Chapter 6: Re-Architecting for NoSQL - Design Principles, Models and Best Practices

# Working with Memchaced – Key/Value pair database

# pseudocode to define new cache and server

memcacheclt = new Memcache  
memcacheclt:add\_server('63.1.4.52:2200')

# Define data cache (value) and key for data within Sales table

sql = "SELECT \* FROM Sales WHERE Product\_id = ?"  
key = 'SQL:' . Product\_id . ':' . md5sum(sql)

# To start with, check if the key / value pair is defined

if (defined result = memcacheclt:get(key)) {  
        return result  
} else {

# execute sql to create key / value pair

      handler = run\_sql(sql, Product\_id)

        # Since you get back a handler or pointer object (after

# executing SQL), you need to convert it to an array for caching

        final\_array = handler:convert\_to\_an\_array

        # Cache it for ten minutes  
        memcacheclt:set(key, final\_array, 10 \* 60)  
        return final\_array  
}

// Working with Neo4j query language Cypher:

// Create Entities or Nodes

CREATE (James:Person { name: 'James', age: 37 })

CREATE (Scott:Person { name: 'Scott', age: 53 })

CREATE (sch:Object { Object\_name: 'Set of candle-holders', price: '$50' })

// Create relations

CREATE (Scott)-[:COLLEAGUE\_OF { since: 1072339200 }]->(James)

CREATE (James)-[:HAS\_BOUGHT { on: 1418976000 }]->(sch)

CREATE (Scott)-[:HAS\_BOUGHT { on: 1420272000 }]->(sch)

// What's the price of set of candle?

MATCH (sch:Object { object\_name: 'Set of candle-holders' })

RETURN sch.price as price

/\* The following Transact-SQL code generates a JSON formatted data file for Policy table (assumed to be) in a MS SQL Server database. The code is generic and you can substitute any SQL Server table name (and primary key column name) and execute it in a database to generate a JSON data file. The only assumption is that first column is primary key; but you can easily modify it for a composite key \*/

-- declare variables

declare @i smallint, @rowkey varchar(500), @ccount smallint, @final\_str varchar(2000)

declare @cname varchar(500), @rec\_str varchar(1000), @dyn\_str nvarchar(500)

declare @res nvarchar(100), @parm nvarchar(200)

-- initialize them

SET @Parm = N'@res nvarchar(500) OUTPUT';

set @i = 2

select @cname = ' '

select @rec\_str = ''

-- take a count of number of columns table has

select @ccount=count(\*) from information\_schema.COLUMNS where TABLE\_NAME = 'Policy'

-- generate string for rowkey

select rowkey='{"' + convert(varchar(10),PolicyId) + '":{' from policy

-- generate string for rest of the columns

While @i <= @ccount

BEGIN

-- select each column in order

select @cname=COLUMN\_NAME from information\_schema.COLUMNS where TABLE\_NAME = 'Policy' and ORDINAL\_POSITION = @i

-- generate string to execute and get column value

select @dyn\_str = 'select @res=' + @cname + ' from Policy'

EXEC sp\_executesql @dyn\_str, @parm, @res=@res OUTPUT;

set @i = @i+1

-- concatenate the column name and value

select @rec\_str = @rec\_str + '"' + @cname + '": "' + convert(varchar(100),@res) + '",'

END

-- Output JSON for the data record after removing the trailing comma and adding curly brackets

select substring(@rec\_str,1,(LEN(@rec\_str)-1)) + '}}'

/\* The following Transact-SQL code generates a BSON formatted data file for Policy table (assumed to be) in a MS SQL Server database. The code is generic and you can substitute any SQL Server table name (and primary key column name) and execute it in a database to generate a BSON data file. The only assumption is that first column (PolicyId) is primary key \*/

-- declare variables

declare @i smallint, @rowkey varchar(500), @ccount smallint, @final\_str varchar(2000)

declare @cname varchar(500), @rec\_str varchar(1000), @dyn\_str nvarchar(500)

declare @res nvarchar(100), @parm nvarchar(200)

