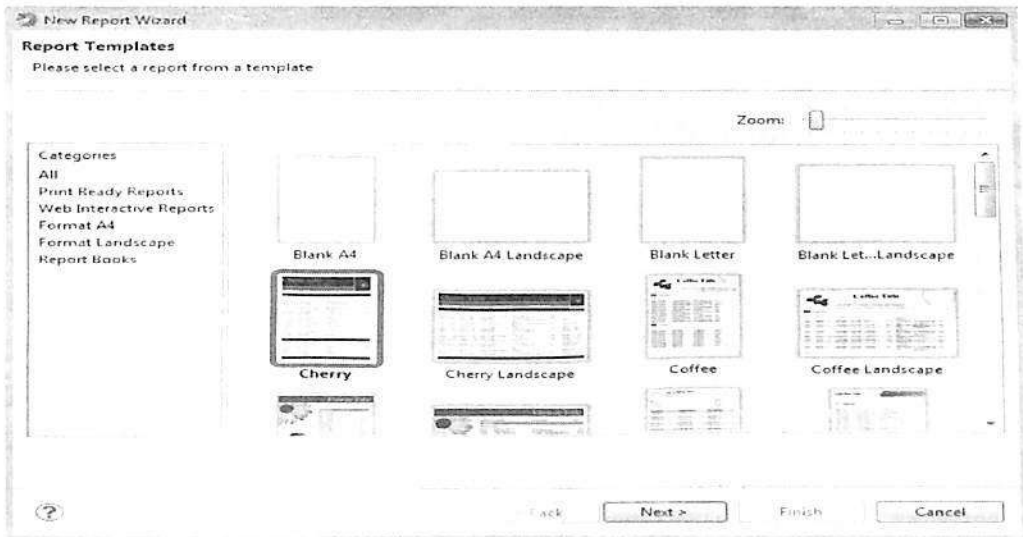
3. Mention project name as **"MongoDB"** and Click **"Finish"**.



4. Now, you can see the newly created project in **"Project Explorer"** window as shown below.

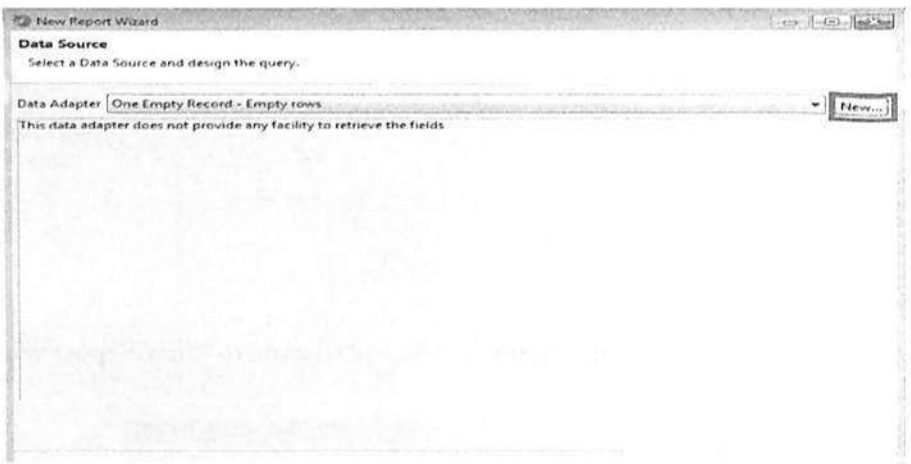


5. Right Click on the project, select **New → JasperReports**. This will take you to **"New Report Wizard"**. Select the required template and click **"Next"**.

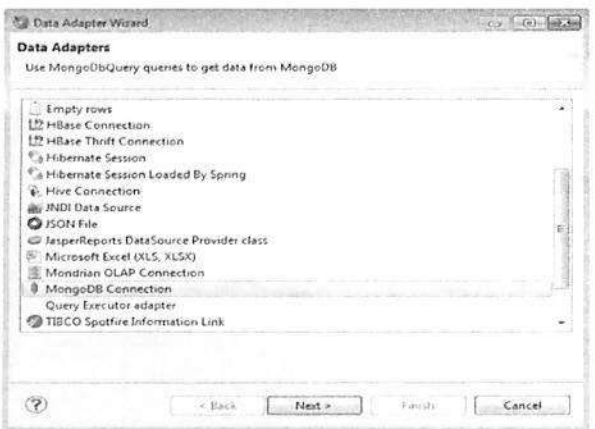


6. Then specify the file name as shown below and click on **"Next"** button.

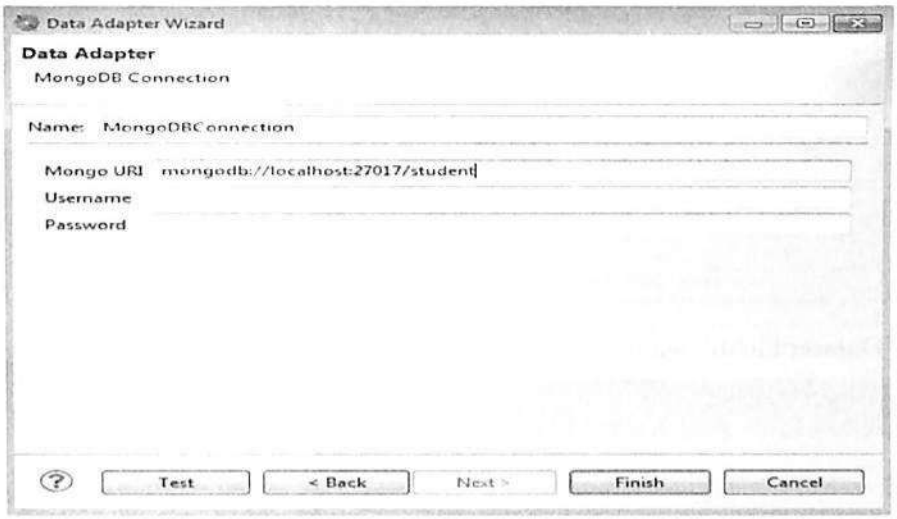
7. It will take you to the **Data Source** wizard. Click on **"New"** button as shown below.



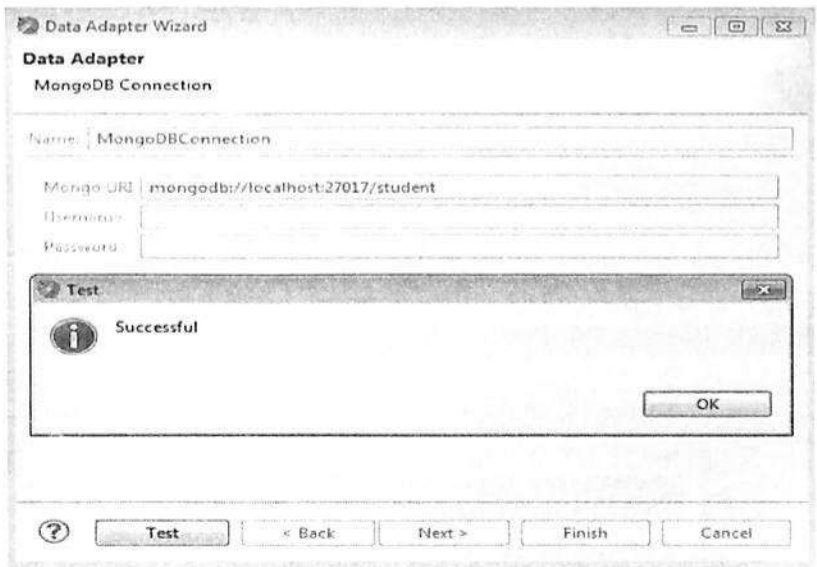
8. In the **"Data Adapter Wizard"**, select **"MongoDB Connection"** as shown below.



9. Specify the details as given below:
Name: MongoDBConnection
Mongo URI: mongodb://localhost:27017/student
Note: Here, "student" is the name of the database.

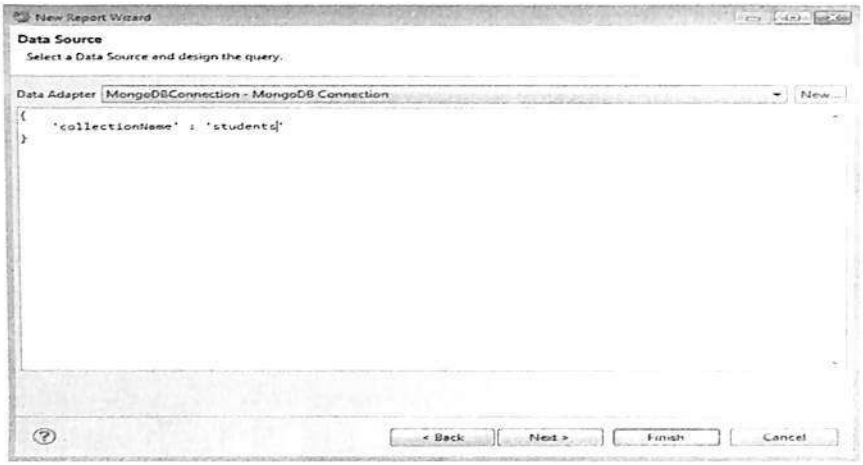


10. Click on **"Test"** button to test the connection. If the connection is properly set, you will get the message **"Successful"**.

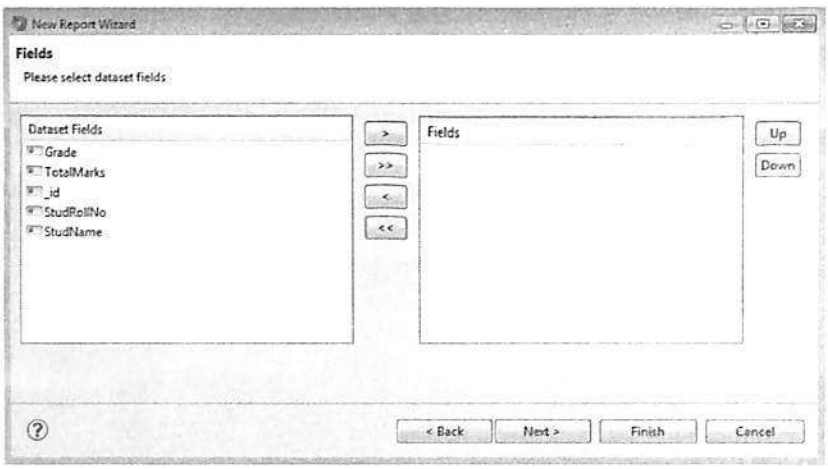


11. Let us write a MongoDB Query to retrieve data from the MongoDB database and click on the **"Next"** button. The bare minimal syntax (MongoDB Query Language) to retrieve all fields from the collection, "students" (the collection exists in the database, "student") is

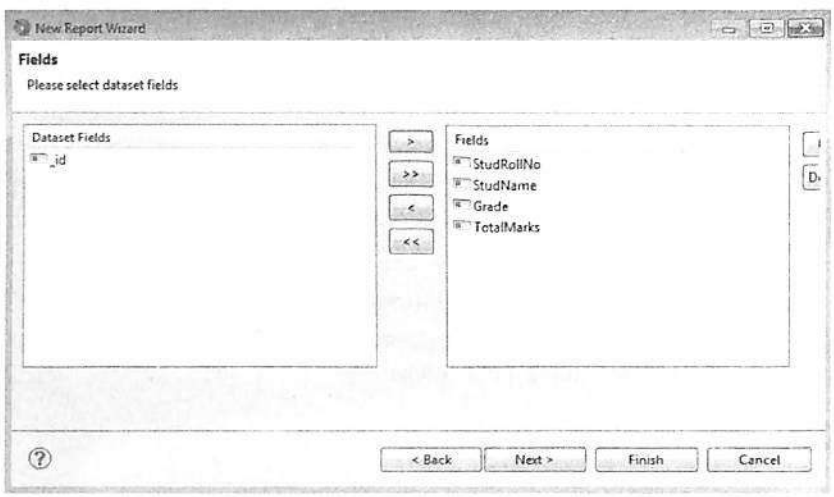
```
{
'collectionName': 'students'
}
```



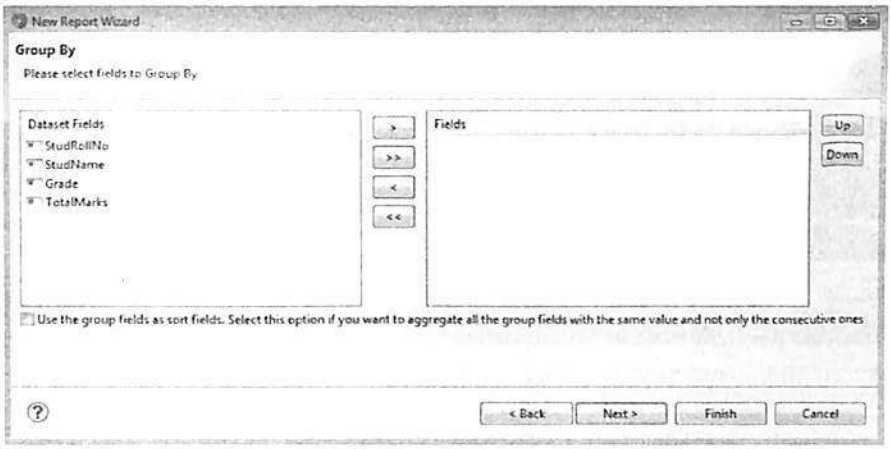
12. This will lead to the **"Dataset Fields"** wizard as shown below.



13. Select the required fields from the left side under **"Dataset Fields"** to the right side **"Fields"**. Click on **"Next"** button.

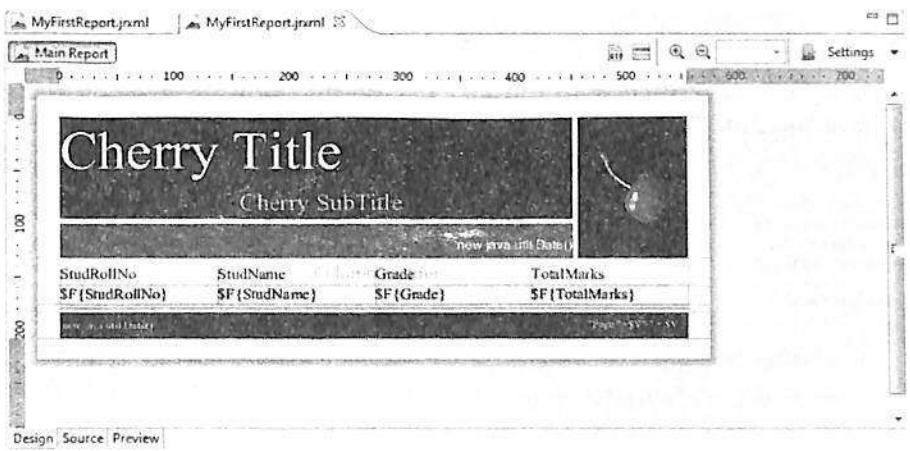


14. Ignore the "Group By" option and click on **"Next"**.

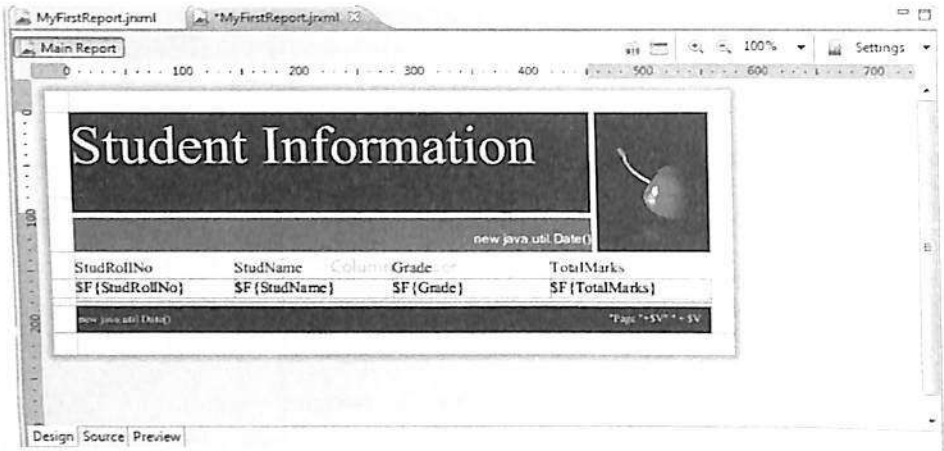


15. Finally, click on **"Finish"** button.

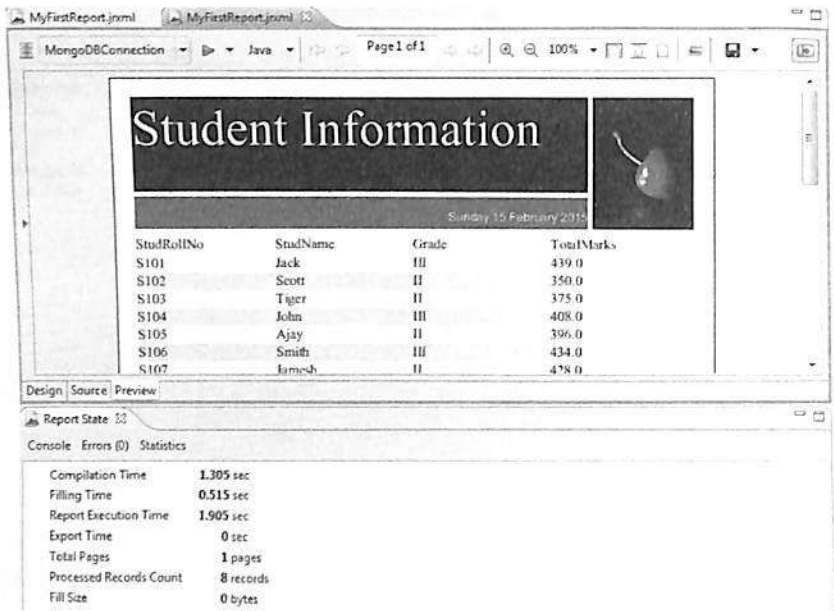
16. Your report will open in design mode as shown below. Double click on the text to change the title and delete the subtitle.



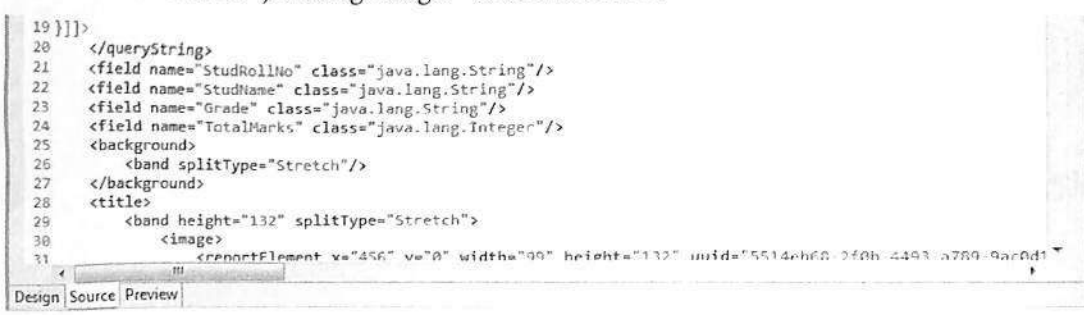
17. Specify the title name as "Student Information" as shown below.



