Apache CarbonData

Ver 1.0

Documentation

The Apache Software Foundation **2017-08-21**

Table of Contents

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

1 Quick Start

Quick Start

This tutorial provides a quick introduction to using CarbonData.

1.1 Prerequisites

- Installation and building CarbonData.
- Create a sample.csv file using the following commands. The CSV file is required for loading data into CarbonData.

```
cd carbondata cat > sample.csv << EOF id,name,city,age 1,david,shenzhen,31
2,eason,shenzhen,27 3,jarry,wuhan,35 EOF
```

1.2 Interactive Analysis with Spark Shell Version 2.1

Apache Spark Shell provides a simple way to learn the API, as well as a powerful tool to analyze data interactively. Please visit Apache Spark Documentation for more details on Spark shell.

1.2.1.1 Basics

Start Spark shell by running the following command in the Spark directory:

```
./bin/spark-shell --jars <carbondata assembly jar path>
```

NOTE: Assembly jar will be available after building CarbonData and can be copied from `./assembly/target/scala-2.1x/carbondata_xxx.jar`

In this shell, SparkSession is readily available as `spark` and Spark context is readily available as `sc`.

In order to create a CarbonSession we will have to configure it explicitly in the following manner :

- Import the following :

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.CarbonSession._
```

- Create a CarbonSession :

```
val carbon = SparkSession.builder().config(sc.getConf)
             .getOrCreateCarbonSession("<hdfs store path>")
```

NOTE: By default metastore location is pointed to `../carbon.metastore`,
user can provide own metastore location to CarbonSession like `SparkSession.builder().config(sc.getConf) .getOrCreateCarbonSession("<hdfs store path>", "<local metastore path>")`

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

1.2.1.2 Executing Queries

1.Creating a Table

```
scala>carbon.sql("CREATE TABLE
                        IF NOT EXISTS test_table(
                                  id string,
                                  name string,
                                  city string,
                                  age Int)
                       STORED BY 'carbondata'")
```

1.Loading Data to a Table

```
scala>carbon.sql("LOAD DATA INPATH '/path/to/sample.csv'
                  INTO TABLE test_table")
```

NOTE: Please provide the real file path of `sample.csv` for the above script.

1.Query Data from a Table

```
scala>carbon.sql("SELECT * FROM test_table").show()

scala>carbon.sql("SELECT city, avg(age), sum(age)
                  FROM test_table
                  GROUP BY city").show()
```

1.3 Interactive Analysis with Spark Shell Version 1.6

1.3.1.1 Basics

Start Spark shell by running the following command in the Spark directory:

```
./bin/spark-shell --jars <carbondata assembly jar path>
```

NOTE: Assembly jar will be available after building CarbonData and can be copied from `./assembly/target/scala-2.1x/carbondata_xxx.jar`

NOTE: In this shell, SparkContext is readily available as `sc`.

- In order to execute the Queries we need to import CarbonContext:

```
import org.apache.spark.sql.CarbonContext
```

- Create an instance of CarbonContext in the following manner :

```
val cc = new CarbonContext(sc, "<hdfs store path>")
```

NOTE: If running on local machine without hdfs, configure the local machine's store path instead of hdfs store path

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

1.3.1.2 Executing Queries

1.Creating a Table

```
scala>cc.sql("CREATE TABLE
              IF NOT EXISTS test_table (
                        id string,
                        name string,
                        city string,
                        age Int)
              STORED BY 'carbondata'")
```

To see the table created :

```
scala>cc.sql("SHOW TABLES").show()
```

1.Loading Data to a Table

```
scala>cc.sql("LOAD DATA INPATH 'sample.csv file path'
              INTO TABLE test_table")
```

NOTE: Please provide the real file path of `sample.csv` for the above script.

1.Querying Data from a Table

```
scala>cc.sql("SELECT * FROM test_table").show()
scala>cc.sql("SELECT city, avg(age), sum(age)
              FROM test_table
              GROUP BY city").show()
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