-- initialize them

SET @Parm = N'@res nvarchar(500) OUTPUT';

set @i = 2

select @cname = ' '

select @rec\_str = ''

-- take a count of number of columns table has

select @ccount=count(\*) from information\_schema.COLUMNS where TABLE\_NAME = 'Policy'

-- Since MongoDB will generate string for '\_id' column or rowkey

select rowkey='{'

-- generate string for rest of the columns

While @i <= @ccount

BEGIN

-- select each column in order

select @cname=COLUMN\_NAME from information\_schema.COLUMNS where TABLE\_NAME = 'Policy' and ORDINAL\_POSITION = @i

-- generate string to execute and get column value

select @dyn\_str = 'select @res=' + @cname + ' from Policy'

EXEC sp\_executesql @dyn\_str, @parm, @res=@res OUTPUT;

set @i = @i+1

-- concatenate the column name and value

select @rec\_str = @rec\_str + '"' + @cname + '": "' + convert(varchar(100),@res) + '",'

END

-- Output BSON for the data record after removing the trailing comma and adding curly bracket

select substring(@rec\_str,1,(LEN(@rec\_str)-1)) + '}'

Chapter 7: Data Lake Integration Design Principles

R code used in this chapter:

> MyDf <- read.table("Mydata.dat", header=T, sep="~");

> MyDf <- read.table("Mydata.dat", header=T, sep="~", nrows = 100);

> install.packages(“RODBC”)

> library(RODBC)

> DBHandle <- odbcConnect(“MyODBC”)

> MySQLDf <- sqlFetch(DBHandle, “MyTable”)

> odbcClose(DBHandle)

> MySQLDf <- sqlQuery(DBHandle, 'select top 100 \* from MyTable')

> DBHandle <- odbcDriverConnect('driver={SQL Server}; server=MySQLhost; database=MyDB; trusted\_connection=true')

> x <- c(1,3,5,7,9,2,4,5) # input data set

> fivenum(x) # compute five number summary

> boxplot(x) # draw a boxplot of dataset x

> barplot(x, main="Weekly Profits in US Million Dollars", xlab="Week (starting least recent)", ylim=c(0,10), ylab="Profits (US Million dollars)",names.arg = c(1,2,3,4,5,6,7,8))

> hist(x, breaks=8, xlim=c(1,10), main="Histogram for weekly profits", xlab="Weekly profits (Million US dollars", col = "grey")

> qplot(x, geom="histogram", xlab="Profit (Million US dollars)", ylab="Frequency",bins=20)

> qplot(ind, values, data=stack(data.frame(IlliProf,IndiProf)), geom="boxplot") + theme(axis.text=element\_text(size=16,face="bold"),axis.title=element\_text(size=12))

>dat1 <- data.frame(c(1,2,3,4,5,6,7,8),IlliProf)

>dat2 <- data.frame(c(1,2,3,4,5,6,7,8),IndiProf)

> colnames(dat1) <- c("WeekNumber", "Profit")

> colnames(dat2) <- c("WeekNumber", "Profit")

> dat <- rbind(dat1, dat2)

> dat$dataset <- factor(c(rep("dat1", dim(dat1)[1]), rep("dat2", dim(dat2)[1])))

> ggplot(dat, aes(x=WeekNumber, y=Profit, shape=dataset)) + geom\_point(size=5) + theme(axis.text=element\_text(size=14,face="bold"), axis.title=element\_text(size=14), legend.text=element\_text(size=14), legend.title=element\_text(size=14))

> cor(IlliProf, IndiProf, method="spearman")

> with(dfSales, cor(Attr1, Attr2, Attr3, Attr4, Attr5, Sales))

>dfSales60561 <- subset(dfSales, Attr == 60561)

>with(dfSales, cor(Attr1, Attr2, Attr3, Attr4, Attr5, Sales))

>dfSales60561Jan16 <- subset(dfSales60561, date >= as.Date("2016-01-01") & date < as.Date("2016-02-01"))