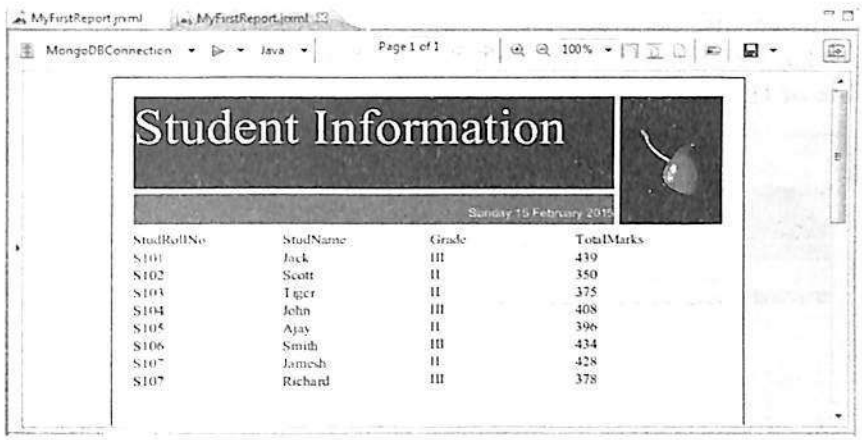
18. Click on the **"Preview"** tab to preview the report.



19. To change the "TotalMarks" column data type to Integer, click on the **"Source"** tab and change the "TotalMarks" class to "java.lang.Integer" as shown below.

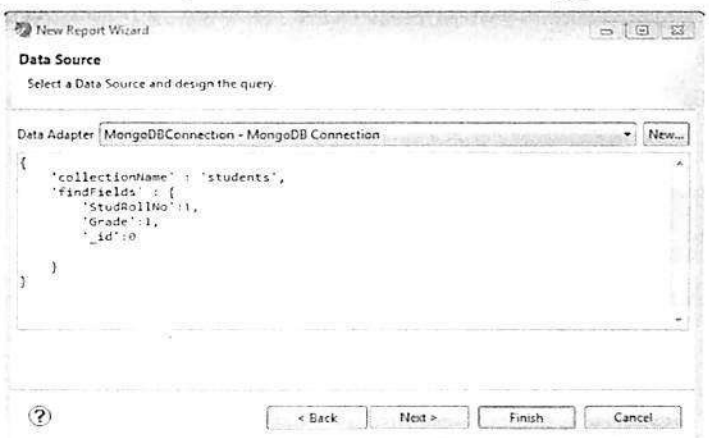


20. Click on the **"Preview"** tab to view the result.

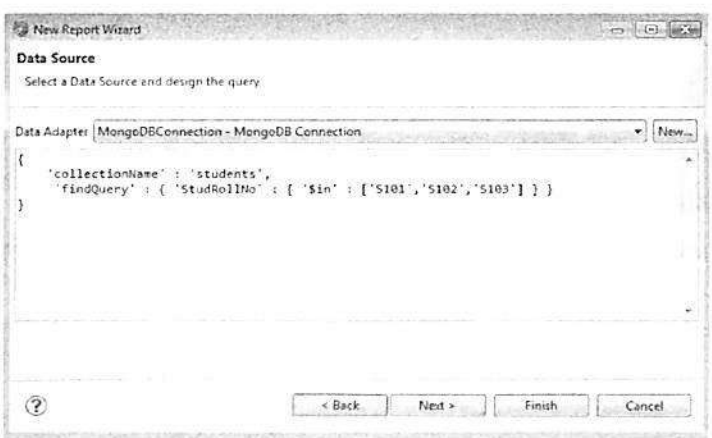


11.2.1 Syntax of Few MongoDB Query Language

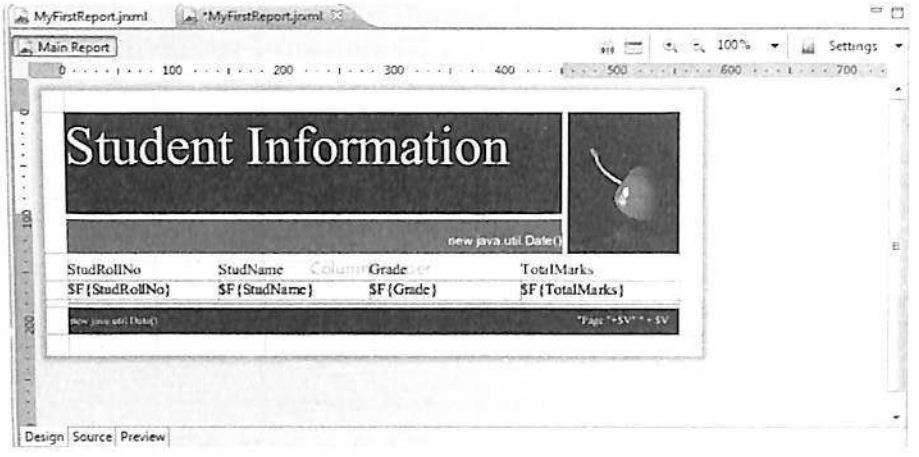
1. To return only StudRollNo, Grade and suppress the _id field.



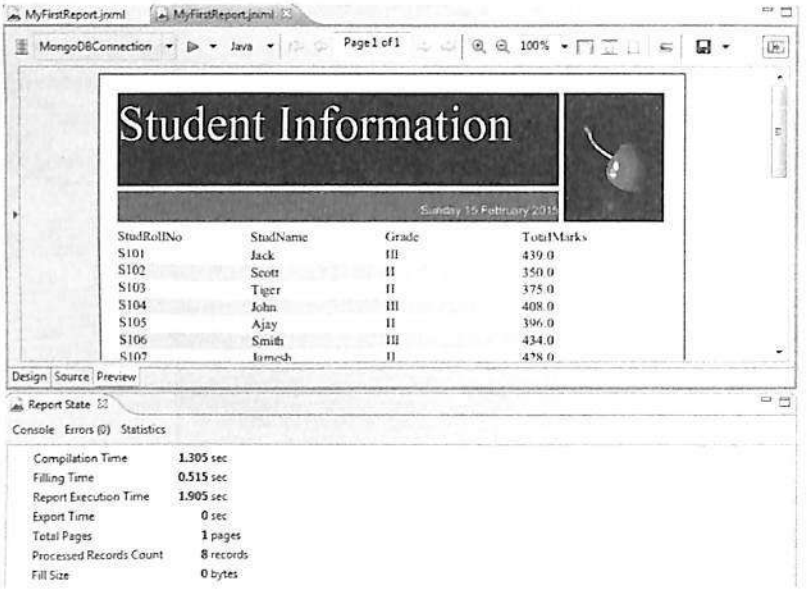
2. To select documents from a collection based on search criteria(s).



17. Specify the title name as "Student Information" as shown below.



18. Click on the **"Preview"** tab to preview the report.

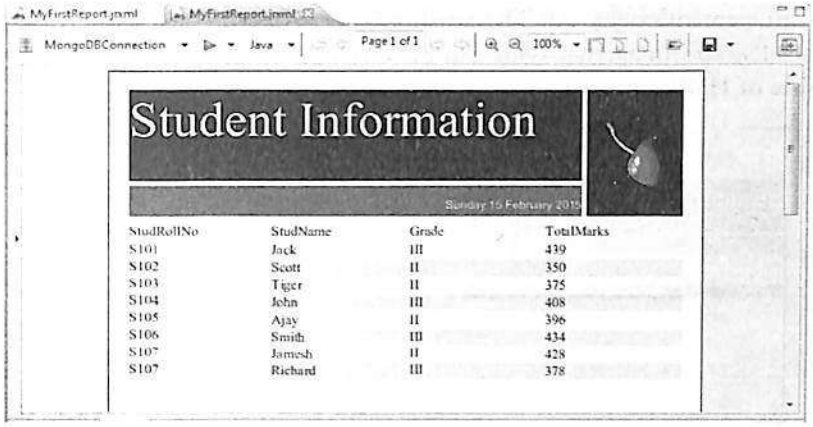


19. To change the "TotalMarks" column data type to Integer, click on the **"Source"** tab and change the "TotalMarks" class to "java.lang.Integer" as shown below.

```
19 ]]]>
20     </queryString>
21     <field name="StudRollNo" class="java.lang.String"/>
22     <field name="StudName" class="java.lang.String"/>
23     <field name="Grade" class="java.lang.String"/>
24     <field name="TotalMarks" class="java.lang.Integer"/>
25     <background>
26         <band splitType="Stretch"/>
27     </background>
28     <title>
29         <band height="132" splitType="Stretch">
30             <image>
31                 <reportElement x="456" y="0" width="99" height="132" uuid="5514eb68-2f0b-4493-a769-9ac0d1
```

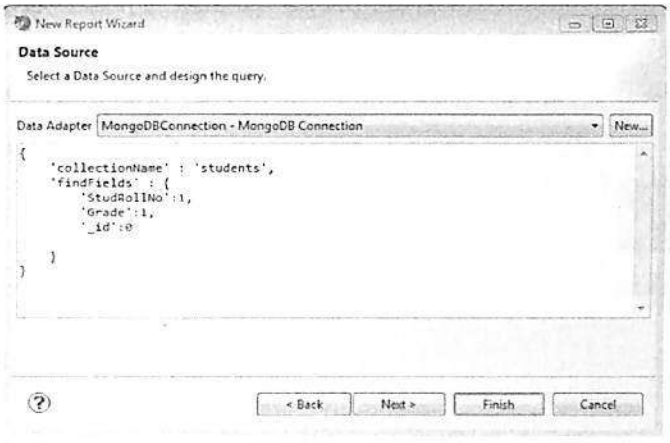
Design Source Preview

20. Click on the **"Preview"** tab to view the result.

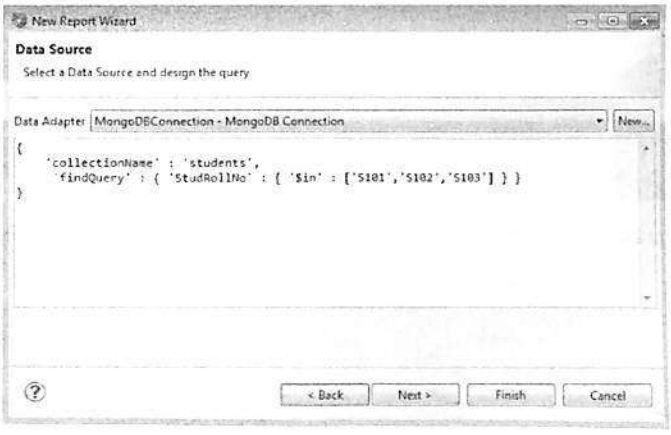


11.2.1 Syntax of Few MongoDB Query Language

1. To return only StudRollNo, Grade and suppress the _id field.

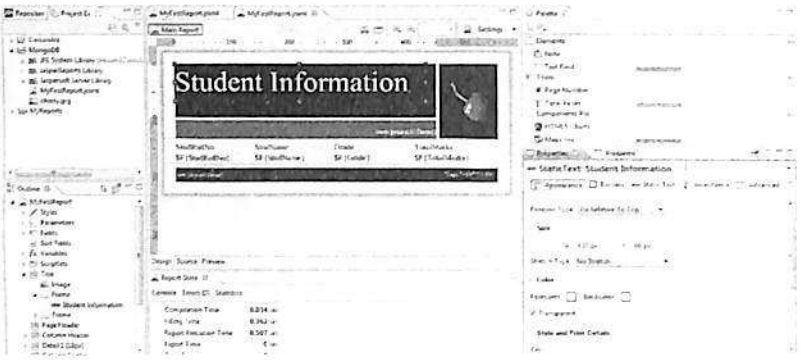


2. To select documents from a collection based on search criteria(s).



11.2.2 Elements and Attributes

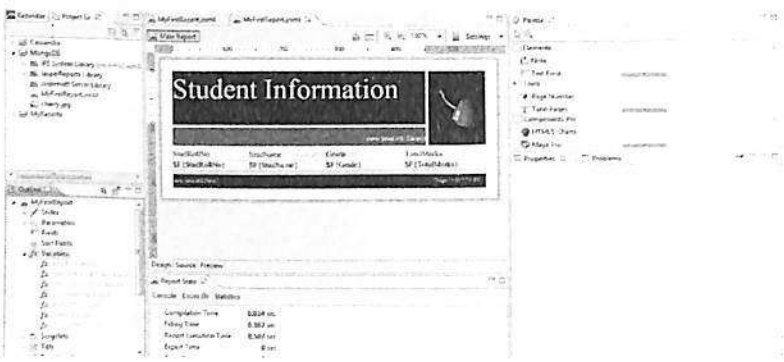
Attributes can be used to specify an element's behavior. The attributes are visible in the property tab. The elements such as title, column header, and image behavior can be specified by attributes available in the **"Properties Window" (Right Side of IDE)** as shown below.



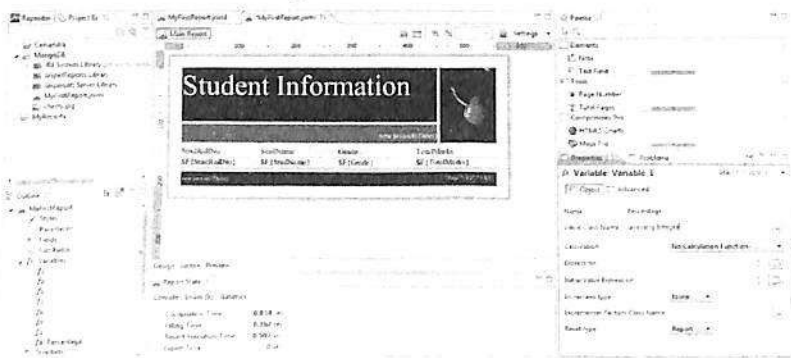
11.2.3 Creating Variables

Variables can be used to do complex calculations on the data extracted from the database. This can be stored and used later.

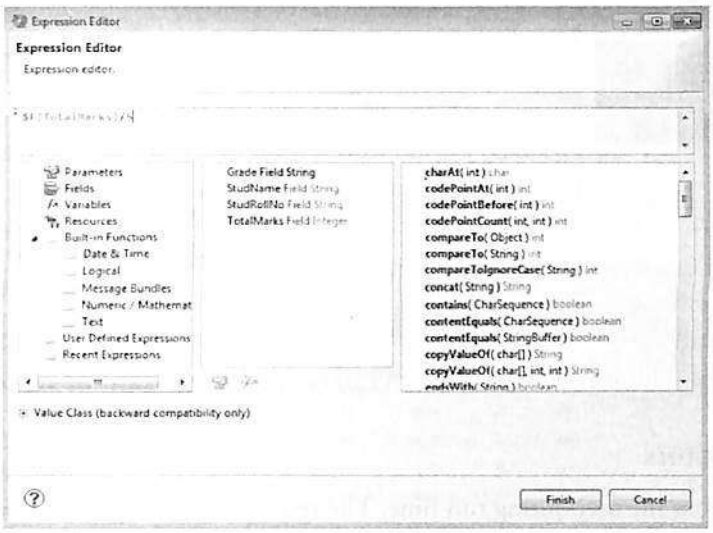
1. From the outline menu select **"Variables"** root node, Right click on it and select **"Create Variable"**.



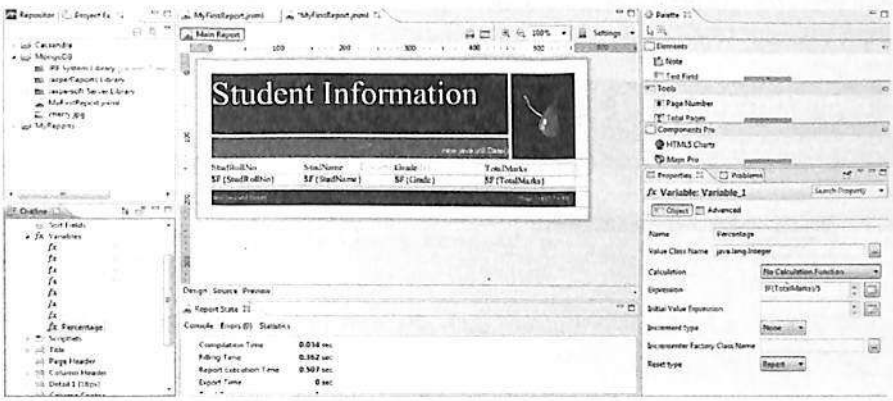
2. Once it is selected, its property will appear in the property window as shown below. Specify the variable name and the data type for the variable as shown below.



3. Click on the **"Expression"** tab to create an expression. Type an expression to calculate Percentage of Marks as shown below.

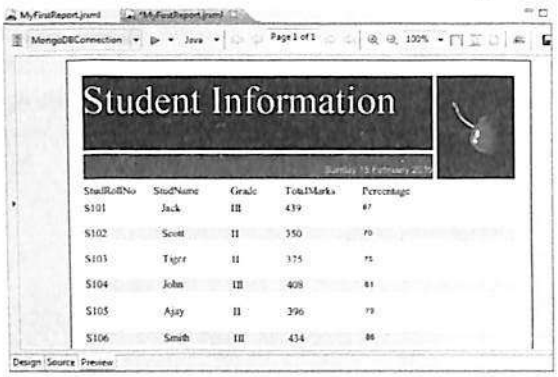


4. You can see the Percentage variable in outline window under the Variables item.



5. Now, drag and drop the variable into the detail band as shown below and add static label in the column header to display the "Percentage" column.

6. Click on the **"Preview"** tab to view the report.

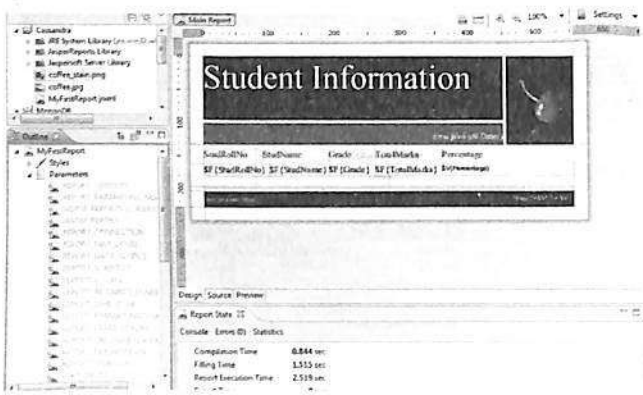


Note: Variable root node also contains built-in variables, which can be used directly in the report.