2 CarbonData File Structure

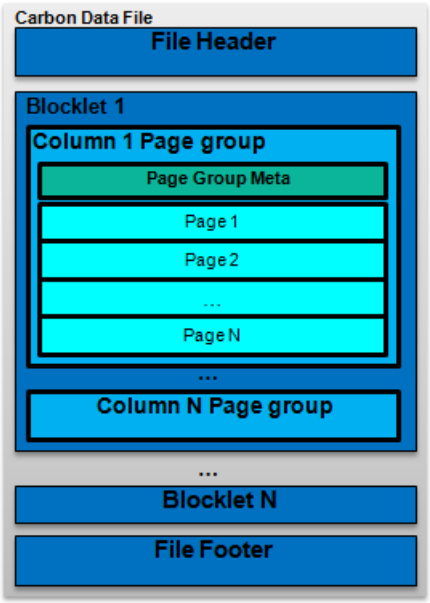
CarbonData File Structure

CarbonData files contain groups of data called blocklets, along with all required information like schema, offsets and indices etc, in a file header and footer, co-located in HDFS.

The file footer can be read once to build the indices in memory, which can be utilized for optimizing the scans and processing for all subsequent queries.

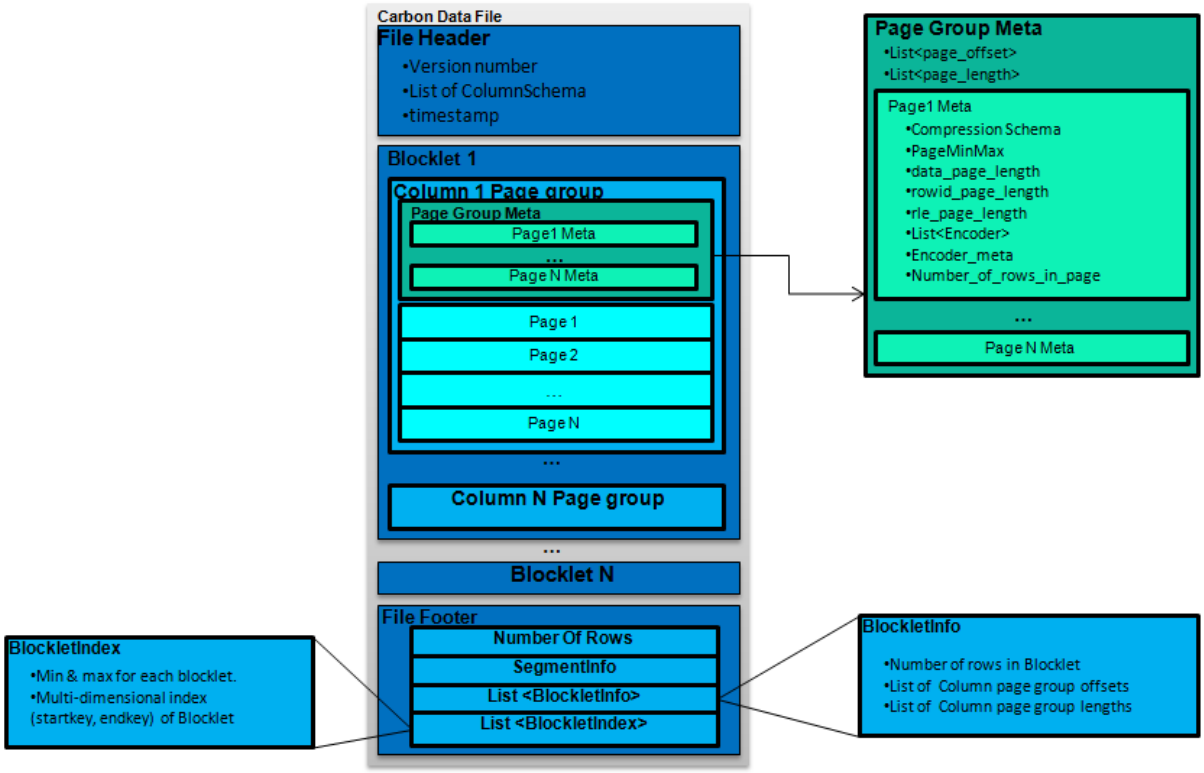
2.1.1 Understanding CarbonData File Structure

- Block : It would be as same as HDFS block, CarbonData creates one file for each data block, user can specify TABLE_BLOCKSIZE during creation table. Each file contains File Header, Blocklets and File Footer.



- File Header : It contains CarbonData file version number, list of column schema and schema updation timestamp.
- File Footer : it contains Number of rows, segmentinfo ,all blocklets' info and index, you can find the detail from the below diagram.
- Blocklet : Rows are grouped to form a blocklet, the size of the blocklet is configurable and default size is 64MB, Blocklet contains Column Page groups for each column.
- Column Page Group : Data of one column and it is further divided into pages, it is guaranteed to be contiguous in file.
- Page : It has the data of one column and the number of row is fixed to 32000 size.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.