> ggplot(dat, aes(x=WeekNumber, y=Profit, linetype=dataset))+geom\_line(size=1) + theme(axis.text=element\_text(size=14,face="bold"),axis.title=element\_text(size=14), legend.text=element\_text(size=14),legend.title=element\_text(size=14))

Chapter 8: Implementing SQOOP and Flume based Data Transfers

# tar -zxf mysql-connector-java-5.1.36.tar.gz

# cd mysql-connector-java-5.1.36

# mv mysql-connector-java-5.1.36-bin.jar /usr/lib/sqoop/lib

#After that, you can use the connector for import / export. For example, the below #command imports a table called MyTbl from MySQL database server to HDFS directory #MyData:

$ sqoop import \

--connect jdbc:mysql://localhost/userdb \

--username root \

--table MyTbl \

--m 1 \

--target-dir /MyData

#you can use that to get data for the last day as follows (assuming you are doing the #incremental load on 2/29/16 and want to load the data for 2/28/16):

$ sqoop import

--connect jdbc:mysql://localhost/userdb \

--username Myusr \

--password Mypwd \

--table MyTbl \

--m 1 \

--target-dir /MyData/incremental\_table

--check-column modified\_date

--incremental lastmodified

--last-value 2016-01-27

#For example, if MyTbl has MyId column that holds the self-incrementing id and you #determine that the last incremental load loaded maximum id value of 9834, then the #following command will fetch all the records that were added after that load:

$ sqoop import

--connect jdbc:mysql://localhost/userdb \

--username Myusr \

--password Mypwd \

--table MyTbl \

--m 1 \

--target-dir /MyData/incremental\_table

--check-column modified\_date

--incremental append

--last-value 9834

#Last, I will mention an additional way to load data incrementally. You can leverage #the “query” parameter and use a SQL select statement to limit the import to new or #changed records only as follows:

$ sqoop import

--connect jdbc:mysql://localhost/userdb \

--username Myusr \

--password Mypwd \

--table MyTbl \

--m 1 \

--target-dir /MyData/incremental\_table

--query 'select \* from MyTbl where modified\_date > 2016-01-27’