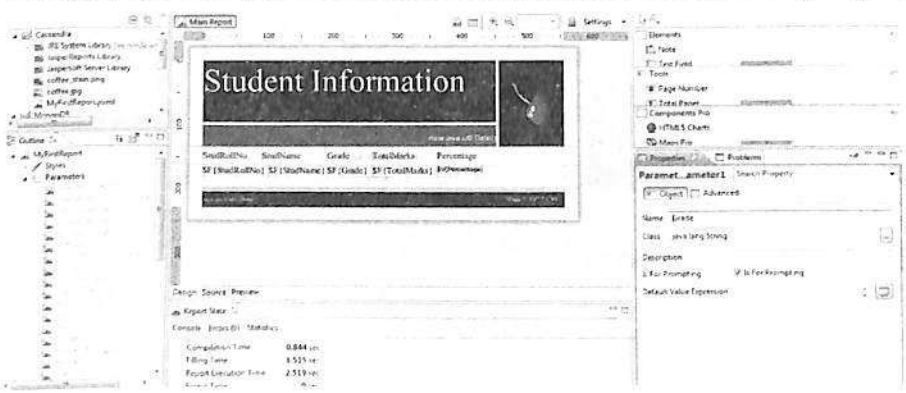
11.2.4 Creating Report Parameters

Report parameter is used to take input from the user during run time. The result will be displayed based on the provided input.

1. From the outline menu select the **"Parameters"** root node. Right click on it and select **"Create Parameter"**.



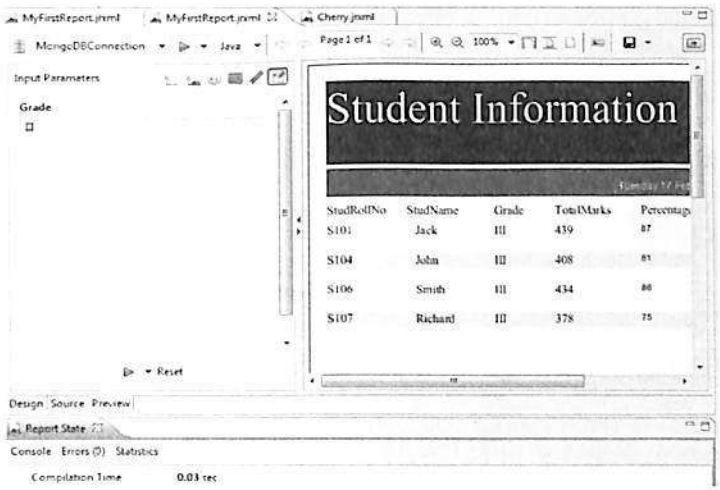
2. Once it is selected, its property will appear in the property window as shown below. Specify the name, data type, and description for the parameter and check **"Is For Prompting"** option.



3. Open the source tab and specify the parameter inside the queryString tag as shown below.

```
17    <queryString language="MongoDbQuery">
18        <![CDATA[{'collectionName' : 'students',
19                  'findQuery' : { 'Grade' : $P{Grade} } }]]>
20    </queryString>
```

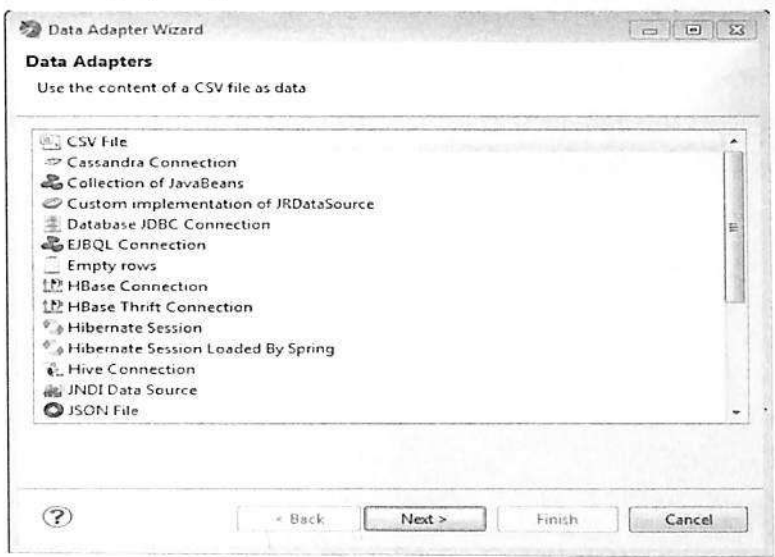
4. Click on the **"Preview"** tab. You will be prompted to enter the value for the "Grade" parameter as shown below. Click on the Run button (green arrow) to view the result.



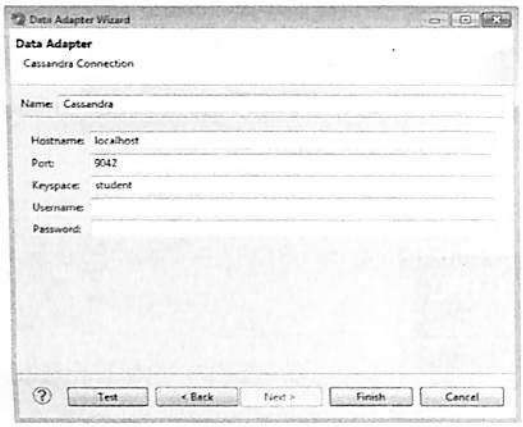
11.3 CONNECTING TO CASSANDRA NoSQL DATABASE

Jaspersoft studio uses Cassandra Query Language (CQL), which is similar to SQL, to retrieve data from Cassandra NoSQL database.

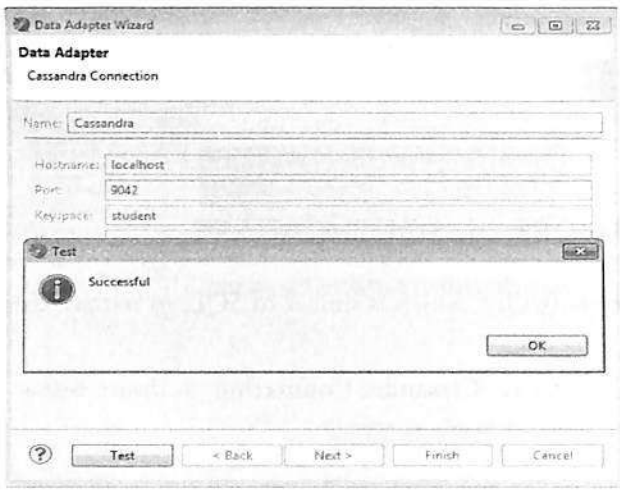
1. Create a JasperReports and state the data source as **"Cassandra Connection"** as shown below.



2. Click **"Next"** to get the Data Adapter Wizard. Mention the hostname, port, and keyspace as shown below.

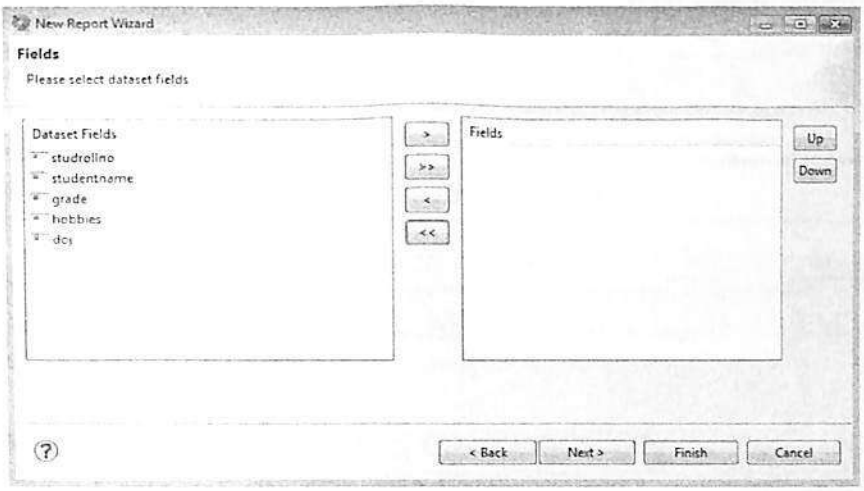


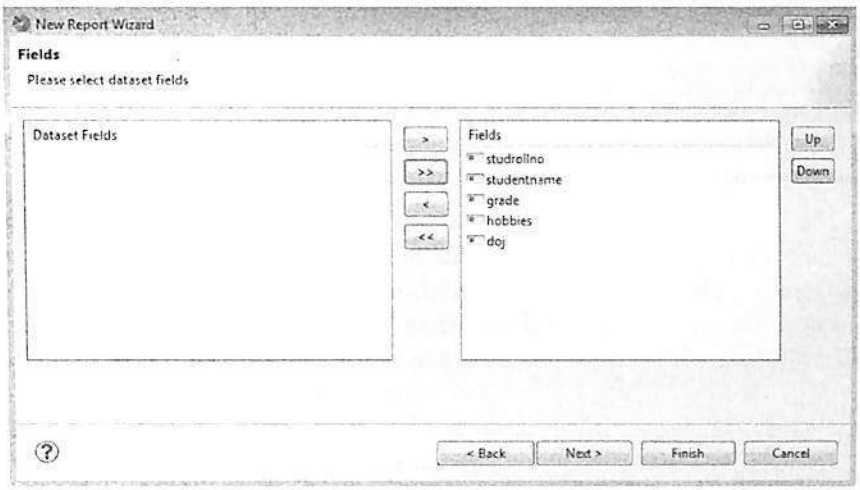
3. Click on the **"Test"** button to test the connection. If the connection is proper, you will get the **"Successful"** message as shown below.



4. Write the CQL (Cassandra Query Language) query to retrieve data from the Cassandra database.

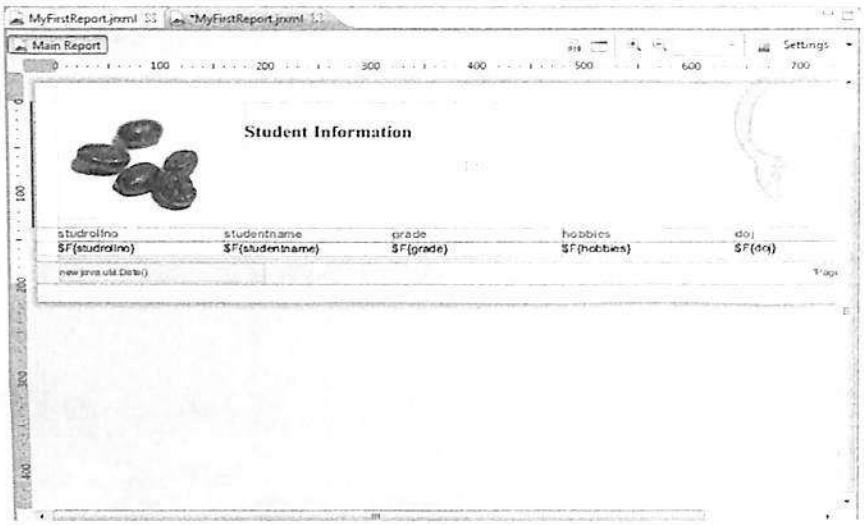
5. Select the required fields (under Dataset fields) for your report.

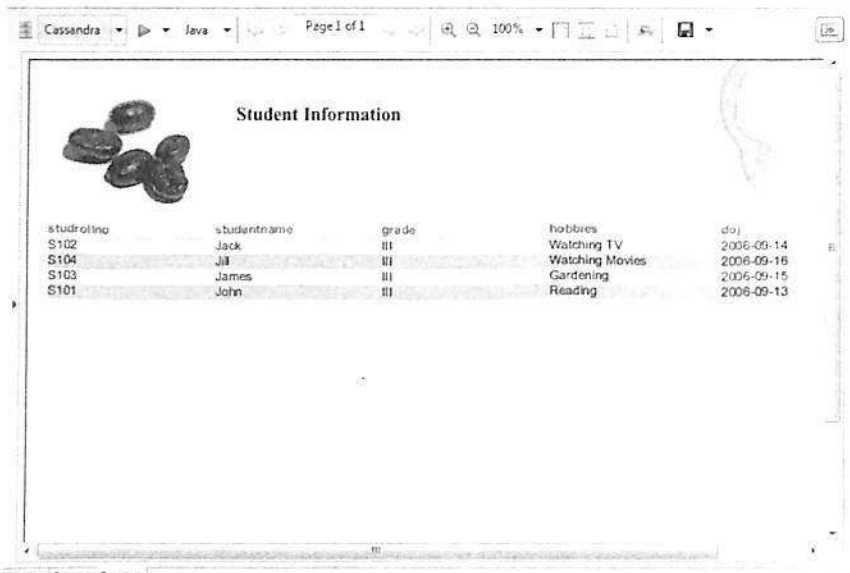




6. Click on **"Finish"** button to get the below message.

7. The report will open in design mode for editing. Create a report based on your requirement and view the report by clicking on the preview button.





REMIND ME

- Open-source reporting engine.
- Reports can be embedded in any Java Application.
- Jaspersoft studio is available in two flavors: Eclipse plugin and as a standalone application. Data can be accessed from multiple sources such as JDBC, XML, CSV, etc. Supports big data components such as MongoDB, Cassandra, Hive, etc.

POINT ME (BOOK)

- JasperReports for Java Developers by David R. Heffelfinger.

CONNECT ME (INTERNET RESOURCES)

- **https://community.jaspersoft.com/wiki/designing-report-jaspersoft-studio**
- **http://community.jaspersoft.com/wiki/jaspersoft-mongodb-query-language**

ASSIGNMENT FOR HANDS-ON PRACTICE

Background

You will be analyzing the scores and percentages of the trainees in various modules. You will be doing the analysis for scores and percentages in various assessments such as Test, Retest, Hands on, and/or Comprehensive Examination. Data for the assignment is available in JasperAssignment.accdb and TimeData.txt (see CD available with the book).

Database Section

Use either MongoDB or Cassandra.

Data for the assignment is provided on the CD (JasperAssignment.accdb and TimeDataForJasper Assignment.txt). Read the data from JasperAssignment.accdb into a .CSV or .TXT. Then from the .CSV or .TXT read the data into MongoDB or Cassandra.

Below are the table structures given in a standard RDBMS. Based on the requirements, create collections in MongoDB or tables in Cassandra.

Create a new database with name as 'JasperAssignment'. This database will include following tables. You are free to make names of table more meaningful.

1. Time (no need to create; load directly from the data provided on the CD).
2. Assessment (to be created).
3. Modules (to be created).
4. Trainees (to be created).
5. Score (to be created).

Table Details

Trainees Table:

Column Name	Data Type	Description
EmpKey	Int	Primary Key
EmpNumber	Int	Not Null
EmpName	Varchar(255)	Not Null

(*Continued*)

Column Name	Data Type	Description
BatchName	Varchar(50)	Not Null
Stream	Varchar(10)	Not Null
IBU	Varchar(4)	If Emp Number is less 100150 then assign SI else TRPU

Assessment Table:

Column Name	Data Type	Description
AssessmentKey	Int	Primary Key
AssessmentType	Varchar(100)	Not Null
DurationInMins	Int	Not Null

Modules Table:

Column Name	Data Type	Description
ModuleKey	Int	Primary Key
ModuleName	Varchar(100)	Not Null
ModuleCreditPoints	Int	Not Null

Score Table:

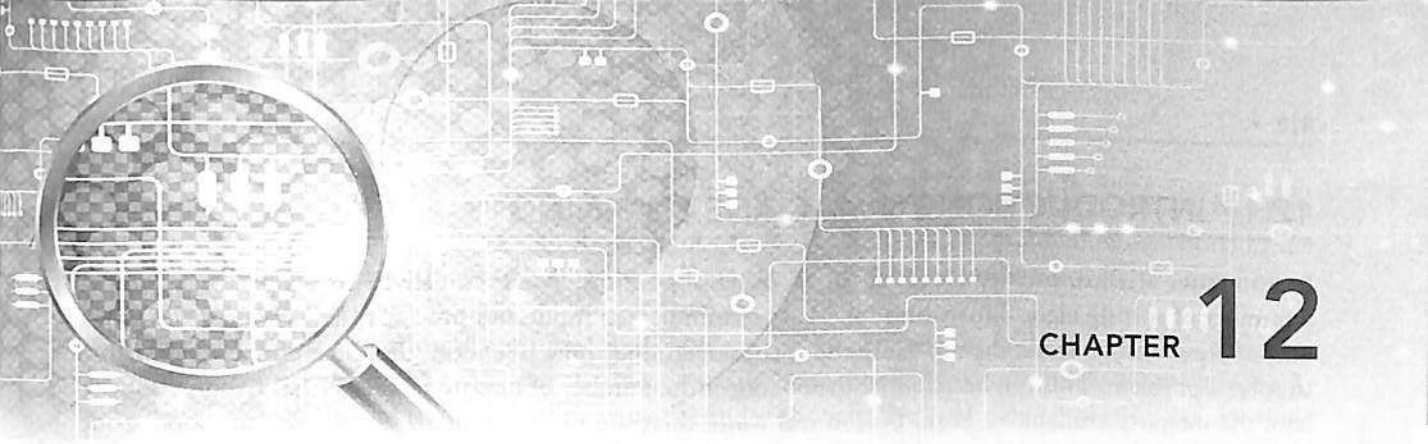
Column Name	Data Type	Description
ScoreKey	Int	Primary Key
AssessmentKey	Int	Not Null
ModuleKey	Int	Not Null
EmpKey	Int	Not Null
TimeKey	Int	Not Null
TotalScore	Int	Not Null
MaximumScore	Int	Not Null
Percentage	Int	Not Null

Reporting Section

Create the following report.

Chart Report:

- This Report has to be built by extracting data from 'JasperAssignment' database.
- Module Names on X-axis, Percentage Scored in the various modules on Y-axis. Use Bar chart.
- Take 'Emp Name' as input parameter. Take values of parameter from drop down list.
- Take 'Assessment Type' as Input Parameter with Available values as 'Test' and 'Retest' in drop down List.



CHAPTER 12

Introduction to Machine Learning

BRIEF CONTENTS

"If the statistics are boring, you have got the wrong numbers."

– Edward Tufte

WHAT'S IN STORE?

The focus of this chapter is to build knowledge about Machine Learning Algorithms. We will discuss supervised, unsupervised learning and few learning algorithms of these categories. We will also discuss implementation of Regression Model and K-means algorithms using **R** statistical tool.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning.

12.1 INTRODUCTION TO MACHINE LEARNING

In computer science, problems can be solved using algorithms. For example, to sort a set of numbers, a number of sorting algorithms which take a set of numbers as input and produce their ordered list as the output are available. Here, the most important consideration is how to choose the most efficient algorithm to solve a problem. This can be achieved by looking at the number of instructions essential to solve a problem, the memory available or both. But, in real world there are some problems which we can't solve using these kinds of algorithms. One such problem is filtering emails. In this case, the input is a file of characters and the output is to ascertain whether the email is spam mail or not.

To resolve this sort of problem, we can use machine learning algorithms. In learning algorithms, we utilize sample data, feed it to system and then train the system to produce the approximate model. For example, for email classification, we can compile thousands of example messages for spam and we can then make the system to learn to distinguish between a mail that qualifies as a spam message and the email that is NOT spam. In this situation, we may not be able to identify the process completely, but we can construct an approximate model. This approximate model may NOT be able to explain everything, but may be able to account for some part of it. This helps us to create certain patterns, which can then be used to understand the process or to make predictions. This is known as "**Machine Learning**". Machine Learning is one of the branches of Artificial Intelligence.

When we apply machine learning methods to large databases, it is known as **Data Mining**. This isn't just a database problem; it is also a part of Artificial Intelligence. The system should be intelligent enough to study the data in different environments.