2.1.2 Each page contains three types of data

- Data Page: Contains the encoded data of a column of columns.
- Row ID Page (optional): Contains the row ID mappings used when the data page is stored as an inverted index.
- RLE Page (optional): Contains additional metadata used when the data page is RLE coded.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

3 Data Types

Data Types

3.1.1.1 CarbonData supports the following data types:

- Numeric Types
  - SMALLINT
  - INT/INTEGER
  - BIGINT
  - DOUBLE
  - DECIMAL
- Date/Time Types
  - TIMESTAMP
  - DATE
- String Types
  - STRING
  - CHAR
  - VARCHAR
- Complex Types
  - arrays: ARRAY `<data_type>`
  - structs: STRUCT `<col_name : data_type COMMENT col_comment, ...>`

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

4 Data Management

Data Management

This tutorial is going to introduce you to the conceptual details of data management like:

- Loading Data
- Deleting Data
- Compacting Data
- Updating Data

4.1 Loading Data

- **Scenario**

After creating a table, you can load data to the table using the LOAD DATA command. The loaded data is available for querying. When data load is triggered, the data is encoded in CarbonData format and copied into HDFS CarbonData store path (specified in carbon.properties file) in compressed, multi dimensional columnar format for quick analysis queries. The same command can be used to load new data or to update the existing data. Only one data load can be triggered for one table. The high cardinality columns of the dictionary encoding are automatically recognized and these columns will not be used for dictionary encoding.

- **Procedure**

Data loading is a process that involves execution of multiple steps to read, sort and encode the data in CarbonData store format. Each step is executed on different threads. After data loading process is complete, the status (success/partial success) is updated to CarbonData store metadata. The table below lists the possible load status.

Status	Description
Success	All the data is loaded into table and no bad records found.
Partial Success	Data is loaded into table and bad records are found. Bad records are stored at carbon.badrecords.location.

In case of failure, the error will be logged in error log. Details of loads can be seen with SHOW SEGMENTS command. The show segment command output consists of :

- SegmentSequenceId
- Status
- Load Start Time
- Load End Time

The latest load will be displayed first in the output.

Refer to DML operations on CarbonData for load commands.

4.2 Deleting Data

- **Scenario**

If you have loaded wrong data into the table, or too many bad records are present and you want to modify and reload the data, you can delete required data loads. The load can be deleted using the Segment Sequence Id or if the table contains date field then the data can be deleted using the date field. If there are some specific records that need to be deleted based on some filter condition(s) we can delete by records.

- **Procedure**

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

The loaded data can be deleted in the following ways:

- Delete by Segment ID

  After you get the segment ID of the segment that you want to delete, execute the delete command for the selected segment. The status of deleted segment is updated to Marked for delete / Marked for Update.

SegmentSequenceId	Status	Load Start Time	Load End Time
0	Success	2015-11-19 19:14:…	2015-11-19 19:14:…
1	Marked for Update	2015-11-19 19:54:…	2015-11-19 20:08:…
2	Marked for Delete	2015-11-19 20:25:…	2015-11-19 20:49:…

- Delete by Date Field

  If the table contains date field, you can delete the data based on a specific date.

- Delete by Record

  To delete records from CarbonData table based on some filter Condition(s).

  For delete commands refer to DML operations on CarbonData.

- **NOTE**:
  - When the delete segment DML is called, segment will not be deleted physically from the file system. Instead the segment status will be marked as “Marked for Delete”. For the query execution, this deleted segment will be excluded.
  - The deleted segment will be deleted physically during the next load operation and only after the maximum query execution time configured using “max.query.execution.time”. By default it is 60 minutes.
  - If the user wants to force delete the segment physically then he can use CLEAN FILES Command.

Example :

```
CLEAN FILES FOR TABLE table1
```

This DML will physically delete the segment which are “Marked for delete” immediately.

4.3 Compacting Data

- **Scenario**

Frequent data ingestion results in several fragmented CarbonData files in the store directory. Since data is sorted only within each load, the indices perform only within each load. This means that there will be one index for each load and as number of data load increases, the number of indices also increases. As each index works only on one load, the performance of indices is reduced. CarbonData provides provision for compacting the loads. Compaction process combines several segments into one large segment by merge sorting the data from across the segments.