#Using Flume

s\_agent.sources=s1

s\_agent.sources.s1.type=exec

s\_agent.sources.s1.channels=c1

s\_agent.sources.s1.command= tail -f hdfs-audit.log

s\_agent.sinks.sink1.type=hdfs

s\_agent.sinks.sink1.hdfs.path=/usr/flume/mydata

s\_agent.sinks.sink1.channel=ch1

## SOURCE AGENT ##

## configuration file location: /etc/flume/conf/flume-src.conf

## START Agent: /etc/flume/bin/flume-ng agent -c conf -f conf/flume-src.conf -n s\_agent

## exec-source

s\_agent.sources = MyServer

s\_agent.sources.apache\_server.type = exec

s\_agent.sources.apache\_server.command = tail -f /etc/httpd/logs/access\_log

s\_agent.sources.apache\_server.batchSize = 1

s\_agent.sources.apache\_server.channels = memoryChannel

s\_agent.sources.apache\_server.interceptors = itime

## timestamp-interceptor

s\_agent.sources.apache\_server.interceptors.itime.type = timestamp

## memory-channel

s\_agent.channels = memoryChannel

s\_agent.channels.memoryChannel.type = memory

s\_agent.channels.memoryChannel.capacity = 100

## Send to Flume Collector

## avro-sink

s\_agent.sinks = avro\_sink

s\_agent.sinks.avro\_sink.type = avro

s\_agent.sinks.avro\_sink.channel = memoryChannel

s\_agent.sinks.avro\_sink.hostname = 10.243.169.122

s\_agent.sinks.avro\_sink.port = 4545

## TARGET AGENT ##

## configuration file location: /etc/flume/conf/flume-col.conf

## START Agent: flume-ng agent -c conf -f /etc/flume/conf/flume-col.conf -n target

## avro-source

target.sources = AvroIn

target.sources.AvroIn.type = avro

target.sources.AvroIn.bind = 0.0.0.0

target.sources.AvroIn.port = 4545

target.sources.AvroIn.channels = mc1

## Channels ##

## Source writes to a channel for one sink

target.channels = mc1

## memory-channel

target.channels.mc1.type = memory

target.channels.mc1.capacity = 500

## Sinks ##

target.sinks = LogConsolidator

## Write to HDFS

## hdfs-sink

target.sinks.LogConsolidator.type = hdfs

target.sinks.LogConsolidator.channel = mc1

target.sinks.LogConsolidator.hdfs.path = /user/flume/MyData/%{log\_type}/%y%m%d

target.sinks.LogConsolidator.hdfs.fileType = DataStream

target.sinks.LogConsolidator.hdfs.writeFormat = Text

target.sinks.LogConsolidator.hdfs.rollSize = 0

target.sinks.LogConsolidator.hdfs.rollCount = 10000

target.sinks.LogConsolidator.hdfs.rollInterval = 600

$ /etc/flume/bin/flume-ng agent -c conf -f conf/flume-src.conf -n s\_agent

$ /etc/flume/bin/flume-ng agent -c conf -f /etc/flume/conf/flume-col.conf -n target

Chapter 9: Lambda architecture for real- time Hadoop applications

# the batch layer needs the following processing

CREATE TABLE CorpBusDetails(

Corporation STRING,

BusCategory STRING,

BusDetails STRING

)

PARTITIONED BY (AsOf TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

CREATE TABLE CorpCustSupDetails(

Corporation STRING,

CustSuppStrength INT,

Males INT)

PARTITIONED BY (AsOf TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

CREATE TABLE CorpITDetails(

Corporation STRING,

ITStrength INT,

ColGrads INT)

PARTITIONED BY (AsOf TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

CREATE TABLE CorpFinDetails(

Corporation STRING,

YrlyRevenue BIGINT,

ProfLastYear BIGINT)

PARTITIONED BY (AsOf TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

CREATE EXTERNAL TABLE CorpBusDetails\_stg(

Corporation STRING,

BusCategory STRING,

BusDetails STRING,

AsOf TIMESTAMP

) ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;"

LOCATION "/TrackingInfo/CorpBusDetails/staging";

FROM CorpBusDetails\_stg INSERT OVERWRITE TABLE CorpBusDetails PARTITION (AsOf) SELECT Corporation,BusCategory,BusDetails,AsOf;

Create table Profitemp1 as Select ProfLastYear, AsOf, Corporation from CorpProfitDetails where year(AsOf) <= year(from\_unixtime(unix\_timestamp())) and year(AsOf) >= (year(from\_unixtime(unix\_timestamp())) – 5)

Create table ProfLFiveView as select Corporation, sum(ProfLastYear) as ProfLastFive from Profitemp1 group by Corporation having sum(ProfLastYear) > 0

INSERT INTO TABLE BatchProcHist

VALUES (‘ProfLFiveView', from\_unixtime(unix\_timestamp());

# For serving layer, the following command will add the second index while generating # the tablespace AveCS1000Space:

hadoop jar splout-hadoop-\*-hadoop.jar simple-generate –it HIVE –hdb TrackInfo –htn AveCS1000View -o out-TrackingInfo\_splout\_example -pby Corporation -p 1 -idx "TotalCustSup" -t AveCS1000View -tb AveCS1000Space

# After the tablespace is generated successfully, you need to deploy it as follows:

hadoop jar splout-hadoop-\*-hadoop.jar deploy -q http://localhost:4412 -root out-TrackingInfo\_splout\_example -ts AveCS1000Space

# for the speed layer, the following processing is needed:

Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘ProfLFiveView\_S';

#Now, I just need to determine which of these records is most recent and use that as a #basis to process the records for the first speed layer view:

Create table MaxTbl1 as select max(MaxDate) as MaxDate from MaxTable;