Real-Time Example for Machine Learning

1. **Google Search Engine:** It works well because of the learning algorithm implemented by Google. This learning algorithm has learned how to rank web pages.
2. **Facebook:** When you use phototyping application, it is able to recognize your friend's photos because of machine learning algorithm.

12.1.1 Machine Learning Definition

Arthur Samuel (1959), Machine Learning: It is a field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998), Well-posed Learning Problem: A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Example

Tom's definition is the latest Machine Learning Definition. According to Tom's definition, we will try to identify E, P, T in Email Spam Classification Problem. In this problem,

Task – Classifying email as spam or not.
Experience – Labeling the email as spam or not.
Performance – Number of emails correctly classified as spam or not.

12.2 Machine Learning Algorithms

Machine Learning Algorithms can be classified into two categories.

1. **Supervised Learning:** In supervised learning, the classifier undergoes a process of training based on known classifications and through supervision it attempts to learn the information contained in the training dataset. Example: Predicting the selling price of the house based on pricing of other houses for sale in the neighborhood.
2. **Unsupervised Learning:** In unsupervised learning, the classifier tries to find some structure from the given data set without any known classification. That structure is known as Cluster. Example: Google News collects tens of thousands of news stories and automatically clusters them together. So that news stories that have the same content are displayed together.

12.2.1 Regression Model – Linear Regression

Regression Model is used to predict numbers. With the help of regression you can predict profit, sales, house values, etc. Regression Model serves as a good example for classification (supervised learning). Here, we will discuss *Linear Regression*.

Linear Regression is used to predict the relationship between two variables. The variable that needs to be predicted is known as the dependent variable and the variables that are used to predict the value of the dependent variable are known as independent variables.

12.2.1.1 Methodology

The general equation for Regression Model is as follows:

$$Y = a + b_1x_1 + b_2x_2 + \ldots + b_nx_n$$

Here, Y is the dependent variable; x_1, x_2, x_3 are the independent variables (these variables are used to predict the value of Y); b_1, b_2, …, b_n are the co-efficient of the respective independent variables (these values are determined from the input data); a is the intercept. Slope and intercept can be calculated using the below expression.

$$\text{Slope } (b) = [N\Sigma XY - (\Sigma X)(\Sigma Y)]/[N\Sigma X^2 - (\Sigma X)^2]$$
$$\text{Intercept } (a) = [\Sigma Y - b(\Sigma X)]/N$$

Example: Assume you wish to predict the housing price in Washington. Further assume that you have a dataset and you can plot the dataset as shown in Figure 12.1. Here, the horizontal axis indicates the sizes of different houses with the area specified in square feet and the vertical axis indicates the price of the houses

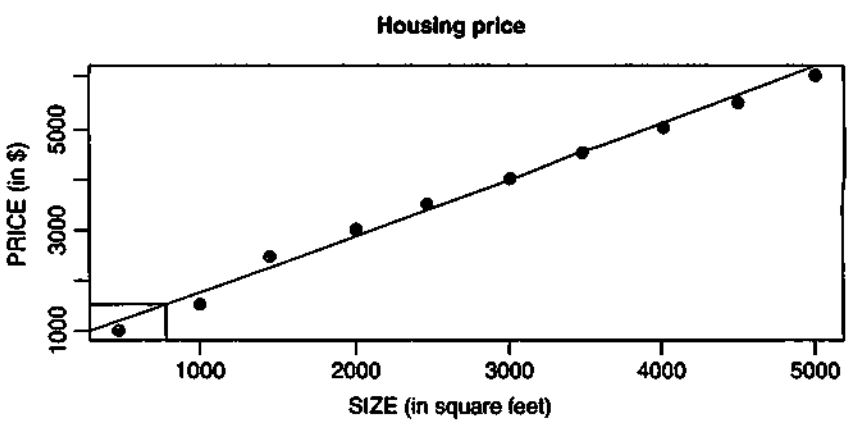


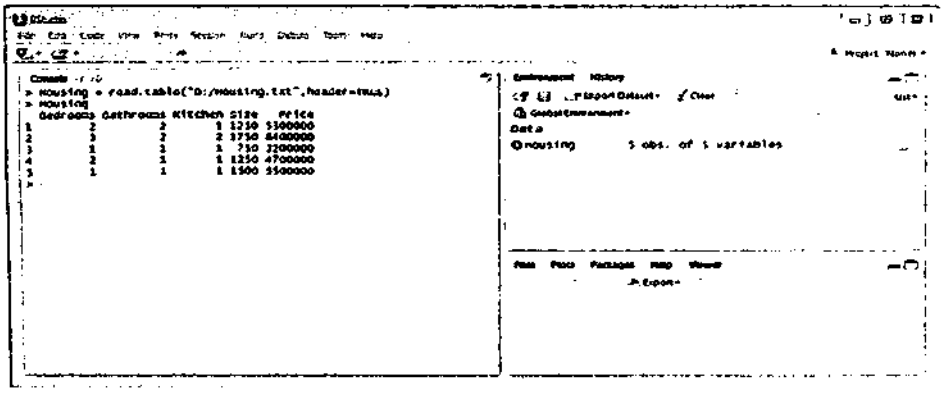
Figure 12.1 Housing price prediction.

in thousands of dollar. Let us say that you want to predict the price of house spread over 750 square feet. In this case, you can use learning algorithm to place a straight line through the data and based on that you can predict that you can get the house for about $150,000.

12.2.1.2 Implementation of Regression Model using R

R is a programming language used for data analysis. The following steps describe how to implement **"Housing Price Prediction"** using **RStudio.**

1. You can import data into the Environment as shown below. The name of the file is Housing.txt.



The following lines read data from the file "D:/Housing.txt" and prints the data on the console. Here "Housing" is a variable.

```
>Housing = read.table("D:/Housing.txt",header=TRUE)
>Housing
```

2. Let us consider all the attributes for predicting the price of a house. Construct a data frame as shown below. In **R**, the data frame is an array-like structure.

```
> myhouse = c("Bedrooms","Bathrooms","Kitchen","Size","Price")
> Housing1 = Housing[myhouse]
> Housing1
  Bedrooms Bathrooms Kitchen Size   Price
1        2         2       1 1250 5300000
2        3         2       2 1750 8400000
3        1         1       1  750 3200000
4        2         1       1 1250 4700000
5        1         1       1 1500 5500000
> |
```

3. To construct a multiple linear regression model with "houseprice" as the response variable and all the other attributes as the explanatory variables, use lm (linear model) command:

```
> houseprice = lm(Price ~ Bedrooms + Bathrooms + Kitchen + Size, da
ta=Housing)
> houseprice

Call:
lm(formula = Price ~ Bedrooms + Bathrooms + Kitchen + Size, data =
Housing)

Coefficients:
(Intercept)     Bedrooms    Bathrooms      Kitchen
   -1266667       -33333       600000      1600000
       Size
       3067

> |
```

4. To predict housing price, use predict command.

```
> predict(houseprice,data.frame(Size=1500, Bathrooms=2,Bedrooms=2,K
itchen=1))
       1
6066667
> |
```

The result displays the housing price prediction for a house whose size is 1500 square feet, 2 bedrooms, 2 bathrooms, and 1 kitchen.

Note: Price is in lakhs and Size is in square feet.

12.2.2 Clustering

Clustering is the process of grouping similar objects together. Clustering is an example of unsupervised learning. One can use clustering algorithms to segment data as in classification algorithms. However, classification models are used to segment data based on previously defined classes that are mentioned in the target, whereas clustering models do not use any target.

Clustering can be used to group items in a supermarket. For example, butter, cheese and milk can be placed in the "dairy products" group.

You can use clustering when you want to explore data. Clustering algorithms are mainly used for natural groupings. There are different categories of clustering. Refer Figure 12.2 for Clustering categories.

1. **Hierarchical:** Hierarchical cluster identifies the cluster within the cluster. A news article group can further have other groups such as business, politics, and sports in which each group can still have subgroups. For example, inside sports news there could be news on baseball sport, news on basketball sport, and so on.
2. **Partitional:** Partitional creates a fixed number of clusters. The K-means clustering algorithm belongs to this category. Let us study the K-mean clustering algorithm in detail.

12.2.2.1 K-Means

The steps involved in K-means algorithm are as follows:

1. Choose the final required number of clusters.
2. Examine each element in the population and assign it to one of the clusters depending on the minimum distance.
3. Each time a new element is added to the cluster, the centroid's position is recalculated. This process is performed until all the elements are grouped into the required number of clusters.

Centroid: It is a point whose parameter values are the mean of the parameter values of all the points in the cluster.

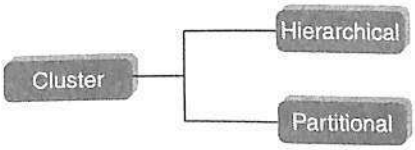
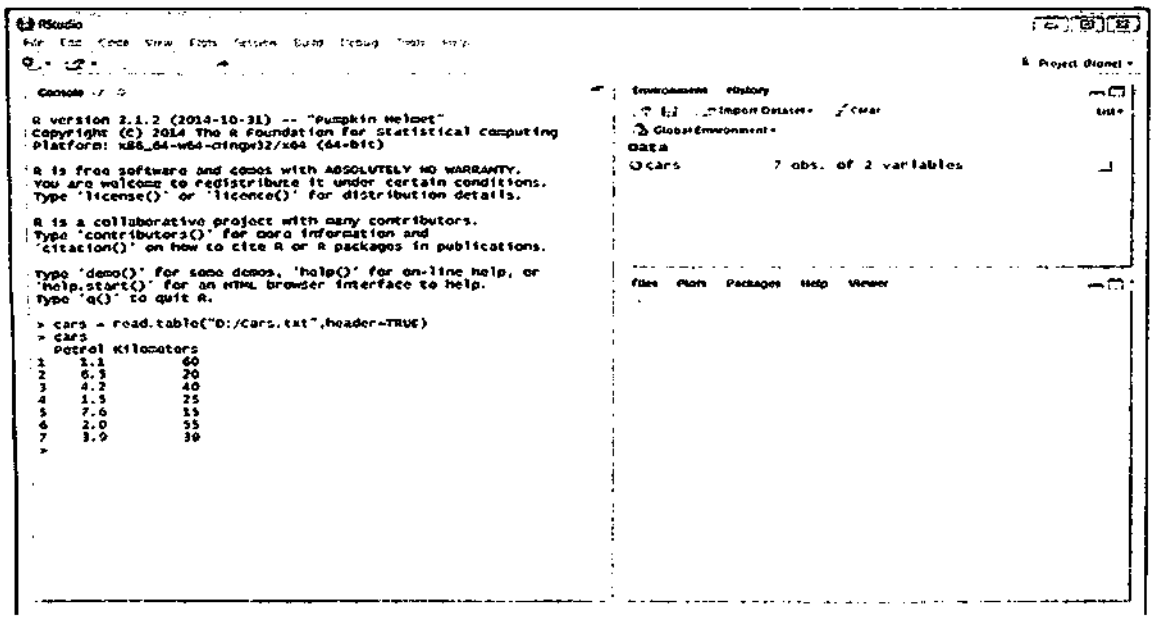


Figure 12.2 Categories of clustering.

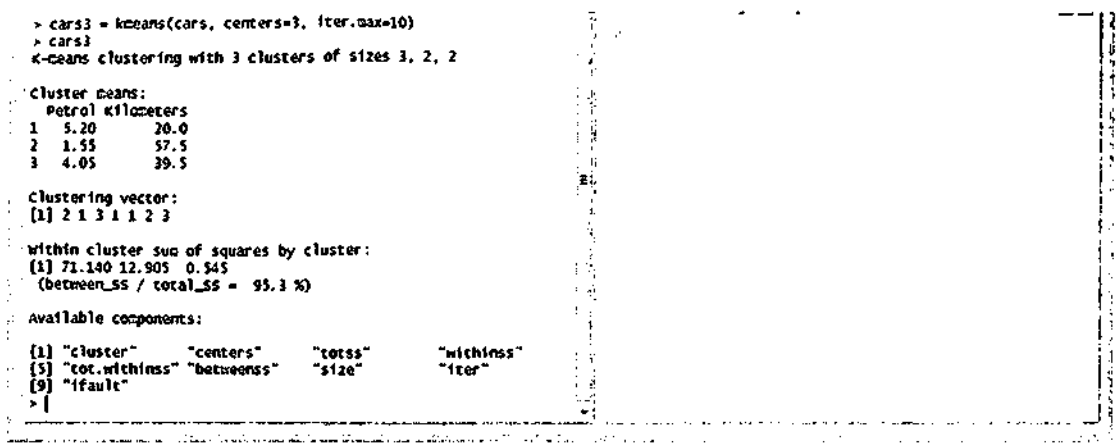
12.2.2.2 K-Means Algorithm Implementation using R

Let us discuss implementation of K-means clustering using R.

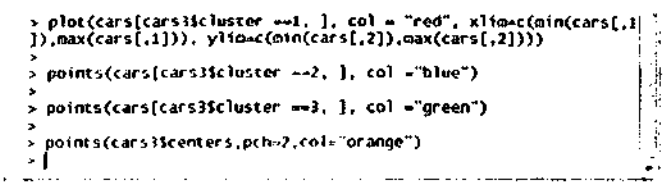
1. You can import data into the Environment as shown below. The name of the file is Cars.txt. This file contains entry for Petrol cars and their corresponding mileage in kilometers.

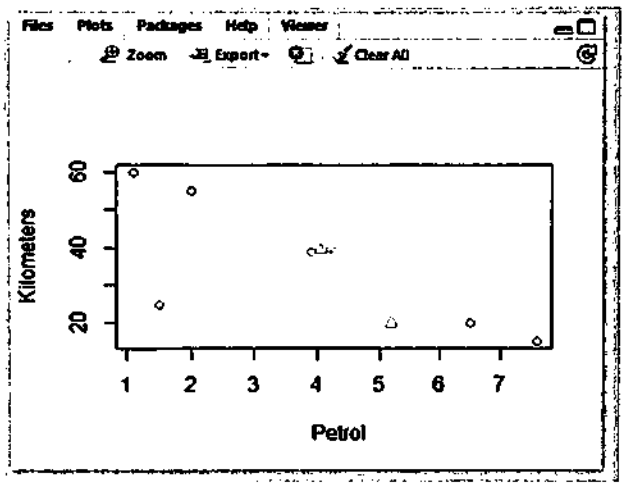


2. Apply K-means algorithm as shown below. The data set is split into 3 clusters and the maximum iteration is 10.



3. Next, you can plot clusters as shown below:





12.2.3 Collaborative Filtering

Collaborative filtering is a technique used for recommendation. Let us assume there are two people A and B. A likes Apples and B also likes Apples. In this case, we can assume that B has similar liking as A. So we can go ahead and recommend options for A to B as well.

12.2.3.1 History of Collaborative Filtering

The history of collaborative filtering started with Information Retrieval and Information Filtering.

Information Retrieval

The information retrieval era was from 1960s to 1980s. It is about retrieving information based on the queries/questions and these contents are mostly static in nature. Indexes, if built, help with the retrieval of information. For example, a collection of books can be indexed by title, author, and summary specified for the book. However, information requirements do not stay the same and change from time to time. This kind of information is known as dynamic content. An example of dynamic content is Google Search Engine, which provides us with dynamic content based on our search criteria.

Information Filtering

The exponential growth of Web has led to the explosion of information. So we require some technique to reduce the information overload and sieve information that is relevant to the users. This is then utilized to build long-term profile of the users' needs. Email Filtering (Filtering Spam Messages) is an example for information filtering.

Collaborative Filtering

Collaborative filtering is a category of information filtering. It is nothing but predicting user preferences based on the preferences of a group of users. You can use collaborative filtering when information needs are more complex than keywords or topics. Collaborative filtering is concerned with quality and taste.

Collaborative filtering can be defined as **Social Navigation.** We say that human beings are social animals owing to their tendency to follow other people's advice or judgment when looking for information or buying products. This is in a way similar to an ant looking for food trudging behind other ants.

12.2.3.2 Algorithms of Collaborative Filtering

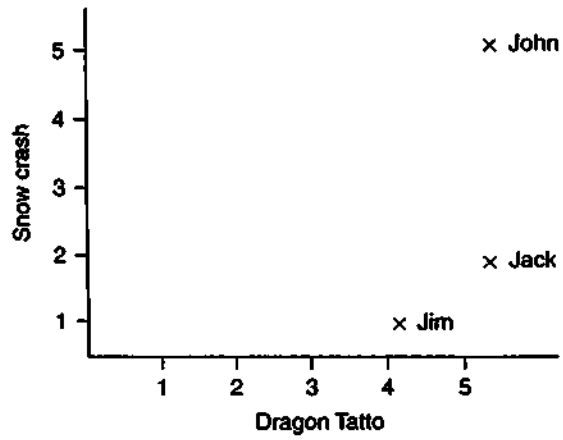
Let us discuss few of the collaborative filtering algorithms. In collaborative filtering, the important concern is "How to find someone who is similar?" It involves two steps:

1. **Collecting Preferences**

	Snow Crash	Dragon Tattoo
John	5★	5★
Jack	2★	5★
Jim	1★	4★

2. **Finding Similar Users:** Any of the following algorithms can be used to find similar users.
   - Euclidean Distance Score
   - Manhattan Distance or Cab Driver
   - Pearson–Correlation Co-efficient

Start by plotting the data as shown below. Here, X represents the Dragon Tatto and Y represents the Snow Crash.



Euclidean Distance: The source of Euclidean distance is Pythagorean Theorem:

$$c = \sqrt{a^2 + b^2}$$

The distance between two points in the plane with coordinates (x, y) and (a, b) is given by

$$\text{dist}((x, y), (a, b)) = \sqrt{(x-a)^2 + (y-b)^2}$$

Example:

$$\text{dist}((2,-1), (-2, 2)) = \sqrt{[2-(-2)]^2 + [(-1)-2]^2} = \sqrt{(2+2)^2 + (-1-2)^2}$$
$$= \sqrt{(4)^2 + (-3)^2} = \sqrt{16+9} = \sqrt{25} = 5$$

The distance between Ms. Y (a person) and John is given in Figure 12.3.

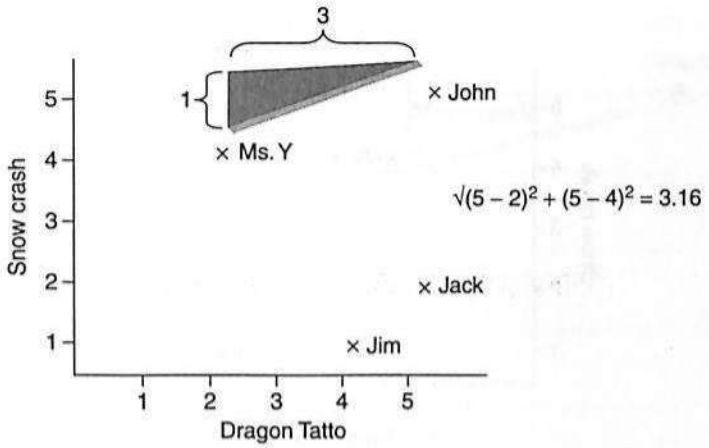


Figure 12.3 Euclidean distance between Ms. Y and John.