- **Procedure**

There are two types of compaction Minor and Major compaction.

- **Minor Compaction**

  In minor compaction the user can specify how many loads to be merged. Minor compaction triggers for every data load if the parameter carbon.enable.auto.load.merge is set. If any

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

segments are available to be merged, then compaction will run parallel with data load. There are 2 levels in minor compaction.

- Level 1: Merging of the segments which are not yet compacted.
- Level 2: Merging of the compacted segments again to form a bigger segment.

- **Major Compaction**

  In Major compaction, many segments can be merged into one big segment. User will specify the compaction size until which segments can be merged. Major compaction is usually done during the off-peak time.

There are number of parameters related to Compaction that can be set in carbon.properties file

Parameter	Default	Application	Description	Valid Values
carbon.compaction.le\	4, 3	Minor	This property is for minor compaction which decides how many segments to be merged. Example: If it is set as 2, 3 then minor compaction will be triggered for every 2 segments. 3 is the number of level 1 compacted segment which is further compacted to new segment.	NA
carbon.major.compac\	1024 MB	Major	Major compaction size can be configured using this parameter. Sum of the segments which is below this threshold will be merged.	NA
carbon.numberof.pres	0	Minor/Major	This property configures number of segments to preserve from being compacted. Example: carbon.numberof.pres then 2 latest segments will always be excluded from the compaction. No segments will be preserved by default.	0-100

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.allowed.compa	0	Minor/Major	Compaction will merge the segments which are loaded within the specific number of days configured. Example: If the configuration is 2, then the segments which are loaded in the time frame of 2 days only will get merged. Segments which are loaded 2 days apart will not be merged. This is disabled by default.	0-100
carbon.number.of.core	2	Minor/Major	Number of cores which is used to write data during compaction.	0-100

For compaction commands refer to DDL operations on CarbonData

4.4 Updating Data

- **Scenario**

  Sometimes after the data has been ingested into the System, it is required to be updated. Also there may be situations where some specific columns need to be updated on the basis of column expression and optional filter conditions.

- **Procedure**

  To update we need to specify the column expression with an optional filter condition(s).

  For update commands refer to DML operations on CarbonData.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

5 DDL

DDL Operations on CarbonData

This tutorial guides you through the data definition language support provided by CarbonData.

5.1 Overview

The following DDL operations are supported in CarbonData :

- CREATE TABLE
- SHOW TABLE
- ALTER TABLE
- RENAME TABLE
- ADD COLUMN
- DROP COLUMNS
- CHANGE DATA TYPE
- DROP TABLE
- COMPACTION
- BUCKETING

5.2 CREATE TABLE

This command can be used to create a CarbonData table by specifying the list of fields along with the table properties.

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
                  [(col_name data_type , ...)]
STORED BY 'carbondata'
[TBLPROPERTIES (property_name=property_value, ...)]
// All Carbon's additional table options will go into properties
```

5.2.1 Parameter Description

Parameter	Description	Optional
db_name	Name of the database. Database name should consist of alphanumeric characters and underscore(_) special character.	YES
field_list	Comma separated List of fields with data type. The field names should consist of alphanumeric characters and underscore(_) special character.	NO
table_name	The name of the table in Database. Table name should consist of alphanumeric characters and underscore(_) special character.	NO

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

STORED BY	“org.apache.carbondata.format”, identifies and creates a CarbonData table.	NO
TBLPROPERTIES	List of CarbonData table properties.	YES

5.2.2 Usage Guidelines

Following are the guidelines for using table properties.

- **Dictionary Encoding Configuration**

  Dictionary encoding is enabled by default for all String columns, and disabled for non-String columns. You can include and exclude columns for dictionary encoding.

```
TBLPROPERTIES ('DICTIONARY_EXCLUDE'='column1, column2')
TBLPROPERTIES ('DICTIONARY_INCLUDE'='column1, column2')
```

Here, DICTIONARY_EXCLUDE will exclude dictionary creation. This is applicable for high-cardinality columns and is an optional parameter. DICTIONARY_INCLUDE will generate dictionary for the columns specified in the list.

- **Table Block Size Configuration**

  The block size of table files can be defined using the property TABLE_BLOCKSIZE. It accepts only integer values. The default value is 1024 MB and supports a range of 1 MB to 2048 MB. If you do not specify this value in the DDL command, default value is used.