# Finally, get the unprocessed records from the master data set and create the speed layer view. Also, add the timestamp and write a record to the audit history table:

Create table Profitemp1\_S as Select a.ProfLastYear, a.AsOf, a.Corporation from CorpProfitDetails a, MaxTbl1 b where a.AsOf > b.MaxDate;

Create table ProfLFiveView\_S as select Corporation, sum(ProfLastYear) as ProfLastFive from Profitemp1 group by Corporation

INSERT INTO TABLE BatchProcHist

VALUES (‘ProfLFiveView\_S', from\_unixtime(unix\_timestamp());

Create table MaxTable as select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘AveCS1000View’;

Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘AveCS1000View \_S';

Create table MaxTbl1 as select max(MaxDate) as MaxDate from MaxTable;

# Finally, get the unprocessed records from the master data set and create the speed

# layer view. Also, add the timestamp and write a record to the audit history table:

Create table CustSuptemp1\_S as Select a.CustSuppStrength, a.AsOf, a.Corporation from CorpCustSupDetails a, MaxTbl1 b where a.AsOf > b.MaxDate;

Create table CustSuptemp2\_S as select Corporation, sum(CustSuppStrength) as TotalCustSup, count(CustSuppStrength) as CountCustSup from CustSuptemp1\_S group by Corporation;

Create table AveCS1000View\_S as select a.Corporation, (a.TotalCustSup + b.TotalCustSup) as TotalCustSup, (a.CountCustSup + b.CountCustSup) as CountCustSup, ((a.TotalCustSup + b.TotalCustSup) / (a.CountCustSup + b.CountCustSup)) as AveLastFive from CustSuptemp2\_S a, AveCS1000View b where a.Corporation = b.Corporation and (AveLastFive >= 1000)

INSERT INTO TABLE BatchProcHist

VALUES (‘AveCS1000View\_S', from\_unixtime(unix\_timestamp());

# Spark interface and implementation of speed layer using Spark

# Here, I am using Scala and sc is an existing SparkContext:

**val** sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

**val** sqlContext.sql("Create table MaxTable as select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘AveCS1000View’")

**val** sqlContext.sql("Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘AveCS1000View \_S'")

# read the result in a Data frame as follows

**val** resultsDF **=** sqlContext.sql("select a.Corporation, (a.TotalCustSup + b.TotalCustSup) as TotalCustSup, (a.CountCustSup + b.CountCustSup) as CountCustSup, ((a.TotalCustSup + b.TotalCustSup) / (a.CountCustSup + b.CountCustSup)) as AveLastFive from CustSuptemp2\_S a, AveCS1000View b where a.Corporation = b.Corporation and (AveLastFive >= 1000)")

# Register the resultant Data Frame as a temporary table

**val** resultsDF.registerTempTable("AveCS1000View \_S")

**val** results **=** sqlContext.sql("SELECT Corporation FROM AveCS1000View \_S")

Chapter 11: Case study: Implementing Lambda Architecture

#Here are the Hive tables that constitute master data:

CREATE TABLE **ClaimsMaster**(

ClaimId INT,

ClaimSeqId INT,

PolicyId INT,

ClaimSubmissionDate TIMESTAMP,

ClaimType STRING,

ClaimAmount INT,

ClaimSTATUS STRING,

ClaimWeatherCond STRING

)

PARTITIONED BY (LastModified TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

CREATE TABLE **PolicyMaster**(

PolicyId INT,

PolicyOwnerId INT,

PolicyOwnerDOB DATE,

PolicyOwnerSSNFEIN STRING,

PolicyOwnerType STRING,

PolicyType STRING,

PolicyStartDate TIMESTAMP

PolicyEndDate TIMESTAMP)

PARTITIONED BY (LastModified TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

CREATE TABLE **PO\_Drv\_Hist**(

PolicyOwnerId INT,

ViolationNum SMALLINT,

ViolationSeverity TINYINT,

ViolationDate TIMESTAMP,

ViolationDetails STRING)