The distance between Ms. Y and all the three people is given below.

	Distance from Ms. Y
John	3.16
Jack	3.61
Jim	3.61

Note: For *N*-dimensional thinking refer the book specified in the Point Me Section.

Manhattan Distance or Cab Driver Distance: Here each person is represented by x and y.

$$(x_1, y_1) \Rightarrow \textbf{John and } (x_2, y_2) \Rightarrow \textbf{Ms. Y}$$

Formula to calculate Manhattan Distance for 2D is

$$|x_1 - x_2| + |y_1 - y_2|$$

Manhattan distance for John and Ms. Y is 4 (Figure 12.4).

So the distances between Ms. Y and all three people are as below:

	Distance from Ms. Y
John	4
Jack	5
Jim	5

John is the closest match. Manhattan distance is fast to compute. It is suitable for applications like Facebook to find another almost similar user amongst the millions of users.

Note: For *N*-dimensional thinking refer the book specified in the Point Me Section.

But the users have different behavior when it comes to their rating.

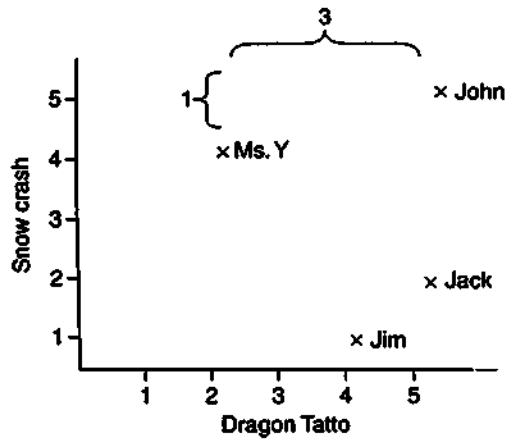


Figure 12.4 Manhattan Distance between Ms. Y and John.

		Bill's ratings is between 2 and 4			Hailey's ratings is between 1 and 4			
	Angelica	**Bill**	**Chan**	**Dan**	**Hailey**	**Jordyn**	**Sam**	**Veronica**
Blues Traveler	3.5	2	5	3	–	–	5	3
Broken bells	2	3.5	1	4	4	4.5	2	–
Deadmau5	–	4	1	4.5	1	4	–	–
Norah Jones	4.5	–	3	–	4	5	3	5
Phoenix	5	2	5	3	–	5	5	4
Slightly Stoopid	1.5	3.5	1	4.5	–	4.5	4	2.5
The Strokes	2.5	–	–	4	4	4	5	3
Vampire Weekend	2	3	–	2	1	4	–	–

Problem: How do we compare the similarities?

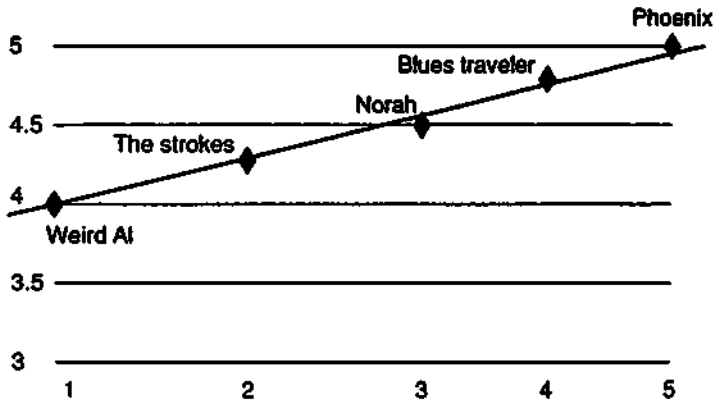
For example, does Hailey's "4" mean the same as Jordan's "4" or Jordan's "5"? The variability in this can create problems in the recommendation system.

The solution to this is **Pearson–Correlation Co-efficient**.

Pearson–Correlation Co-efficient

	Blues Traveler	**Norah Jones**	**Phoenix**	**The Strokes**	**Weird Al**
Clara	4.75	4.5	5	4.25	4
Robert	4	3	5	2	1

When we plot the chart, the data appears as below:



Here, the straight line indicates perfect agreement.

The formula to calculate PCC (Pearson–Correlation Co-efficient) is as stated below:

$$r = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}}{\sqrt{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}} \sqrt{\sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}}}$$

Return ranges are between 1 and −1. 1 means perfect agreement and −1 implies disagreement. Let's first compute the expression in the numerator:

$$(4.75 \times 4) + (4.5 \times 3) + (5 \times 5) + (4.25 \times 2) + (4 \times 1) = 19 + 13.5 + 25 + 8.5 + 4 = 70$$

Proceeding further, let's compute the remaining part of the numerator:

Sum of Clara's ratings is 22.5.

Sum of Robert's ratings is 15.

They rated 5 bands, therefore

$$22.5 \times 15/5 = 67.5$$

Numerator value is: 70 – 67.5 = 2.5

Let us further compute the denominator:

Step 1:

$$(4.75)2 + (4.5)2 + (5)2 + (4.25)2 + (4)2 = 101.875$$

Step 2: Sum of Clara's ratings is 22.5. When we square that, we get 506.25. Divide this number by the number of co-rated bands (5) and we get 101.25. Putting it all together:

$$\sqrt{101.875 - 101.25} = \sqrt{0.625} = 0.79057$$

Similarly compute it for Robert:

$$\sqrt{55 - 45} = 3.162277$$

Putting it all together into the PCC equation gives us the following:

$$r = 2.5/0.79057\ (3.162277) = 2.5/2.5 = 1.00$$

So there is a perfect match between Clara and Robert.

Which Similarity Measure to use when?

1. **Pearson:** Use Pearson when different users use different scales.
2. **Euclidean or Manhattan:** Use Euclidean or Manhattan if you have values for all the attributes.

Note: You can use R to implement Collaborative Filtering Algorithms.

12.2.4 Association Rule Mining

Association rule mining is also referred to as market basket analysis. Few also prefer to call it as affinity analysis. It is a data analysis and data mining technique. It is used to determine co-occurrence relationship among activities performed by individuals and groups.

Examples:

1. It is widely used in retail wherein the retailer seeks to understand the buying behavior of customer. This insight is then used to cross-sell or up-sell to the customers.
2. If you have ever bought a book from Amazon, this should sound familiar to you. The moment you are done selecting and placing the desired book in the shopping cart, pop comes the recommendation stating that customers who bought book "A" also bought book "B".
3. Who can forget the urban legend, the very famous beer and diapers example. The legend goes... there was a retail firm wherein it was observed that when diapers were purchased beer was purchased as well by the customer. The retailer cashed in on this opportunity by stocking beer coolers close to the shelves that housed the diaper. This just to make it convenient for the customers to easily pick both the products.

An association rule has two parts (a) an antecedent (if) and (b) a consequent (then). An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent.

PICTURE THIS...

Retailer "BigDailies" wants to cash in on their customers buying patterns. They want to be able to enact targeted marketing campaigns for specific segments of customers. They wish to have a good inventory management system in place. They wish to learn about which items/products should be stocked together to provide ease of buying to customers, in other words, enhance customer satisfaction.

Where should they start? They have had some internal discussions with their sales and IT staff. The IT staff has been instructed to design an application that can house each customer's transaction data. They wish to have it recorded every single day for every single customer and for every transaction made. They decide to meet after a quarter (3 months) to see if there is some buying pattern.

Table 12.1 illustrates a subset of the transaction data collected over a period of three months.

The table presents an interesting methodology called association analysis to discover interesting relationship in large datasets. The unveiled relationship can be presented in the form of association rules or sets of frequent items. For example, the following rule can be extracted from the above dataset:

$$\{\text{Diapers}\} \rightarrow \{\text{Beer}\}$$

It is pretty obvious from the above rule that a strong relationship exists between the sale of diapers and beer. Customers who pick up a pack or two of diapers also happen to pick a few cans of beers. Retailers can leverage these sort of rules to partake of the opportunity to cross-sell products to their customers.

Table 12.1 Sample transactional dataset

Transaction ID	Transaction details
1	{bread, milk}
2	{bread, milk, eggs, diapers, beer}
3	{bread, milk, beer, diapers}
4	{diapers, beer}
5	{milk, bread, diapers, eggs}
6	{milk, bread, diapers, beer}

Challenges that need to be addressed while progressing with association rule mining are as follows:

1. The larger the dataset, the better would be the analysis results. However, working with large transactional datasets can be and is usually computationally expensive.
2. Sometimes few of the discovered patterns could be spurious or misleading as it could have happened purely by chance or fluke.

12.2.4.1 Binary Representation

Let us look at how we can represent the sample dataset in Table 12.1 in binary format (Table 12.2).

Explanation of the binary representation in Table 12.2: Each row represents a transaction identified by a "Transaction ID". An item (such as bread, milk, eggs, diapers, and beer) is represented by a binary variable. A value of 1 denotes the presence of the item for the said transaction. A value of 0 denotes the absence of the item from the said transaction. Example: for transaction ID = 1, bread and milk are present and are depicted by 1. Eggs, diapers, and beer are absent from the transaction and therefore denoted by zero. The presence of the item is more important than its absence, and for the same reason an item is called as an asymmetric variable.

12.2.4.2 Itemset and Support Count

Let $I = \{i_1, i_2, i_3, \ldots, i_n\}$ be the set of all items in the market basket dataset.
Let $T = \{t_1, t_2, t_3, \ldots, t_n\}$ be the set of all transactions.

Itemset: Each transaction t_i contains a subset of items from set I. A collection of zero or more items is called an itemset. If an itemset contains k elements, it is called a k-item itemset. Example: the itemset {Bread, Milk, Diapers, Beer} is called a 4 itemset.

Table 12.2

Transaction ID	Bread	Milk	Eggs	Diapers	Beer
1	1	1	0	0	0
2	1	1	1	1	1
3	1	1	0	1	1
4	0	0	0	1	1
5	1	1	1	1	0
6	1	1	0	1	1

Transaction width: Transaction width is defined as the number of items present in the transaction. A transaction t_j contains an itemset X if X is a subset of t_j. Example transaction t_6 contains the itemset {bread, diapers} but does not contain the itemset {bread, eggs}.

Item support count: Support is an indication of how frequently the items appear in the dataset. Item support count is defined by the number of transactions that contain a particular itemset. Item support count can be expressed as follows: *Number of transactions that contain a particular itemset.*

Example: Support Count for {Diapers, Beer} is 4.

Mathematically, for an item set X the support count $\sigma(X)$ can be expressed as

$$\sigma(X) = \left|\{t_i\} \middle| X \subseteq t_i,\ t_i \in T\right|$$

The symbol | - | denotes the number of elements in the set.

Association rule: It is an implication rule of the form $X \rightarrow Y$ where X and Y are disjoint items, that is, $X \cap Y = \phi$. Support and confidence are the two factors that are utilized to get to the strength of the association rule mining.

The Support for an itemset is defined as follows:

Support $(x_1, x_2, \ldots)$ = Number of transactions containing $(x_1, x_2, \ldots)$/Total number of transactions (n)

Support for X → Y = Number of transactions containing $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ /n (total number of transactions)

Example:

Support for {Milk, Diapers} → {Beer} as per the dataset in Table 12.1 is as follows:

Support for {Milk, Diapers} → {Beer} = 3/6 = 0.5

Confidence of the rule is

Confidence of $(x_1, x_2, \ldots)$ implies $(y_1, y_2, \ldots)$ = Support for $(x_1, x_2, \ldots)$ implies $(y_1, y_2, \ldots)$/Support for $(x_1, x_2, \ldots)$

Confidence of {Milk, Diapers} → {Beer} = Support for {Milk, Diapers} → {Beer}/Support for {Milk, Diapers}

Substituting, the actual values, we get

Confidence of {Milk, Diapers} → {Beer} = 0.5/Support for {Milk, Diapers} = 0.5/0.67 = 0.7462

Why should you consider support and confidence?

It is vital to consider support and confidence owing to the following reasons: A rule which has low support may occur simply by chance. To place big bets on it may prove futile. If you deliberate from the business perspective, it may prove to be rather unexciting and non-lucrative purely due to the fact that it does not make sense to promote items that customers seldom buy together. Support is used to chuck off uninteresting rules.

Confidence is a measure of reliability of inference of an association rule. For a given rule $X \rightarrow Y$, the higher the confidence the more likely it is for Y to be present in transactions that contain X. Confidence of a rule can also be used to provide an estimate of the conditional probability of Y given X.

The results of association rule analysis should be considered astutely and judiciously. It does not necessarily imply causality. For causality to be in effect, it requires knowledge about the causal and effect attributes

in the data. It requires the relationship to be observed, recorded, and studied over a period of time. For example: Ozone depletion leads to global warming. Think association rule mining, and think establishing co-occurrence relationship between items in the antecedent and consequent of the rule.

Note: You can use R to implement Association Mining Algorithm.

12.2.5 Decision Tree

PICTURE THIS...

It is that time of the year again. The college fest is going to be next week. It is going to be a week long affair. Your friends have started planning on the kind of stalls that they will put up. You too want to try out this stall thing. However, you are yet to decide on what stall you should go for. You have to communicate your decision to the organizing committee in a day's time. The time is short. The decision has to be made quickly. You do not want to end up with a wrong decision. It is your first time at putting up a stall. You want to go for maximum profit.

You have zeroed down your choice to either an ice-cream stall or a burger stall from a gamut of choices available. How about using a decision tree to decide on the same? Let us look at how we can go about creating a decision tree.

Decision to be made: Either an ice-cream stall or a burger stall.

Payoff: 3500 INR in profit if you put up a burger stall and a 4000 INR in profit if you put up an ice-cream stall.

12.2.5.1 What are the uncertainties?

There is a 50% chance of you succeeding to make profit with a burger stall and a 50% chance of you failing at it.

As per the weather forecast, it will be downcast sky and may drizzle or pour slightly throughout the week. Keeping this into consideration, there is a 40% chance of success and 60% chance of failure with an ice-cream stall.

Let us look at the cost of the raw materials:

For Burger: 700 INR for the burger buns, the fillings, and a microwave oven to keep it warm.

For Ice-creams: 1000 INR for the cone, the ice-cream, and a freezer to keep it cold. Refer Figure 12.5.

Let us compute the effective value as per the below formula:

$$\text{Expected value for burger} = 0.5 \times 3500 \text{ INR} - 0.5 \times 700 \text{ INR} = 1400 \text{ INR}$$

$$\text{Expected value for ice-cream} = 0.4 \times 4000 \text{ INR} - 0.6 \times 1000 \text{ INR} = 1000 \text{ INR}$$

The choice is obvious. Refer Figure 12.6. Going by the expected value, you will gain by putting up a burger stall.

The expected value does not imply that you will make a profit of 1400 INR. Nevertheless, this amount is useful for decision-making, as it will maximize your expected returns in the long run if you continue to use this approach.

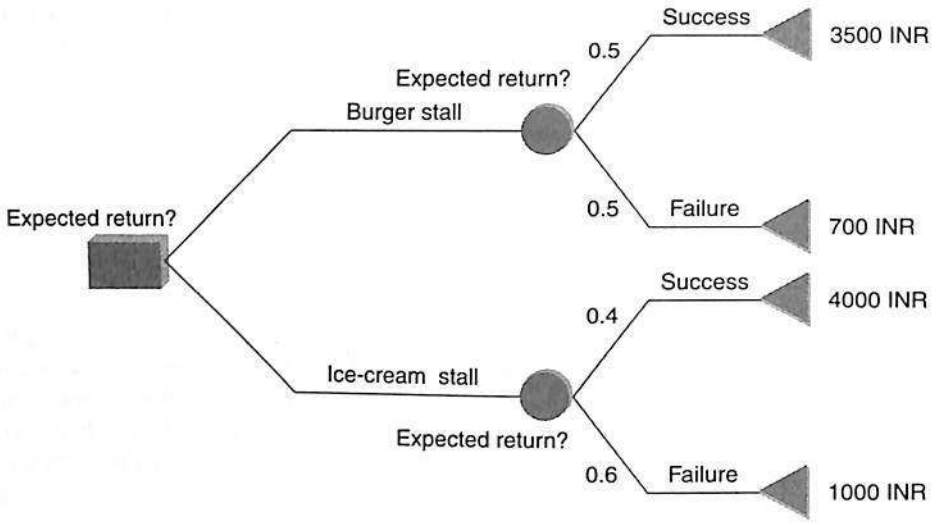


Figure 12.5 A sample decision tree with expected return value yet to be arrived at.

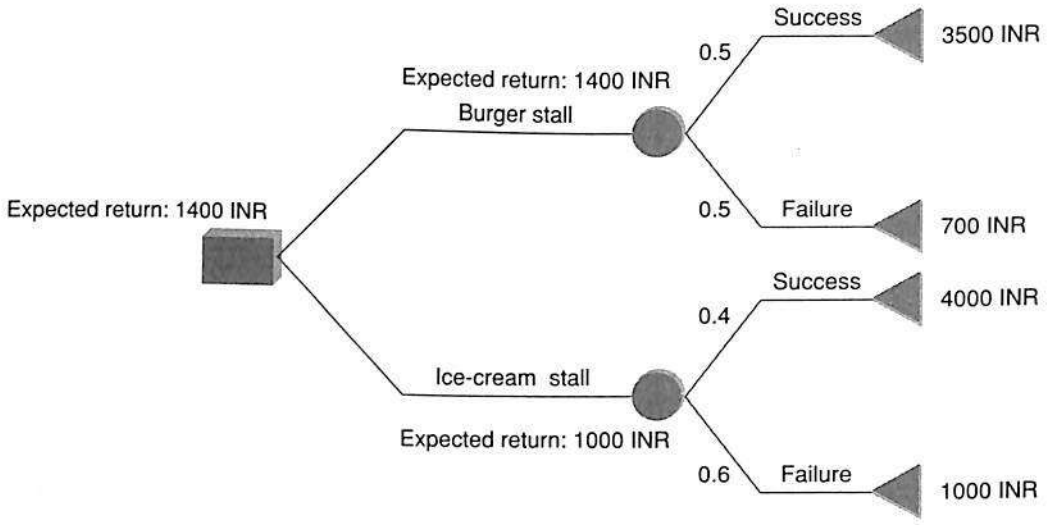


Figure 12.6 A sample decision tree with computed expected return.

PICTURE THIS...

You have just completed writing the script of a romantic story. There are two takers for it. (a) The television network: they are interested in making a daily soap of it that will be telecast on prime time. (b) XYZ Movie Company: they have also shown interest. You are confused. Should you sell the rights to the TV Network or XYZ Movie Company?

The TV network payout will be a flat 500,000 INR. XYZ Movie Company will pay in accordance to the audience response to the movie.