```
TBLPROPERTIES ('TABLE_BLOCKSIZE'='512')
```

Here 512 MB means the block size of this table is 512 MB, you can also set it as 512M or 512.

- **Inverted Index Configuration**

  Inverted index is very useful to improve compression ratio and query speed, especially for those low-cardinality columns which are in reward position. By default inverted index is enabled. The user can disable the inverted index creation for some columns.

```
TBLPROPERTIES ('NO_INVERTED_INDEX'='column1, column3')
```

No inverted index shall be generated for the columns specified in NO_INVERTED_INDEX. This property is applicable on columns with high-cardinality and is an optional parameter.

NOTE:

- By default all columns other than numeric datatype are treated as dimensions and all columns of numeric datatype are treated as measures.
- All dimensions except complex datatype columns are part of multi dimensional key(MDK). This behavior can be overridden by using TBLPROPERTIES. If the user wants to keep any column (except columns of complex datatype) in multi dimensional key then he can keep the columns either in DICTIONARY_EXCLUDE or DICTIONARY_INCLUDE.
- **Sort Columns Configuration**

  “SORT_COLUMN” property is for users to specify which columns belong to the MDK index. If user don’t specify “SORT_COLUMN” property, by default MDK index be built by using all dimension columns except complex datatype column.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
      TBLPROPERTIES ('SORT_COLUMNS'='column1, column3')
```

5.2.3 Example:

```
   CREATE TABLE IF NOT EXISTS productSchema.productSalesTable (
                                productNumber Int,
                                productName String,
                                storeCity String,
                                storeProvince String,
                                productCategory String,
                                productBatch String,
                                saleQuantity Int,
                                revenue Int)
     STORED BY 'carbondata'
     TBLPROPERTIES ('DICTIONARY_EXCLUDE'='storeCity',
                    'DICTIONARY_INCLUDE'='productNumber',
                    'NO_INVERTED_INDEX'='productBatch',
                    'SORT_COLUMNS'='productName,storeCity')
```

- **SORT_COLUMNS**

  This table property specifies the order of the sort column.

```
   TBLPROPERTIES('SORT_COLUMNS'='column1, column3')
```

NOTE:

- If this property is not specified, then by default SORT_COLUMNS consist of all dimension (exclude Complex Column).
- If this property is specified but with empty argument, then the table will be loaded without sort. For example, (‘SORT_COLUMNS’=’’)

5.3 SHOW TABLE

This command can be used to list all the tables in current database or all the tables of a specific database. `SHOW TABLES [IN db_Name];`

5.3.1 Parameter Description

Parameter	Description	Optional
IN db_Name	Name of the database. Required only if tables of this specific database are to be listed.	YES

5.3.2 Example:

```
  SHOW TABLES IN ProductSchema;
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

5.4 ALTER TABLE

The following section shall discuss the commands to modify the physical or logical state of the existing table(s).

5.4.1 RENAME TABLE

This command is used to rename the existing table. `ALTER TABLE [db_name.]table_name RENAME TO new_table_name;`

5.4.1.1 Parameter Description

Parameter	Description	Optional
db_Name	Name of the database. If this parameter is left unspecified, the current database is selected.	YES
table_name	Name of the existing table.	NO
new_table_name	New table name for the existing table.	NO

5.4.1.2 Usage Guidelines

- Queries that require the formation of path using the table name for reading carbon store files, running in parallel with Rename command might fail during the renaming operation.
- Renaming of Secondary index table(s) is not permitted.

5.4.1.3 Examples:

```
ALTER TABLE carbon RENAME TO carbondata;
```

```
ALTER TABLE test_db.carbon RENAME TO test_db.carbondata;
```

5.4.2 ADD COLUMN

This command is used to add a new column to the existing table.