PARTITIONED BY (LastModified TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

# add a new partition to a table as follows:

CREATE EXTERNAL TABLE **ClaimsMaster\_stg**(

ClaimId INT,

ClaimSeqId INT,

PolicyId INT,

ClaimSubmissionDate TIMESTAMP,

ClaimType STRING,

ClaimAmount INT,

ClaimSTATUS STRING,

ClaimWeatherCond STRING,

LastModified TIMESTAMP

) ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;"

LOCATION "/InsuranceExample/ClaimsMaster/staging";

FROM **ClaimsMaster\_stg** INSERT OVERWRITE TABLE **ClaimsMaster** PARTITION (LastModified) SELECT ClaimId, ClaimSeqId, PolicyId,

ClaimSubmissionDate, ClaimType, ClaimAmount, ClaimSTATUS, ClaimWeatherCond, LastModified;

# table BatchProcHist to maintain history of batch views created:

CREATE TABLE BatchProcHist(

ViewName STRING,

CreatedAt timestamp)

ROW FORMAT DELIMITED FIELDS TERMINATED BY "\;";

# get the Policy owner details; constituting the batch view:

Create table ClaimDefView as Select P.PolicyOwnerID, P.PolicyId, C.Claimcount from PolicyMaster P, ClaimDeftemp1 C where P.PolicyId = C.PolicyId;

# Writing to history table:

INSERT INTO TABLE BatchProcHist

VALUES (‘ClaimDefView', from\_unixtime(unix\_timestamp());

# the second batch view filters Claims by weather conditions such as snow, rain,

# storm, avalanche or tornado:

CREATE table ClaimWeatherView as Select distinct ClaimId,

ClaimWeatherCond from ClaimsMaster where ClaimWeatherCond in (‘Snow’,’Rain’,’Storm’,’Avalanche’,’Tornado’);

# Writing to history table:

INSERT INTO TABLE BatchProcHist

VALUES (‘ClaimWeatherView', from\_unixtime(unix\_timestamp());

# third batch view is to list teenage drivers with at least 2 severity 1 violations in # last 12 months

Create table TeenageVioltemp1 as Select PolicyOwnerId, count(ViolationSeverity) as TotalViolations

from PO\_Drv\_Hist where (datediff(current\_date, add\_months(to\_date(ViolationDate),12)) <= 0) and (ViolationSeverity = 1)

group by PolicyOwnerId

having count(ViolationSeverity) > 2

# As a second step, the entries (from temporary table) will be filtered using DOB #(date of birth) from the denormalized Policy entity

Create table TeenageViolView as Select T.PolicyOwnerId, T.TotalViolations from TeenageVioltemp1 T, PolicyMaster P where T.PolicyOwnerId = P.PolicyOwnerId and

(datediff(current\_date,add\_months(P.PolicyOwnerDOB,228)) <= 0)

# Writing to history table:

INSERT INTO TABLE BatchProcHist

VALUES (‘TeenageViolView', from\_unixtime(unix\_timestamp());

# the fourth batch view lists adult drivers with 3 or more violations

Create table Violtemp1 as Select PolicyOwnerId, count(ViolationSeverity) as TotalViolations

from PO\_Drv\_Hist where (datediff(current\_date, add\_months(to\_date(ViolationDate),12)) <= 0) and (ViolationSeverity = 1)

group by PolicyOwnerId

having count(ViolationSeverity) > 3

# Filtering for adult drivers:

Create table AdultViolView as Select T.PolicyOwnerId, T.TotalViolations from Violtemp1 T, PolicyMaster P where T.PolicyOwnerId = P.PolicyOwnerId and

(datediff(add\_months(P.PolicyOwnerDOB,228),current\_date) < 0)

# Writing to history table:

INSERT INTO TABLE BatchProcHist

VALUES (‘AdultViolView ', from\_unixtime(unix\_timestamp());