Payouts and Probabilities

TV Network payout:
Flat Rate: 500,000 INR

XYZ Movie Company Payout:

Small Box Office: 250,000 INR
Medium Box Office: 600,000 INR
Large Box Office: 800,000 INR

Probabilities:

P (Small Box Office): 0.3
P (Medium Box Office): 0.5
P (Large Box Office): 0.2

For greater understanding, let us create a payoff table:

Decisions	Small Box Office	Medium Box Office	Large Box Office
Sign up with TV Network	500,000 INR	500,000 INR	500,000 INR
Sign up with XYZ Movie Company	250,000 INR	600,000 INR	800,000 INR
Probabilities	0.3	0.5	0.2

Refer Figure 12.7.

Let us compute the effective value as per the below formula:

Expected value for TV Network = $0.3 \times 500{,}000 + 0.5 \times 500{,}000 + 0.2 \times 500{,}000 = 500{,}000$ INR

Expected value for XYZ Movie Company = $0.3 \times 250{,}000 + 0.5 \times 600{,}000 + 0.2 \times 800{,}000 = 535{,}000$ INR

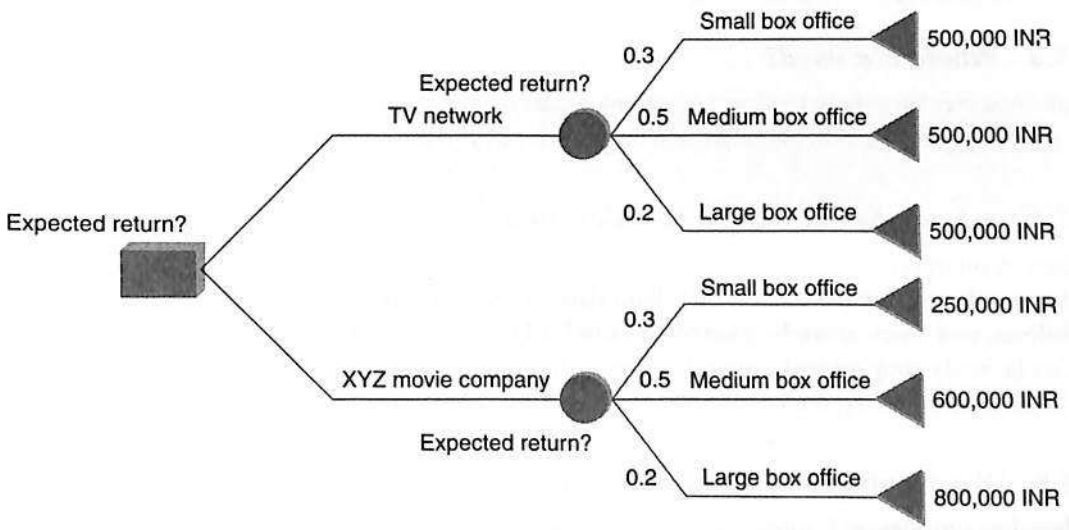


Figure 12.7 A sample decision tree with expected return value yet to be arrived at.

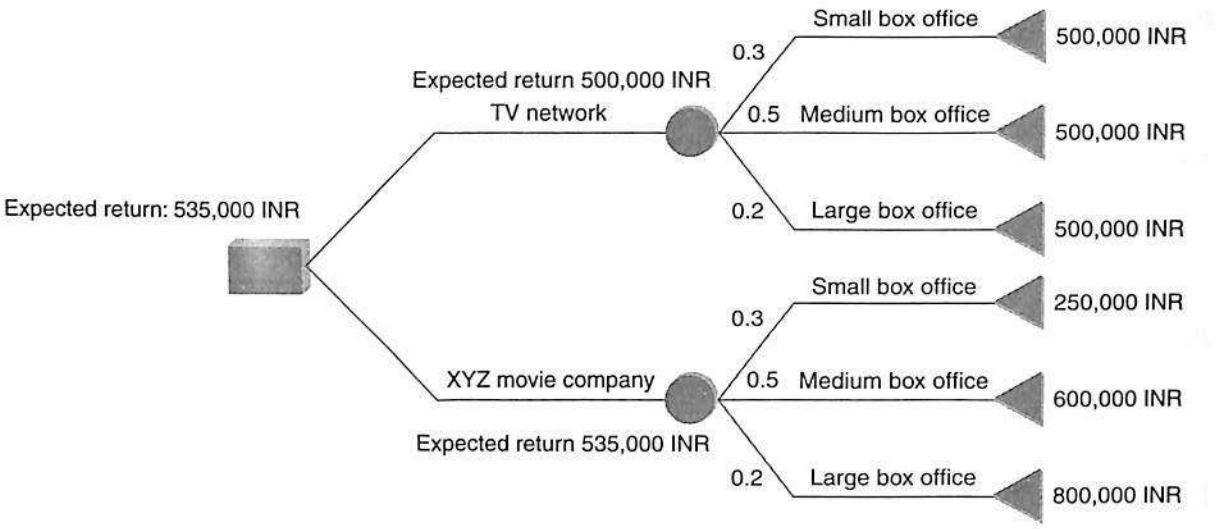


Figure 12.8 A sample decision tree with computed expected return.

The choice is obvious. Refer Figure 12.8. Going by the expected value, you will gain by selling the rights of your script to XYZ Movie Company. The expected value does not imply that you will make a profit of 535,000 INR.

12.2.5.2 What is a Decision Tree?

A decision tree is a decision support tool. It uses a tree-like graph to depict decision and their consequences. The following are the three constituents of a decision tree:

1. **Decision nodes:** Commonly represented by squares
2. **Chance nodes:** Represented by circles
3. **End nodes:** Represented by triangles

12.2.5.3 Where is it used?

Decision trees are commonly used in operations research, specifically in decision analysis. It is used to zero down on a strategy that is most likely to reach its goals. It can also be used to compute conditional probabilities.

12.2.5.4 Advantages of using a Decision Tree

1. Easy to interpret.
2. Easy to plot even when there is little hard data. If one is aware of little data such as alternatives, probabilities, and costs, it can be plotted and can lead to useful insights.
3. Can be easily coupled with other decision techniques.
4. Helps in determining the best, worst, and expected value for a given scenario or scenarios.

12.2.5.5 Disadvantages of Decision Trees

1. **Requires experience:** Business owners and managers should have a certain level of experience to complete the decision tree. It also calls for an understanding of quantitative and statistical analytical techniques.

2. **Incomplete information:** It is difficult to plot a decision tree without having complete information of the business and its operating environment.
3. **Too much information:** Too much information can be over-whelming and lead to what is called as the "paralysis of analysis".

REMIND ME

There are two types of Learning Algorithms:

- **Supervised Learning:** In this learning, classifier undergoes a process of training based on known classifications, and through supervision it attempts to learn the information contained in the training dataset. Regression Model is an example for supervised learning.
- **Unsupervised Learning:** In this learning, the classifier tries to find some structure from the given dataset without any known classification. That structure is known as Cluster. Clustering is an example for unsupervised learning.
- Collaborative filtering is an example for user based recommendation.
- Association rule mining is an example for item-based recommendation.
- Decision tree is a decision support system and used in operations research.

POINT ME (BOOK)

- A Programmer's Guide to Data Mining by Ron Zacharski.

CONNECT ME (INTERNET RESOURCES)

- https://www.coursera.org/course/ml: Coursera "Machine Learning" course from Stanford
- http://www.rstudio.com/resources/training/online-learning/

TEST ME

A. Fill Me

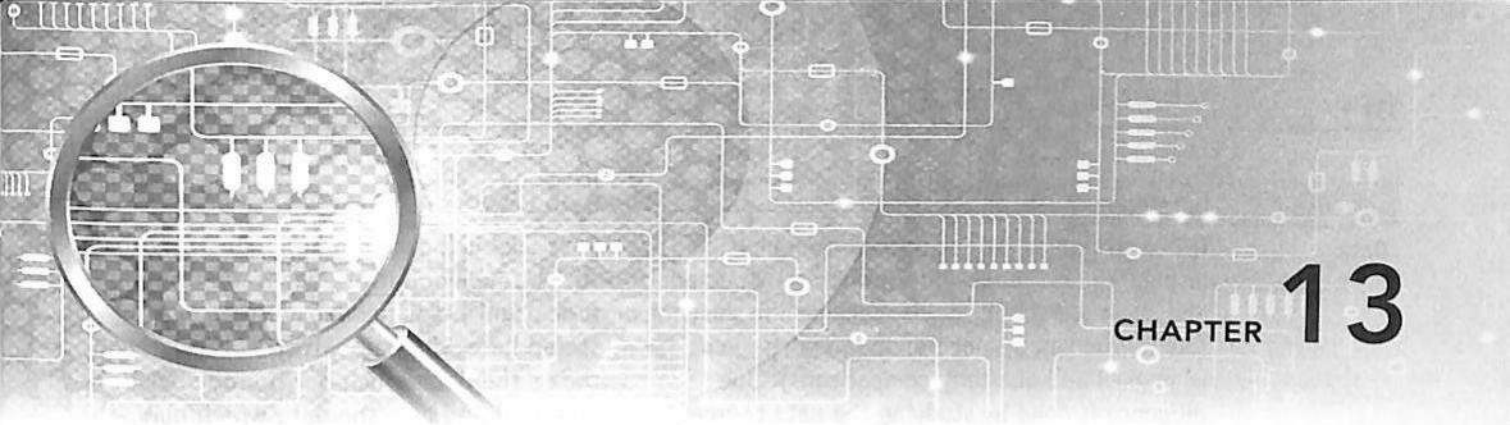
1. Decision tree is a _____ method.
2. R is _____ tool.
3. _____ is a user-based recommendation algorithm.
4. K-mean splits dataset in to _____ of Clusters.
5. Regression Model deals with _____.

Answers:

1. Supervised Learning
2. Statistical
3. Collaborative Filtering
4. Fixed number
5. Numeric Values

ASSIGNMENT FOR HANDS-ON PRACTICE

Write an R Program to implement Association Mining for frequently used item set. [Hint: You can construct your own dataset]



CHAPTER 13

Few Interesting Differences

BRIEF CONTENTS

- Difference between Data Warehouse and Data Lakes.
- Difference between RDBMS and HDFS.
- Difference between HDFS and HBase.
- Difference between MapReduce and Pig.
- Difference between MapReduce and Spark.
- Difference between Pig and Hive.

"Data is the new science. Big Data holds the answers."

– Pat Gelsinger

WHAT'S IN STORE?

The focus of this chapter is to bring out the differences between various Hadoop ecosystem components for easy lookup and remembrance. It will be good to read this chapter sequentially for a better absorption. Starting with data warehouse versus data lakes, it builds further to explain the differences between HDFS and RDBMS, then goes on to highlight differences of MapReduce with Pig, Spark, etc.

13.1 DIFFERENCE BETWEEN DATA WAREHOUSE AND DATA LAKE

Who coined the term data lake? It was Pentaho CTO, James Dixon. He described data mart (a subset of data warehouse) as akin to a bottle of water "cleansed, packaged and structured for easy consumption" while a data lake is more like a body of water in its natural state. Data flows from the streams (the source systems) to the lake.

	Data Warehouse	Data Lake
Data	It is a structured data extracted from multiple disparate data sources, integrated, transformed and made suitable to guide management decisions (allows the creation of trending reports such as manual and quarterly comparisons). Due diligence is done by studying the data sources, understanding business processes and profiling data before data is stored.	Data lake is a data storage repository to store huge amount of data in its original format until it is needed. All the data is stored here. The data which is needed today, the data which may be needed and used and also the data which may never be used basis it may be used someday.
Schema	Schema on write. Before data is written into the data warehouse, schema has to be compulsorily defined. In fact, data is not loaded into the warehouse until the use for it has been defined.	Schema on read. It stores raw data (the data that has not yet been processed for a purpose) in its original format. It is only at the time of reading that shape and structure need to be defined for the data.
Data types	It is structured data from transactional systems.	This includes data from web logs, sensor data, social media data, text and images.
Purpose	In-use clearly defined purpose.	Undetermined.
Cost associated with storing data	High cost storage.	Commodity, off-the-shelf servers with low-cost storage are used and therefore it helps to quickly scale from terabytes to petabytes to exabytes.
Agility	Data warehouses are less agile and have somewhat a fixed configuration.	Since data lakes lack a structure, it is easy to configure, reconfigure data models, queries and applications.
Data Security	Data warehouses have been around for decades and that has led to gaining maturity when it comes to securing data.	Data lakes are usually open-source, low-cost storage. Although significant strides are being made to secure data, it will still be some time before maturity sets in.
Users	The data in data warehouse is highly structured, purpose built and organized well therefore it is easy to use and comprehend. About 80% of the users in the organization are able to use the data to meet their operational needs like pull up reports, track their key performance metrics, etc. Few users (about 10%) need the data for more analysis. And the last 10% of the users – the data scientist tribe – will perform deeper analysis on the data. They usually tend to visit the original data sources (also a few new ones) in addition to the D/W.	Usually, data scientists who can play around with the data from varied sources.
Accessibility	More complicated and costly to make any changes to the structure, etc.	Highly accessible, quick to update.

13.2 DIFFERENCE BETWEEN RDBMS AND HDFS

	RDBMS	HDFS
Data Model	It is a relational database. Data are stored in tables comprising rows and columns.	It is not a database. It is a distributed clustered file system with support for parallelism and redundancy.
Data	Essentially used for structured data. Supports semi-structured data to a limited extent.	It is used for all varieties of data – structured, semi-structured and unstructured. It supports serialization and various file formats such as text, JSON, XML, Avro, etc.
Data Storage	Usually used for datasets (sizes in GBs, TBs).	Used for large datasets (sizes ranging in terabytes, petabytes, exabytes, etc.).
Querying	SQL (Structured Query Language).	Declarative language – HiveQL.
Schema	"Schema on Write". Mandates that the schema be pre-defined before placing data in it.	"Schema on Read". Stores data in its native format. It is only to be formatted at the time of reading data from it.
Speed	Reads are fast.	Both writes and reads are fast.
Cost	Available both as open-source (MySQL, PostgreSQL, etc.) as well as licensed ones such as Oracle, IBM-DB2, MS SQL Server, Teradata, etc.	Open-source and free (Apache Hadoop).
Usage	OLTP (Online Transaction Processing).	Data discovery and data analytics.
Throughput (throughput refers to the amount of data processed in a period of time)	Low	High
Scalability	Vertical (scale up). Scales by increasing the horsepower of the machine (CPU, RAM, Hard disk capacity, etc.).	Horizontal. Also called as linear scalability or scaling out. Scales by adding nodes to the cluster.
Hardware	High-end servers.	Commodity hardware.
Integrity	High. This is owing to strict adherence to the ACID (Atomicity, Consistency, Isolation/Integrity and Durability) properties of transactions.	Low. It supports BASE (Basically Available Soft state Eventual consistency).

13.3 DIFFERENCE BETWEEN HDFS AND HBase

	Hadoop HDFS	HBase
What is it?	Hadoop Distributed File System (HDFS) stores huge amounts and types of data in a distributed (provides faster read/write access) and redundant (provides better availability and fault tolerance) manner on industry standard commodity hardware.	It is an open-source, distributed, non-relational scalable NoSQL database that runs on top of the Hadoop cluster and provides a random real-time read–write access to the data. It is like a layer on top of HDFS. Using APIs, it is possible to write NoSQL queries and get the results. It is modeled after Google's BigTable.
Storage	Datasets are divided into smaller subsets called chunks/blocks and stored across clusters.	Data is stored in key–value pairs in column-oriented database that uses column families to group similar or frequently accessed data together.
Flexibility to read-write	Write once, Read many times.	Multiple read–write of data stored in HDFS.
Scalability	Multiple nodes can be added to the cluster and therefore hugely and infinitely scalable.	Can store huge amounts of data.
Analytics	Supports batch analytics. Batch analytics is high latency analytics whereby a huge volume of data is processed. There is latency as the data is collected, stored and then processed for analysis and reporting.	Supports batch and real-time analytics. Real-time analytics is low latency as the data from multiple disparate data sources and in any data format is filtered, aggregated, enriched and analyzed to help identify simple patterns, urgent situations, automate immediate actions etc.
Access to data	Only sequential access to data.	Random access to data.
Write Pattern	Append only.	Random write, bulk incremental.
Read Pattern	Full table scan.	Random read, small range scan or table scan.
Dynamic changes	Rigid architecture that does not allow changes.	Allows for dynamic changes and can be utilized for standalone applications.
Latency (how long does it take a file system/database to respond to a single request)	High latency operations.	Low latency access to small amounts of data from within a larger dataset.
Accessibility	Primarily accessed through MapReduce jobs.	Can be accessed through shell commands, client APIs in Java, REST, Avro or thrift.

13.4 HADOOP MapReduce VERSUS PIG

	Hadoop MapReduce	Pig
What is it?	• It is a low-level language. • It leads to a huge amount of custom user code that is difficult to maintain and reuse.	• It is a high-level scripting language. • It is a data flow language. • Pig is an application that runs atop MapReduce, YARN or Tez. • It is written in Java. • It converts (compiles) Pig Latin scripts into MapReduce jobs. • Approx. 10 lines of Pig code are equivalent to 200 lines of Java code. • Pig Latin scripts are easier to read for someone without a Java background.
Joins	Performing datasets joins is very difficult.	Pig helps easily with sort, parse, join, etc.
Extensibility	Not easy to extend. Functions need to be written from scratch.	Easily extendable using UDFs. The UDFs are fairly reusable.
Development cycle	It is fairly long comprising writing mappers, reducers – compiling, packaging the code, submitting the jobs, etc.	No need of compiling and packaging. The Pig operators are converted to MapReduce tasks internally.
Code Portability	May not be supported with all versions of Hadoop.	Works with any version of Hadoop.

13.5 DIFFERENCE BETWEEN HADOOP MapReduce AND SPARK

Spark can run standalone or on top of Hadoop YARN (Yet Another Resource Negotiator). In other words, Spark is not bound to Hadoop, although both Spark and Hadoop MapReduce are included in the distributions by Hortonworks and Cloudera.

	Hadoop MapReduce	Spark
Data storage	Stores data in local disk. This implies that data is written to and read from hard disk.	Stores data in memory.
Speed	Slow speed. However, MapReduce has given good results for ETL style jobs and transformations where it needs to pass through a piece of data only once.	Fast speed. This is primarily because data is stored in memory for processing. However, there could be major performance issues if the size of the program is too big to fit into memory. The performance issue is compounded if Spark is running on top of YARN along with other resource demanding services. Spark has been known to perform well for iterative computations that go over the same data many times.