```
ALTER TABLE [db_name.]table_name ADD COLUMNS (col_name data_type,...)
TBLPROPERTIES('DICTIONARY_INCLUDE'='col_name,...',
'DICTIONARY_EXCLUDE'='col_name,...',
'DEFAULT.VALUE.COLUMN_NAME'='default_value');
```

5.4.2.1 Parameter Description

Parameter	Description	Optional
db_Name	Name of the database. If this parameter is left unspecified, the current database is selected.	YES

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

table_name	Name of the existing table.	NO
col_name data_type	Name of comma-separated column with data type. Column names contain letters, digits, and underscores (_).	NO

NOTE: Do not name the column after name, tupleId, PositionId, and PositionReference when creating Carbon tables because they are used internally by UPDATE, DELETE, and secondary index.

5.4.2.2 Usage Guidelines

- Apart from DICTIONARY_INCLUDE, DICTIONARY_EXCLUDE and default_value no other property will be read. If any other property name is specified, error will not be thrown, it will be ignored.
- If default value is not specified, then NULL will be considered as the default value for the column.
- For addition of column, if DICTIONARY_INCLUDE and DICTIONARY_EXCLUDE are not specified, then the decision will be taken based on data type of the column.

5.4.2.3 Examples:

```
ALTER TABLE carbon ADD COLUMNS (a1 INT, b1 STRING);
```

```
ALTER TABLE carbon ADD COLUMNS (a1 INT, b1 STRING)
TBLPROPERTIES('DICTIONARY_EXCLUDE'='b1');
```

```
ALTER TABLE carbon ADD COLUMNS (a1 INT, b1 STRING)
TBLPROPERTIES('DICTIONARY_INCLUDE'='a1');
```

```
ALTER TABLE carbon ADD COLUMNS (a1 INT, b1 STRING)
TBLPROPERTIES('DEFAULT.VALUE.a1'='10');
```

5.4.3 DROP COLUMNS

This command is used to delete a existing column or multiple columns in a table.

```
ALTER TABLE [db_name.]table_name DROP COLUMNS (col_name, ...);
```

5.4.3.1 Parameter Description

Parameter	Description	Optional
db_Name	Name of the database. If this parameter is left unspecified, the current database is selected.	YES
table_name	Name of the existing table.	NO

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

col_name	Name of comma-separated column with data type. Column names contain letters, digits, and underscores (_)	NO

5.4.3.2 Usage Guidelines

- Deleting a column will also clear the dictionary files, provided the column is of type dictionary.
- For delete column operation, there should be at least one key column that exists in the schema after deletion else error message will be displayed and the operation shall fail.

5.4.3.3 Examples:

If the table contains 4 columns namely a1, b1, c1, and d1.

- **To delete a single column:**

```
ALTER TABLE carbon DROP COLUMNS (b1);
```

```
ALTER TABLE test_db.carbon DROP COLUMNS (b1);
```

- **To delete multiple columns:**

```
ALTER TABLE carbon DROP COLUMNS (c1,d1);
```

5.4.4 CHANGE DATA TYPE

This command is used to change the data type from INT to BIGINT or decimal precision from lower to higher.

```
ALTER TABLE [db_name.]table_name
CHANGE col_name col_name changed_column_type;
```

5.4.4.1 Parameter Description

Parameter	Description	Optional
db_Name	Name of the database. If this parameter is left unspecified, the current database is selected.	YES
table_name	Name of the existing table.	NO
col_name	Name of comma-separated column with data type. Column names contain letters, digits, and underscores (_).	NO
changed_column_type	The change in the data type.	NO

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

5.4.4.2 Usage Guidelines

- Change of decimal data type from lower precision to higher precision will only be supported for cases where there is no data loss.

5.4.4.3 Valid Scenarios

- Invalid scenario - Change of decimal precision from (10,2) to (10,5) is invalid as in this case only scale is increased but total number of digits remains the same.
- Valid scenario - Change of decimal precision from (10,2) to (12,3) is valid as the total number of digits are increased by 2 but scale is increased only by 1 which will not lead to any data loss.
- Note :The allowed range is 38,38 (precision, scale) and is a valid upper case scenario which is not resulting in data loss.

5.4.4.4 Examples:

- **Changing data type of column a1 from INT to BIGINT**

```
ALTER TABLE test_db.carbon CHANGE a1 a1 BIGINT;
```

- **Changing decimal precision of column a1 from 10 to 18.**

```
ALTER TABLE test_db.carbon CHANGE a1 a1 DECIMAL(18,2);
```

5.5 DROP TABLE

This command is used to delete an existing table. `DROP TABLE [IF EXISTS] [db_name.]table_name;`

5.5.1 Parameter Description

Parameter	Description	Optional
db_Name	Name of the database. If not specified, current database will be selected.	YES
table_name	Name of the table to be deleted.	NO

5.5.2 Example:

```
DROP TABLE IF EXISTS productSchema.productSalesTable;
```

5.6 COMPACTION

This command merges the specified number of segments into one segment. This enhances the query performance of the table. `ALTER TABLE [db_name.]table_name COMPACT 'MINOR/MAJOR';`

To get details about Compaction refer to Data Management

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

5.6.1 Parameter Description

Parameter	Description	Optional
db_name	Database name, if it is not specified then it uses current database.	YES
table_name	The name of the table in provided database.	NO

5.6.2 Syntax

- **Minor Compaction** `ALTER TABLE table_name COMPACT 'MINOR';`
- **Major Compaction** `ALTER TABLE table_name COMPACT 'MAJOR';`

5.7 BUCKETING

Bucketing feature can be used to distribute/organize the table/partition data into multiple files such that similar records are present in the same file. While creating a table, a user needs to specify the columns to be used for bucketing and the number of buckets. For the selection of bucket the Hash value of columns is used.

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
                  [(col_name data_type, ...)]
STORED BY 'carbondata'
TBLPROPERTIES('BUCKETNUMBER'='noOfBuckets',
'BUCKETCOLUMNS'='columnname')
```

5.7.1 Parameter Description

Parameter	Description	Optional
BUCKETNUMBER	Specifies the number of Buckets to be created.	No
BUCKETCOLUMNS	Specify the columns to be considered for Bucketing	No

5.7.2 Usage Guidelines

- The feature is supported for Spark 1.6.2 onwards, but the performance optimization is evident from Spark 2.1 onwards.
- Bucketing can not be performed for columns of Complex Data Types.
- Columns in the BUCKETCOLUMN parameter must be only dimension. The BUCKETCOLUMN parameter can not be a measure or a combination of measures and dimensions.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

5.7.3 Example:

```
CREATE TABLE IF NOT EXISTS productSchema.productSalesTable (
                                productNumber Int,
                                saleQuantity Int,
                                productName String,
                                storeCity String,
                                storeProvince String,
                                productCategory String,
                                productBatch String,
                                revenue Int)
  STORED BY 'carbondata'
  TBLPROPERTIES ('DICTIONARY_EXCLUDE'='productName',
                 'DICTIONARY_INCLUDE'='productNumber,saleQuantity',
                 'NO_INVERTED_INDEX'='productBatch',
                 'BUCKETNUMBER'='4',
                 'BUCKETCOLUMNS'='productName')
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

6 DML

DML Operations on CarbonData

This tutorial guides you through the data manipulation language support provided by CarbonData.

6.1 Overview

The following DML operations are supported in CarbonData :

- LOAD DATA
- INSERT DATA INTO A CARBONDATA TABLE
- SHOW SEGMENTS
- DELETE SEGMENT BY ID
- DELETE SEGMENT BY DATE
- UPDATE CARBONDATA TABLE
- DELETE RECORDS FROM CARBONDATA TABLE

6.2 LOAD DATA

This command loads the user data in raw format to the CarbonData specific data format store, this allows CarbonData to provide good performance while querying the data. Please visit Data Management for more details on LOAD.

6.2.1 Syntax

```
LOAD DATA [LOCAL] INPATH 'folder_path'
INTO TABLE [db_name.]table_name
OPTIONS(property_name=property_value, ...)
```

OPTIONS are not mandatory for data loading process. Inside OPTIONS user can provide either of any options like DELIMITER, QUOTECHAR, ESCAPECHAR, MULTILINE as per requirement.

NOTE: The path shall be canonical path.

6.2.2 Parameter Description

Parameter	Description	Optional
folder_path	Path of raw csv data folder or file.	NO
db_name	Database name, if it is not specified then it uses the current database.	YES
table_name	The name of the table in provided database.	NO
OPTIONS	Extra options provided to Load	YES

6.2.3 Usage Guidelines

You can use the following options to load data:

- **DELIMITER:** Delimiters can be provided in the load command.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
OPTIONS('DELIMITER=',')
```

- **QUOTECHAR:** Quote Characters can be provided in the load command.

```
OPTIONS('QUOTECHAR='"')
```

- **COMMENTCHAR:** Comment Characters can be provided in the load command if user want to comment lines.

```
OPTIONS('COMMENTCHAR='#')
```

- **FILEHEADER:** Headers can be provided in the LOAD DATA command if headers are missing in the source files.

```
OPTIONS('FILEHEADER='column1,column2')
```

- **MULTILINE:** CSV with new line character in quotes.

```
OPTIONS('MULTILINE='true')
```