# descriptor file for Splout (for serving layer)

{

"name": "PolicyTblspace",

"nPartitions": 12,

"partitionedTables": [{

"name": "ClaimDefView",

"partitionFields": "PolicyOwnerId",

"tableInputs": [{

"inputType": "HIVE",

"hiveTableName": "ClaimDefView",

"hiveDbName": "MyHiveDB"

}]

},

{

"name": "TeenageViolView",

"partitionFields": "PolicyOwnerId",

"tableInputs": [{

"inputType": "HIVE",

"hiveTableName": "TeenageViolView",

"hiveDbName": "MyHiveDB"

}]

},

{

"name": "AdultViolView",

"partitionFields": "PolicyOwnerId",

"tableInputs": [{

"inputType": "HIVE",

"hiveTableName": "AdultViolView",

"hiveDbName": "MyHiveDB"

}]

}]

}

# to deploy this tablespace, following command can be executed

hadoop jar splout-\*-hadoop.jar generate -tf file:///`pwd`/ PolicyTblspace.json -o out-MyHiveDB\_splout\_example

# following command will create an additional index while generating the tablespace #ClaimTblspace:

hadoop jar splout-hadoop-\*-hadoop.jar simple-generate –it HIVE –hdb MyHiveDB –htn ClaimWeatherView -o out-MyHiveDB\_splout\_example -pby ClaimId -p 1 -idx "ClaimWeatherCond" -t ClaimWeatherView -tb ClaimTblspace

# After the tablespaces are generated, deploy them as follows:

hadoop jar splout-hadoop-\*-hadoop.jar deploy -q http://localhost:4412 -root out-MyHiveDB1\_splout\_example -ts PolicyTblspace

hadoop jar splout-hadoop-\*-hadoop.jar deploy -q http://localhost:4412 -root out-MyHiveDB2\_splout\_example -ts ClaimTblspace

# Implement speed layer - write the most recent record for the first batch view to a

# table

Create table MaxTable as select ViewName, max(CreatedAt) as MaxDate from BatchProcHist group by ViewName having ViewName = ‘ClaimDefView';

# write the most recent record for the first speed layer view to the same table

Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘ClaimDefView\_S';

# determine which of these records is most recent

Create table MaxTbl1 as select max(MaxDate) as MaxDate from MaxTable;

# get the unprocessed records from the master data set and create the speed layer

# view. Also, add the timestamp and write a record to the audit history table:

Create table ClaimDeftemp11 as Select PolicyId, ClaimId from ClaimsMaster a, MaxTbl1 b where where a.LastModified > b.MaxDate;

Create table ClaimDeftemp12 as Select PolicyId, count(ClaimId) as Claimcount from ClaimDeftemp11 where datediff(current\_date, add\_months(to\_date(ClaimSubmissionDate),12)) <= 0

group by PolicyId having count(ClaimId) > 5

Create table ClaimDefView\_S as Select P.PolicyOwnerID, P.PolicyId, C.Claimcount from PolicyMaster P, ClaimDeftemp12 C where P.PolicyId = C.PolicyId;

INSERT INTO TABLE BatchProcHist

VALUES (‘ClaimDefView\_S', from\_unixtime(unix\_timestamp());

# Drop the temporary tables now

Drop Table MaxTable;

Drop Table MaxTbl1;

Drop Table ClaimDeftemp11;

Drop Table ClaimDeftemp12;

#Other Speed layer views can be similarly created as the following:

--View 2;

Create table MaxTable as select ViewName, max(CreatedAt) as MaxDate from BatchProcHist group by ViewName having ViewName = ‘ClaimWeatherView';

Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘ClaimWeatherView\_S';

Create table MaxTbl1 as select max(MaxDate) as MaxDate from MaxTable;

CREATE table ClaimWeatherView\_S as Select distinct a.ClaimId, a.ClaimWeatherCond from ClaimsMaster a, MaxTbl1 b where a.LastModified > b.MaxDate and a.ClaimWeatherCond in (‘Snow’,’Rain’,’Storm’,’Avalanche’,’Tornado’);