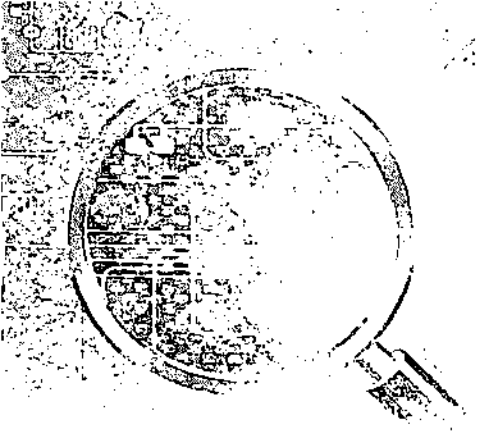
	Hadoop MapReduce	Spark
Real Time	Suitable for batch processing.	Suitable for batch and real-time processing. Spark can process and analyze data the moment it is captured and insights fed back to the user for appropriate action.
Scheduler	External schedulers such as Oozie are required.	Schedules tasks itself.
Latency	High latency.	Low latency.
Interactivity	No in-built interactive mode although Hive has a command line interface.	Has interactive mode that helps to run commands with immediate feedback.
Cost	Less expensive hardware.	Lots of RAM required for in-memory processing; increasing it in the cluster gradually increases its cost.
Difficult level	Difficult to program. We need to code/handle each process.	Fairly easy to program with the availability of RDD (Resilient Distributed Dataset). RDDs provide fault tolerance to Spark.
Platform developed on	Developed using Java.	Has APIs for Python, Java Scala and Spark SQL.
SQL support	MapReduce runs queries using Hive Query Language.	Spark has its own query language called Spark SQL.
Machine Learning	Supports Apache Mahout tool for machine learning.	Has its own machine learning library called MLib.
Caching	MapReduce does not cache in memory data so it is not as fast as Spark.	Spark caches in memory data, therefore it is very apt for iterative analytics.
Language Supported	MapReduce basically supports C, C++, Ruby, Groovy, Perl, and Python.	Spark supports Scala, Java, Python, R, and SQL.
Security	Hadoop supports Kerberos and LDAP for authentication. It also has support for traditional file permissions as well as ACLs (Access Control Lists).	Spark security is not as mature as Hadoop. However, Spark can integrate with HDFS and use HDFS ACLs and file level permissions.

Components in Hadoop Ecosystem and Their Equivalent in Spark

Hadoop	Spark
Hive	Spark SQL
Apache Graph	GraphX
Impala	Spark SQL
Apache Storm	Spark Streaming
Apache Mahout	MLib

13.6 DIFFERENCE BETWEEN PIG AND HIVE

	Pig	Hive
Language used	Pig has a procedural scripting language called Pig Latin for describing operations like reading, writing, filtering, transforming, merging and joining data. These are largely the operations that were performed using MapReduce. The difference being now it can be done without having to write thousands of lines of Java code. Pig is "Scripting for Hadoop".	Hive has a declarative language called HiveQL. Hive is "SQL for Hadoop". It basically is used for data summarizations, ad-hoc querying and analysis of large datasets.
Data	Works with both structured and semi-structured data.	Essentially works with structured data.
Who uses?	Researchers and Programmers	Data Analysts
Operates on	Client side of the cluster.	Server side of the cluster.
Support for Avro file format	Supports Avro file format.	Does not have support for Avro file format. However, with the evolution of Serge, support for Avro file format can be provided.
Developed at	Yahoo	Facebook
Partitioning and Bucketing	Does not support partitioning and bucketing.	Makes use of partitioning and bucketing.
Speed of loading and execution	Faster than Hive.	Slower than Pig.
Usage	It can be suitably used for prototyping and rapidly developing MapReduce jobs.	It can be used to perform ad hoc queries or generate report data spread across multiple nodes in the cluster.
Schema	Hive defines tables schema + stores schema information in database.	Pig does not have dedicated metadata of database.



CHAPTER 14

Big Data Trends in 2019 and Beyond

BRIEF CONTENTS

- What's in Store?
- Rise of the New Age "Data Curators"
- CDOs are Stepping up
- Dark Data in the Cloud
- Streaming the IOT for Machine Learning
- Edge Computing
- Open Source
- Hadoop is Fundamental
- Chat bots will get Smarter
- Container(ed) Revolution
- Visualization Commoditization

"Information is the oil of the 21st century, and analytics is the combustion engine."

– Peter Sondergaard, Senior Vice President, Gartner

WHAT'S IN STORE?

This chapter discusses the big data trends in 2019 and beyond. The years ahead will see an increase in the adoption of open-source technologies. Hadoop is and will remain fundamental although there will be increased usage of the in-memory Spark. The years ahead will also awake the container(ed) revolution. The last half-a-decade has been witness to the commoditization of visualization. The rising wave of IoT (Internet of Things) will lead to processing being done on the edge of the network before moving it to the central data center in the cloud. The world will witness the power of empowered computing – edge and quantum. It is time to utilize and draw value/insight from the abundant dark data. Also bots will mature and get smarter in the coming years.

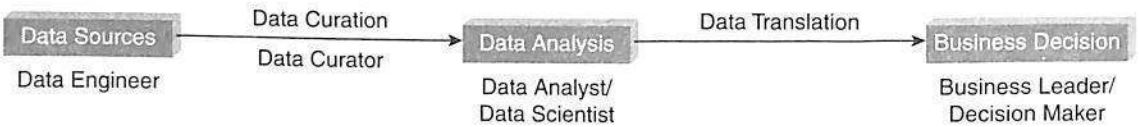
14.1 RISE OF THE NEW AGE "DATA CURATORS"

In the coming times, data curator will become a very significant role. To understand the role and responsibilities of the data curators, let us first understand what data curation is. In simple terms, data curation implies ensuring that people can easily find and use the data now and in the future. It can be detailed down to:

1. Identifying the most relevant data sources.
2. Sourcing (getting) the data from data owners.
3. Fixing any issues with the data such as missing values, unexpected values, and mysterious variables. Also preparing the data if not suitable for analysis; for example, if the data has too much details or is too granular to be picked up for analysis then maybe summarization or aggregation might help in the analysis.
4. Using annotations, tags, appropriate and crisp documentation, etc. to help make the data easy to find, use, reuse and present.
5. Preserving the data such that it is available for use, reuse, etc.

Data curator is a role that sits between data engineers and data consumers (data analysts, data scientists, etc.).

Let us discuss how data was made available to data consumers when there was no role as data curator. Data consumers, basis their analytics task, would raise requests for data. They would provide specifications/details such as the data sources or datasets required for the job, the format in which data should be provided, the tools that they will use to run analysis on the data, how frequently data should be updated, and some details about the analysis task on hand. Data engineers would then run the errand of getting the data from the identified sources, ask for more details if required, blend the data from two or more disparate sources, transform it as per requirements, secure access, etc. The data engineers have a complete understanding of the infrastructure and the formatting of the data; however, they may or may not understand the data completely. The data consumers, on the other hand, have a fairly good understanding of the data but usually do not have much idea of the systems, processes and tools to bring data to the present form.



Enter the role of the data curator to bridge this gap between IT – data engineers and data consumers (data analysts, scientists, etc.). They work closely with both the data engineers as well as data consumers. On one hand, they have up-to-date knowledge about datasets, their provenance (origin), and what data curation is needed; on the other hand, they also understand the different types of analysis to be performed on specific datasets as well as the expectations of latency and availability set by diverse business users.

14.2 CDOS ARE STEPPING UP

With data becoming the new oil, the clear mandate is to create more and more value from the organization's data. Enter the role of CDO (Chief Data Officer) to help with data leveraging (use the existing data assets in the best possible way), data enrichment (augment the value of data by blending, bringing together internal and external data), data monetization (exploring newer avenues of earnings and revenues), data upkeep (ensuring proper data quality and governance), data protection (ensuring security and adequate protection of data), etc.

Let us look at few statistics:

- Estimated number of CDOs globally as per Gartner:
  2010: 15
  2014: 400
  2018: 4000+
- Percentage of large companies with a CDO in place:

	CDO	
Year	Yes	No
2012	12%	88%
2017	56%	44%
2018	63%	375

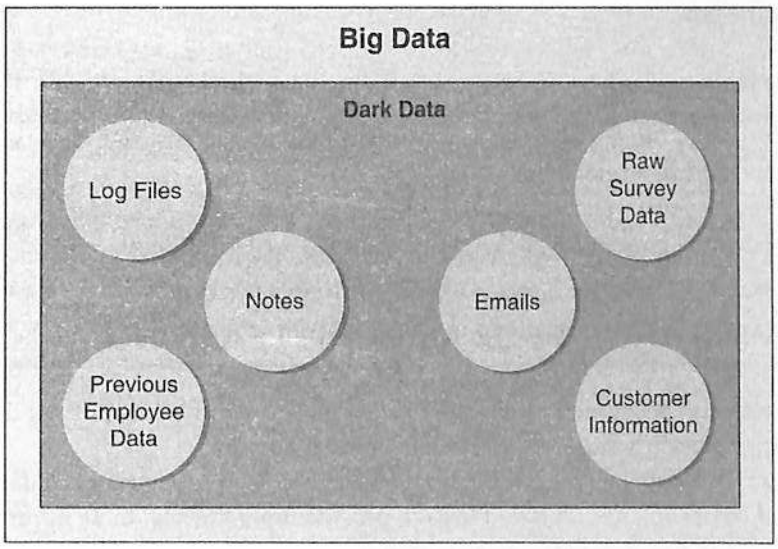
14.3 DARK DATA IN THE CLOUD

What is Dark Data?

According to Gartner, dark data is "the information assets organization's collect, process and store during regular business activities, but generally fail to use for other purposes".

We all know about big data. Data that is big in volume, variety and velocity. Not all the big data that is collected by the organizations are processed, analyzed or used. This data that is collected and stored by the organizations thinking that it will be used for some analysis sometime in future but is never put to any use is "dark data". Dark data is a subset of big data. It also happens to constitute the biggest portion of the large volume of big data that organizations collect and store.

As per IDC (International Data corporation), 90% of unstructured data is "dark data".



Why is it important then to consider dark data? Primarily because this dark data could mean opportunity or opportunities lost for an organization. It may have untapped, undiscovered useful insights which could spell success for the organization. Another reason why it becomes important is if the dark unanalyzed data is not handled well, it can result in a lot of problems such as legal and security problems.

Why then the Dark Data has not been handled the way it should have been?

There are several reasons for this. Few are listed below:

1. **Not knowing what to do with the data:**
   Picture this:
   A bank receives online applications for credit cards. Their focus is mainly on collecting and analyzing customer details and eligibility. They attach little or no priority to finding out how the customer came to the application page. If this data was collected and analyzed, it could have provided useful insights about the usage of the bank website or could have helped improve the application.
2. **Disconnect among departments:** Typically in large organizations, departments operate in silos. They have their own data collection and storage processes. They may or may not reveal these processes to other departments. So, there is a good chance of data lying unused even though the possibility is that some or all of this data may be relevant/useful to some other department.
3. **Technology and tool constraint:** Again, if we take the case of a large organization, there is a high possibility that all applications do not use the same technology and tools for data collection, storage, etc. Sometime the integration of all this data becomes a problem and at times impossible. There could be integration issue, data quality issues, data governance issues, data ownership issues, etc.

What problems can dark data cause?

1. **Opportunity lost:** This is the data that is as yet untapped. It may have useful insights which can help the company surge ahead of competition.
2. **Legal and regulatory issues:** A lot of this data is lying around sometimes secured, sometimes unguarded. Any inadvertent disclosures could lead to intelligence risk, legal liabilities and even loss of reputation for the firm.

What is the way forward when it comes to handling dark data?

1. Properly structuring or categorizing the data will go a long way in ensuring that the process of searching for the data later can be eased out.
2. Securing the data by way of encryption, etc.
3. Having clearly defined policies for dark data retention as well as disposal.

14.4 STREAMING THE IoT FOR MACHINE LEARNING

Let us quickly do a recap on Machine Learning (ML). Machine Learning uses "stored data" for training in a "controlled" learning environment.

With the rise of IoT (Internet of Things), the need of the hour is to use "streaming data in real time" in a "much less controlled environment". This will help to provide more flexible, more appropriate responses to a variety of situations including communicating with humans.

Gone are the days of operating in silos. Today, IoT data, streaming analytics, machine learning, and distributed computing have come together to offer a very powerful, yet an inexpensive proposition to store and analyze big volume and varied types of digital data.

Some examples of IoT, Big Data, and Machine Learning working together include:

1. **Healthcare:** Continuous monitoring of chronic diseases.
2. **Smart Cities:** Traffic patterns and congestion management.
3. **Transportation:** Optimizing routes and fuel consumption.
4. **Automobile:** Smart cars.
5. **Retail:** Location-based advertising.

14.5 EDGE COMPUTING

Edge computing allows data from IoT devices to be analyzed at the edge of the network before being sent to a data center or cloud.

It takes advantage of microservices architectures to allow some portion of computing to be moved to the edge of the network which reduces network traffic and processing time.

Benefits that edge computing provides are as follows:

1. **Cost-effective data processing solution:** Edge computing lowers IoT costs by locally performing vital data computing. Businesses can then decide which services to run locally and which will reside in the Cloud. This greatly optimizes IoT solution costs, by augmenting with cellular-based technologies at a lower-cost, and improves return on investment.
2. **Faster response times:** Data latency (the time the request is submitted to the time the response/outcome is available) is reduced by cutting out round trips to the Cloud. This delivers faster responses. This in turn ensures that critical operations function smoothly without breaking down or incidents occurring.
3. **Security and compliance:** Edge computing reduces or avoids general data transfer between devices and the Cloud. Data can be filtered basis its sensitivity. Sensitive information can be processed locally with non-sensitive data sent for further processing to adhere to strict security and compliance frameworks.
4. **Interoperability between legacy and new devices:** Enterprises typically will have legacy as well as modern smart devices. Edge computing acts as an interpreter between legacy and modern devices. Legacy devices have their own communication protocols. Edge computing helps convert the same for easy consumption and comprehension by new smart devices and the Cloud. This enables legacy devices to be connected to the latest IoT platforms, extending the life of older IT architecture.
5. **Dependable operations with sporadic Internet connectivity:** Edge computing enables equipment that is remotely placed or has intermittent Internet connectivity to work seamlessly. Equipment can function offline without any disruption providing an ideal scenario for fast analysis of data in distant or remote locations such as oil rigs, solar farms, rural areas, etc. It also can detect equipment failure even in instances of restricted connectivity.

14.6 OPEN SOURCE

The future of open source is looking brighter than ever with corporate buy-ins. Two major acquisitions which made headline in 2018 were that of GitHub by Microsoft for an estimated 7.5 billion USD and Red Hat by IBM for a whopping 34 billion USD.

The coming years will see a widespread adoption of open-source components/tools in the software development lifecycle as also the tools that can help developers manage open source to avoid late stage security and compliance issues.

14.7 HADOOP IS FUNDAMENTAL AND WILL REMAIN SO!

The last decade witnessed the failure of quite a few big data projects. While there were reasons galore, two prominent ones were:

1. Spark displacing Hadoop as a stand-alone installation.
2. Data lakes becoming popular in Cloud storage layers.

However, Hadoop is fundamental and is likely to remain so with various Hadoop ecosystem components being used to analyze data. It is yet again touted that Hadoop together with Spark, Business Intelligence tools for integration and visualization will rule the data analytics markets.

14.8 CHATBOTS WILL GET SMARTER

"By 2020, over 50% of medium to large enterprises will have deployed product chatbots," said Van Baker, Research Vice President at Gartner at the Gartner Application Architecture, Development & Integration Summit, held March 12–13, 2018 in Mumbai.

We have already come a long way from the subpar interactions with bots like Siri, Alexa, Google Assistant and have seen them mature over a period of time. Yet there is still room for improvement.

Some of the business use cases for chatbots are:

1. Use your phones or mobile apps to place orders.
2. Handle customer functions: order status, order cancellations, return instructions, tracking numbers, order modifications, account balance, etc.
3. Act as personal digital assistants that help employees do basic tasks: reserving conference rooms, registering mileage, recording expenses, etc.
4. Provide automated support responses to customer inquiries .
5. Navigate through customer services, such as government or administrative services.

Chatbots with their ability to use natural language processing to map spoken or written input to intent, are increasingly entering the workspace. This technology is here to stay.

14.9 CONTAINER(ED) REVOLUTION

Docker-based container technology is becoming popular by the day. The cloud providers are taking advantage of container technology to make the provisioning of nodes faster and to facilitate greater resource sharing – allowing ephemeral clusters to seem persistent.

Hadoop itself, which recently hit its 3.0 release, will soon support the ability for code deployed to YARN to run in the context of a Docker container, thus allowing Hadoop job code dependencies to differ from what may be installed on each node in the cluster.

14.10 COMMODITIZATION OF VISUALIZATION

Visualization is now commoditized. Now visualization ceases to provide competitive advantage. Good analytics and visualization are available for free for most part. The constraint no longer seems to be the monetary investment but rather the constraint and limitation are more in terms of the available time to perform analysis and reporting.

Just as an example, one of the market leaders, Microstrategy, now provides connectors to Tableau, QlikView and PowerBI – the self-service BI tools and the three market leading visualization tools (Tableau, QlikView and PowerBI) between them together with Plot.ly, D3 ecosystem and few open-source geospatial tools provides entry level analytics for free.

Glossary

A

Ad-Hoc Query Any query that cannot be determined prior to the moment the query is issued. A query that consists of dynamically constructed SQL, which is usually constructed by desk-top resident query tools.

Ad-Hoc Query Tool An end-user tool that accepts an English-like or point-and-click request for data and constructs an ad-hoc query to retrieve the desired result.

Affinity Analysis Refer to Association rule mining.

Ambari Component of Hadoop Ecosystem. It is a web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters.

Apache Spark It is a cluster computing framework which is open source and distributed. It is fast, in-memory data processing engine. It can be a standalone installation or can sit atop the Hadoop cluster.

Association Rule Mining It is a well-researched method of finding relations between variables in large databases.

Availability In distributed system, availability implies that a read/write request from a client will always be catered to.

B

Big Data Big data is data that is big in Volume, Variety and Velocity. It is difficult to store and process big data using traditional database and software techniques. Big data has challenges with the following:

- Capture
- Curation
- Storage
- Search

- Transfer
- Analysis
- Visualization
- Information privacy, etc.

Big Data Analytics Big data analytics is the process of examining big data to uncover patterns, unearth trends, and find unknown correlations and other useful information to make faster and better decisions.

BigTable It is a compressed, proprietary data storage system built on Google File System.

BSON Binary JSON.

Business Data Information about people, places, things, business rules, and events, which is used to operate the business. It is not metadata. (Metadata defines and describes business data.)

C

Cassandra Born at FaceBook, Cassandra is an open source, NoSQL database, and a wide column data store. It supports tunable consistency. It has support for MapReduce.

Central Warehouse A database created from operational extracts that adheres to a single, consistent, enterprise data model to ensure consistency of decision-support data across the corporation. A style of computing where all the information systems are located and managed from a single physical location.

Clustering Clustering is the process of grouping similar objects together. The objects in the same group are more similar (called a cluster) to each other than to those in other groups (clusters).

Chatbot It is an Artificial Intelligence (AI) program designed to simulate conversations with humans. It is also called Chatterbot/Smartbot/Talkbot, etc. Example: Siri, Alexa, Google Assistant, etc.

Chukwa Component of Hadoop Ecosystem. It is a data collection system for managing large distributed systems.

Collaborative Filtering It is used by several recommender systems. Example: it is about making predictions about the interests or preferences of a user by collecting preferences/interests or taste information from many users.

Column Database Each storage block has data from only one column. For example: Cassandra, HBase, etc.

Consistency In distributed system, consistency implies that a read request from a client will always fetch the most recent write.

CouchDB CouchDB is a document-oriented database. It has a JavaScript interface. It uses a multi-version concurrency version. It uses JSON to store data.