INSERT INTO TABLE BatchProcHist

VALUES (‘ClaimWeatherView\_S', from\_unixtime(unix\_timestamp());

Drop Table MaxTable;

Drop Table MaxTbl1;

--View 3;

Create table MaxTable as select ViewName, max(CreatedAt) as MaxDate from BatchProcHist group by ViewName having ViewName = ‘TeenageViolView';

Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘TeenageViolView\_S';

Create table MaxTbl1 as select max(MaxDate) as MaxDate from MaxTable;

Create table TeenageVioltemp11 as Select a.PolicyOwnerId, a.ViolationSeverity from PO\_Drv\_Hist a, MaxTbl1 b where a.LastModified > b.MaxDate;

Create table TeenageVioltemp12 as Select PolicyOwnerId, count(ViolationSeverity) as TotalViolations

from TeenageVioltemp11 where (datediff(current\_date, add\_months(to\_date(ViolationDate),12)) <= 0) and (ViolationSeverity = 1)

group by PolicyOwnerId

having count(ViolationSeverity) > 2

Create table TeenageViolView\_S as Select T.PolicyOwnerId, T.TotalViolations from TeenageVioltemp12 T, PolicyMaster P where T.PolicyOwnerId = P.PolicyOwnerId and

(datediff(current\_date,add\_months(P.PolicyOwnerDOB,228)) <= 0)

INSERT INTO TABLE BatchProcHist

VALUES (‘TeenageViolView\_S', from\_unixtime(unix\_timestamp());

Drop Table MaxTable;

Drop Table MaxTbl1;

Drop Table TeenageVioltemp11;

Drop Table TeenageVioltemp12;

--View 4;

Create table MaxTable as select ViewName, max(CreatedAt) as MaxDate from BatchProcHist group by ViewName having ViewName = ‘AdultViolView';

Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘AdultViolView\_S';

Create table MaxTbl1 as select max(MaxDate) as MaxDate from MaxTable;

Create table Violtemp11 as Select a.PolicyOwnerId, a.ViolationSeverity from PO\_Drv\_Hist a, MaxTbl1 b where a.LastModified > b.MaxDate;

Create table Violtemp12 as Select PolicyOwnerId, count(ViolationSeverity) as TotalViolations

from Violtemp11 where (datediff(current\_date, add\_months(to\_date(ViolationDate),12)) <= 0) and (ViolationSeverity = 1)

group by PolicyOwnerId

having count(ViolationSeverity) > 3

Create table AdultViolView\_S as Select T.PolicyOwnerId, T.TotalViolations from Violtemp12 T, PolicyMaster P where T.PolicyOwnerId = P.PolicyOwnerId and

(datediff(add\_months(P.PolicyOwnerDOB,228),current\_date) < 0)

INSERT INTO TABLE BatchProcHist

VALUES (‘AdultViolView\_S', from\_unixtime(unix\_timestamp());

Drop Table MaxTable;

Drop Table MaxTbl1;

Drop Table Violtemp11;

Drop Table Violtemp12;

# Spark implementation steps (for speed layer)

**val** sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

**val** sqlContext.sql("Create table MaxTable as select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘ClaimDefView’")

**val** sqlContext.sql("Insert into MaxTable select ViewName, max(CreatedAt) from BatchProcHist group by ViewName having ViewName = ‘ClaimDefView\_S'")

# For the last step (when the view is created), instead of creating the view, you can

# simply execute the select statement and read the result in a Data frame as follows:

**val** resultsDF **=** sqlContext.sql("Select P.PolicyOwnerID, P.PolicyId, C.Claimcount from PolicyMaster P, ClaimDeftemp12 C where P.PolicyId = C.PolicyId;")

# You can register the resultant Data Frame as a temporary table and then execute any

# queries against it:

**val** resultsDF.registerTempTable("ClaimDefView\_S")

**val** results **=** sqlContext.sql("SELECT PolicyOwnerId FROM ClaimDefView\_S")