D

Dark Data According to Gartner, dark data is "the information assests organizations collect, process and store during regular business activities, but generally fails to use for other purposes".

Data Items representing facts, text, graphics, bit-mapped images, sound, analog or digital live-video segments. Data is the raw material of a system supplied by data producers and is used by information consumers to create information.

Data Curation In simple terms, data curation implies ensuring that people can easily find and use the data now and in the future.

Data Curator Data curator is a role that sits between data engineers and data consumers (data analysts, data scientists, etc.). They work closely with both the data engineers as well as data consumers. On one hand, they have up-to-date knowledge about datasets, their provenance (origin), and what data curation is needed and on the other hand, they also understand the different types of analysis to be performed on specific datasets as well as the expectations of latency and availability set by diverse business users.

Database Schema The logical and physical definition of a database structure. Example: the schema of an "Employee" table.

Employee Table:

ColumnName	Data Type and Length	Constraints
EmpID	Varchar(6)	Primary Key
EmpName	Varchar(30)	
EmpDesg	Varchar(20)	Not Null
EmpUnit	Varchar(10)	Not Null

Data Integration This concept describes extraction of business data from various disparate sources to produce a unified view for the end user.

Data Loading The process of populating the data warehouse. Data loading is provided by DBMS-specific load processes, DBMS insert processes, and independent fastload processes.

Data Mart A data mart is usually a subset of the data warehouse and is oriented to the needs of a specific business line or team.

Data Mining A technique using software tools, geared for the user who typically does not know exactly what he's searching for, but is looking for particular patterns or trends. Data mining is the process of sifting through large amounts of data to produce data content relationships. This is also known as data surfing.

Data Model A logical map that represents the inherent properties of the data independent of software, hardware or machine performance considerations. The model shows data elements grouped into records, as well as the association around those records.

Data Modeling A method used to define and analyze data requirements needed to support the business functions of an enterprise. These data requirements are recorded as a conceptual data model with associated data definitions. Data modeling defines the relationships between data elements and structures.

Data Store A place where data is stored; data at rest. A generic term that includes databases and flat files.

Data Warehouse An implementation of an informational database used to store sharable data sourced from an operational database-of-record. It is typically a subject database that allows users to tap into a company's vast store of operational data to track and respond to business trends and facilitate forecasting and planning efforts. Ralph Kimball uses the following terms to describe a data warehouse:

- Subject-oriented
- Integrated
- Non-volatile
- Time-variant

Data Visualization A graphic representation of data.

DBA Database Administrator.

DCL Data Control Language statements. DCL statements are as listed below:

- GRANT
- REVOKE

DDL Data Definition Language statements. A few DDL statements are as follows:

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE INDEX
- DROP INDEX
- TRUNCATE TABLE

Decision trees It is a decision support tool. It uses a tree-like graph or model of decisions and their possible consequences.

Descriptive Analytics Descriptive analytics helps to answer the following questions:

- What happened?
- Why did it happen?, etc.

It is reporting on events, occurrences of the past.

DML Data Manipulation Language statements. The DML statements are the following:

- SELECT
- INSERT
- UPDATE
- DELETE

Document Database It maintains data in collections constituted of documents. For example, MongoDB, Apache CouchDB, Couchbase, MarkLogic, etc.

Sample Document in Document Database

```
{
"Book Name":          "Fundamentals of Business Analytics",
"Publisher":          "Wiley India",
"Year of Publication": "2011"
}
```

DW Data Warehouse. Refer to Data Warehouse.

DWH Data Warehouse. Refer to Data Warehouse.

E

Edge Computing It allows data from Internet of things devices to be analyzed at the edge of the network before being send to a data center or cloud.

EDW Enterprise Data Warehouse. Refer to Data Warehouse.

EIS Executive Information System.

ELT Extract, Load and Transform. It is essentially the same as ETL.

Enterprise A complete business consisting of functions, divisions, or other components used to accomplish specific objectives and defined goals.

Enterprise Data Data that is defined for use across a corporate environment.

ETL Extract Transform Load. ETL is a process of extracting data from a multitude of disparate sources, transforming it and loading it into the enterprise data warehouse or data marts.

G

Graph Database They are also called network database. A graph stores data in nodes. For example, Neo4J, HyperGraphDB, etc.

Greenplum Database from EMC.

H

Hadoop Hadoop is an open-source project of the Apache foundation. It is a framework written in Java, originally developed by Doug Cutting in 2005 who named it after his son's toy elephant.

HBase Hadoop Database.

HDFS Hadoop Distributed File System.

Hive Component of Hadoop Ecosystem. It enables analysis of large data sets using a language very similar to standard ANSI SQL.

Horizontal Scaling A data processing system such as Hadoop is designed to run on clusters of low-cost commodity hardware. It is easy to scale such an arrangement by adding more nodes to the cluster as and when there is need for more storage and processing speed.

I

Impala It is an open-source, distributed, SQL query engine, and analytic database from Cloudera architected to leverage the flexibility and scalability of Hadoop.

Information Data that has been processed in such a way that it can increase the knowledge of the person who receives it. Information is the output or "finished goods" of information systems. Information is also what individuals start with before it is fed into a Data Capture transaction processing system.

Internet of Things (IoT) IoT is sometimes referred to as Internet of Everything. IoT refers to all the web-enabled devices that collect, send and act on data they acquire from their surrounding environments using embedded sensors, processors, and communication hardware. Examples: Smart TVs, wearables, smart appliances, etc. Another example of IoT is the use of smart city technologies to monitor traffic, weather conditions, etc.

J

JSON Java Script Object Notation. MongoDB, a NoSQL uses JSON documents in its collection.

K

Key Performance Indicator (KPI) KPIs are measures to gauge the progress of an Enterprise. It is a means to gauge the progress (or lack of it) in realizing the firm's objectives or strategic plans.

Key–Value Database It maintains a big hash table of keys and values. For example, Dynamo, Redis, Riak, etc.

L

Latency Time It is defined as the time interval between the stimulation and the response. In simple words, it is the time elapsed since the input is fed into a system till the desired outcome is made available.

Linear Regression Linear Regression is used to predict the relationship between two variables, a scalar dependent variable "Y" and one or more explanatory variables denoted by "X".

M

Machine Learning A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Mahout Component of Hadoop Ecosystem. It is a scalable machine learning and data mining library.

MapReduce MapReduce is an algorithm design pattern. It essentially consists of three steps:

- *Step 1 – Mapper function:*
  It takes your input data and outputs a series of keys and values to use in calculating the results.
- *Step 2 – Shuffle and Sort:*
  The output from "Step 1 – Mapper function" which is typically the unordered list of keys and values is then put through a shuffle and sort step to ensure that all the fragments that have the same key are placed next to one another in the file.
- *Step 3 – Reducer function:*
  The reducer function works on the sorted output to reduce/aggregate the input to a single output result per key.

Market Basket Analysis Refer to association rule mining.

Metadata Metadata is data about data. Examples of metadata include data element descriptions, data type descriptions, attribute/property descriptions, range/domain descriptions, and process/method descriptions. The repository environment encompasses all corporate metadata resources: database catalogs, data dictionaries, and navigation services. Metadata includes things like the name, length, valid values, and description of a data element. Metadata is stored in a data dictionary and repository. It insulates the data warehouse from changes in the schema of operational systems.

MPP Massive Parallel Processing. The "shared nothing" approach of parallel computing.

MongoDB MongoDB, the term has come from the word "humongous". It is a NoSQL database. It is a document-oriented database with records stored as JSON (Java Script Object Notation) documents. It supports automatic sharding and MapReduce operations.

Example:

```
{
BookTitle: "Big data and Analytics",
BookAuthor1: "Seema Acharya",
BookAuthor2: "Subhashini Chellapan",
Publisher: "Wiley India"
}
```

N

NewSQL NewSQL is a new modern RDBMS. It supports relational data model and uses SQL as their primary interface. It is a database that has the same scalable performance of NoSQL systems for On Line Transaction Processing (OLTP) while still maintaining the ACID guarantees of a traditional database.

Normalization The process of reducing a complex data structure into its simplest, most stable structure. In general, the process entails the removal of redundant attributes, keys, and relationships from a conceptual data model.

NoSQL (Not only SQL) A NoSQL database provides a storage and retrieval mechanism that allows data to be modeled in forms other than relational or tabular forms of conventional databases. Data can be stored in any of the following forms:

- Key–Value pairs: *Example:* Amazon DB
- Document-oriented *Example:* MongoDB
- Column-oriented *Example:* Cassandra
- Graph-based *Example:* Neo4j

The key features of a NoSQL database are its ability to horizontally scale and it's adherence to CAP (Consistency, Availability, and Partition tolerance) theorem.

NUMA Non-Uniform Memory Access. Here, the memory access time depends on the memory location relative to the processor.

O

OBIEE Oracle Business Intelligence Enterprise Edition (formerly Siebel Analytics).

ODBC Open Database Connectivity. A standard for database access co-opted by Microsoft from the SQL Access Group consortium.

ODI Oracle Data Integrator, an ETL tool.

OLAP On-Line Analytical Processing.

OLTP On-Line Transaction Processing. OLTP describes the requirements for a system that is used in an operational environment.

Oozie Component of Hadoop Ecosystem. It is a workflow scheduler system to manage Apache Hadoop jobs.

Operational Database The database-of-record, consisting of system-specific reference data and event data belonging to a transaction-update system. It may also contain system control data such as indicators, flags, and counters. The operational database is the source of data for the data warehouse. It contains detailed data used to run the day-to-day operations of the business. The data continually changes as updates are made, and reflects the current value of the last transaction.

Operational Data Store (ODS) An ODS is an integrated database of operational data. Its sources include legacy systems and it contains current or near term data. An ODS may contain 30 to 60 days of information, while a data warehouse typically contains years of data.

P

Partition Tolerance In distributed system, partition tolerance implies that the system will continue to function even when network partition occurs.

Pig Component of Hadoop Ecosystem. Pig is an easy to understand data flow language.

Predictive Analytics Predictive Analytics helps to answer the following questions:

- What will happen?
- Why will it happen?

It uses data from the past to make predictions for the future.

Prescriptive Analytics The key questions that prescriptive analytics helps to answer are as follows:

- What will happen?
- When will it happen?
- Why will it happen?
- What should be the action taken to take advantage of what will happen?

Q

Query A (usually) complex SELECT statement for decision support. See Ad-Hoc Query or Ad-Hoc Query Software.

Query Response Time The time it takes for the warehouse engine to process a complex query across a large volume of data and return the results to the requester.

Query Tools Software that allows a user to create and direct specific questions to a data base. These tools provide the means for pulling the desired information from a database. They are typically SQL-based tools and allow a user to define data in end-user language.

R

RDBMS It expands to Relational Database Management System. In RDBMS, data is neatly organized into rows and columns. It conforms to the relational data model.

Examples: Oracle Corporation – Oracle, Microsoft – Microsoft SQL Server, IBM – DB2, Teradata Corporation – Teradata, EMC – Greenplum, etc. Amongst the open sources, the popular RDBMS are MySQL and PostgreSQL, etc.

Replication The word "replication" means duplication of data. Replication in database parlance means the electronic copying of data from a database on one computer or server to a database in another computer or server. This ensures that all users share the same level of information.

Replication Factor Replication Factor connotes the number of data copies of a given data item/data block stored across the network.

Regression Analysis It is a statistical process for estimating the relationships among variables.

S

SaaS Software as a Service. It is a software distribution model. Here, the application or applications are hosted by vendors or service providers and made available to the customers over a network.

Scalability The ability to scale to support larger or smaller volumes of data and more or less users. The ability to increase or decrease size or capability in cost-effective increments with minimal impact on the unit cost of business and the procurement of additional services.

Schema The logical and physical definition of data elements, physical characteristics and inter-relationships.

Scorecard The concept of Balanced Scorecard was given by Kaplan and Norton in 1992. It is a measurement system built on integrated data and helps an organization view business performance.

Securability The ability to provide differing access to individuals according to the classification of data and the user's business function, regardless of the variations.

SELECT An SQL statement (command) that specifies data retrieval operations for rows of data in a relational database.

Semantic Mapping The mapping of the meaning of a piece of data.

Semi-structured Data Semi-structured data is a cross between the structured and unstructured data. It is a type of structured data, but does not have the strict data model structure. Semi-structured data is also referred to as self-describing data. It makes use of tags or other types of markers to identify certain elements within the data; however, the data doesn't have a rigid structure.

Example of semi-structured data: XML (EXtensible Markup Language), JSON (Java Script Object Notation), etc.

Sharding The word "shard" means part of a whole. Sharding in database parlance means a horizontal partitioning of the data in the database into smaller, faster and more easily managed shards.

Shared Disk Architecture It is a distributed computing architecture. Here, all disks are accessible from all cluster nodes.

Shared Memory Architecture It is a multiprocessing design where numerous processors access a globally shared memory.

Shared Nothing Architecture In shared nothing architecture, neither the memory nor the disk is shared. Each node is independent and self-sufficient. There is no single point of contention across the system.

Sqoop Component of Hadoop Ecosystem. It is used to transfer bulk data between Hadoop and structured data stores such as relational databases.

Structured Data Structured data is data that has strict adherence to a data model or pre-defined schema. Before we store the data, it requires defining what fields of data will be stored, what will be the data types (integer, date, character, variable length character string, etc.) of each of the data field, what will be the restrictions/constraints (PRIMARY KEY, NOT NULL, UNIQUE, etc.) on these data fields, etc.

Structured data is usually managed using SQL (Structured Query Language).

Example of structured data: Any Relational Database Management System (RDBMS) such as Oracle, DB2, MS SQL Server, MySQL, PostgreSQL, etc.

Symmetric Multiprocessing System (SMP) In SMP (Symmetric Multiprocessing System), two or more identical processors share a common main memory. The processors have complete access to the IO devices and are controlled by a single operating system instance. SMP are tightly coupled multi-processor systems.

T

Throughput Value It is the rate of production or the rate at which something can be processed. In other words, it is the productivity of a system, procedure, process or machine over a unit period.

U

UMA Uniform Memory Access. Here, all processors access the physical memory uniformly.

Unstructured Data Unstructured data cannot be easily classified and fitted into neat boxes. Unstructured data does not have conformance to a pre-defined data model. It is typically text-heavy.

Example of unstructured data: Photos, videos, chat conversations, short text messages (SMS), email, free-form text, audios, blog entries, wikis, etc.

V

Vertical Scaling Traditional database architectures (RDBMS) are designed to run well on a single machine. In order to handle larger volumes of operations, the approach usually employed is to upgrade the machine with a faster processor or more memory.

X

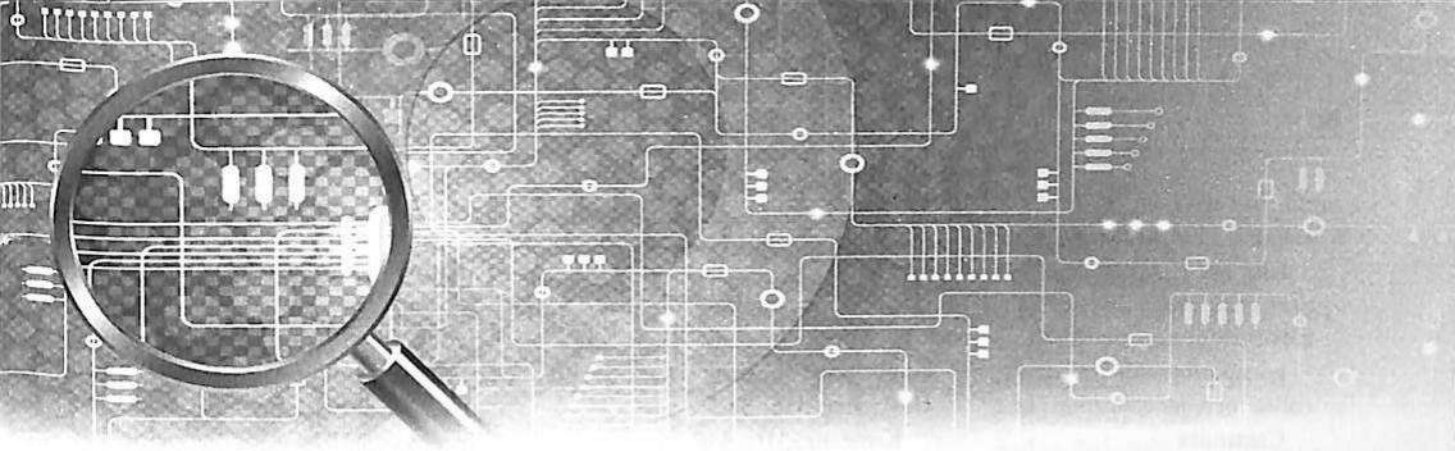
XML EXtensible Markup Language. It was designed to describe data.

Y

YARN Yet Another Resource Negotiator. The fundamental idea of MapReduce version 2.0 (MRv2) is to split up two major functionalities of the JobTracker, resource management and job scheduling/ monitoring, into separate daemons. We now have a global ResourceManager (RM) and a per-application ApplicationMaster (AM).

Z

ZooKeeper Component of Hadoop Ecosystem. It is a coordination service for distributed applications.



Index

About the Book

BIG DATA is a term used for massive mounds of structured, semi-structured and unstructured data that has the potential to be mined for information. The real power lies not just in having colossal data but in what insights can be drawn from this data to facilitate better and faster decisions.

This book *Big Data and Analytics* is a comprehensive coverage on the concepts and practice of Big Data, Hadoop and Analytics. From the *Do It Yourself* steps and guidelines to set up a Hadoop Cluster to the deeper understanding of concepts and ample time-tested hands-on practice exercises on the concepts learned, this ONE book has it all!

Salient Features of the Book

- The book is an exhaustive introduction to Big Data Technology Landscape.
- The concepts are presented in simple language for easy comprehension by beginners.
- The concepts are explained with the help of illustrations and real-life industrial strength application examples.
- The book is accompanied with STEP-BY-STEP Hands-On chapters on
  - NoSQL databases such as MongoDB and Cassandra
  - MapReduce Programming
  - Pig – the data flow language
  - Hive – data warehouse built on top of Hadoop
  - Jasper Soft Studio to design report by pulling the data from MongoDB and Cassandra
- For easy lookup and remembrance, the book carries comparison of various Hadoop components such as HDFS vs. HBase, MapReduce vs. Pig, Pig vs. Hive, Pig vs. Spark, etc.
- For better learning, comprehension and retention, the book is supported by the following:
  - Various types of self-assessment such as "Fill in the Blanks", "Crossword", "Match the Columns", "Hands on Assessment", etc. The solutions to these are also provided.
  - References/web links/bibliography at the end of every chapter.
- **Glossary** of terms frequently used in the big data and analytics at the end of the book.

Resources available on www.wileyindia.com

- Installation guidelines.
- Dataset for JasperReports Assignment.
- Dataset to practice Import from CSV in Cassandra.
- Dataset to practice MongoDB_Import.

Wiley India Pvt. Ltd.
Customer Care +91 120 6291100
csupport@wiley.com
www.wileyindia.com
www.wiley.com

WILEY

ISBN 978-81-265-7951-8
9 788126 579518