- **ESCAPECHAR:** Escape char can be provided if user want strict validation of escape character on CSV.

```
OPTIONS('ESCAPECHAR='\')
```

- **COMPLEX_DELIMITER_LEVEL_1:** Split the complex type data column in a row (eg., a$b $c –> Array = {a,b,c}).

```
OPTIONS('COMPLEX_DELIMITER_LEVEL_1'='$')
```

- **COMPLEX_DELIMITER_LEVEL_2:** Split the complex type nested data column in a row. Applies level_1 delimiter & applies level_2 based on complex data type (eg., a:b$c:d –> Array> = {{a,b},{c,d}}).

```
OPTIONS('COMPLEX_DELIMITER_LEVEL_2'=':')
```

- **ALL_DICTIONARY_PATH:** All dictionary files path.

```
OPTIONS('ALL_DICTIONARY_PATH'='/opt/alldictionary/data.dictionary')
```

- **COLUMNDICT:** Dictionary file path for specified column.

```
OPTIONS('COLUMNDICT'='column1:dictionaryFilePath1,
column2:dictionaryFilePath2')
```

  NOTE: ALL_DICTIONARY_PATH and COLUMNDICT can’t be used together.

- **DATEFORMAT:** Date format for specified column.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
OPTIONS('DATEFORMAT'='column1:dateFormat1, column2:dateFormat2')
```

NOTE: Date formats are specified by date pattern strings. The date pattern letters in CarbonData are same as in JAVA. Refer to SimpleDateFormat.

- **SINGLE_PASS:** Single Pass Loading enables single job to finish data loading with dictionary generation on the fly. It enhances performance in the scenarios where the subsequent data loading after initial load involves fewer incremental updates on the dictionary.

This option specifies whether to use single pass for loading data or not. By default this option is set to FALSE.

```
```
OPTIONS('SINGLE_PASS'='TRUE')
```
```

Note :

- If this option is set to TRUE then data loading will take less time.
- If this option is set to some invalid value other than TRUE or FALSE then it uses the default value.
- If this option is set to TRUE, then high.cardinality.identify.enable property will be disabled during data load.

Example:

```
LOAD DATA local inpath '/opt/rawdata/data.csv' INTO table carbontable
options('DELIMITER'=',', 'QUOTECHAR'='"','COMMENTCHAR'='#',
'FILEHEADER'='empno,empname,designation,doj,workgroupcategory,
 workgroupcategoryname,deptno,deptname,projectcode,
 projectjoindate,projectenddate,attendance,utilization,salary',
'MULTILINE'='true','ESCAPECHAR'='\','COMPLEX_DELIMITER_LEVEL_1'='$',
'COMPLEX_DELIMITER_LEVEL_2'=':',
'ALL_DICTIONARY_PATH'='/opt/alldictionary/data.dictionary',
'SINGLE_PASS'='TRUE'
)
```

- **BAD RECORDS HANDLING:** Methods of handling bad records are as follows:
  - Load all of the data before dealing with the errors.
  - Clean or delete bad records before loading data or stop the loading when bad records are found.

```
OPTIONS('BAD_RECORDS_LOGGER_ENABLE'='true', 'BAD_RECORD_PATH'='hdfs://hacluster/
```

NOTE:

- If the REDIRECT option is used, Carbon will add all bad records in to a separate CSV file. However, this file must not be used for subsequent data loading because the content may not exactly match the source record. You are advised to cleanse the original source record for further data ingestion. This option is used to remind you which records are bad records.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

- In loaded data, if all records are bad records, the BAD_RECORDS_ACTION is invalid and the load operation fails.
- The maximum number of characters per column is 100000. If there are more than 100000 characters in a column, data loading will fail.

6.2.4 Example:

```
LOAD DATA INPATH 'filepath.csv'
INTO TABLE tablename
OPTIONS('BAD_RECORDS_LOGGER_ENABLE'='true',
'BAD_RECORD_PATH'='hdfs://hacluster/tmp/carbon',
'BAD_RECORDS_ACTION'='REDIRECT',
'IS_EMPTY_DATA_BAD_RECORD'='false');
```

Bad Records Management Options:

| Options | Default Value | Description | |—————————|—————|

| BAD_RECORDS_LOGGER_ENABLE | false | Whether to create logs with details about bad records. | | BAD_RECORDS_ACTION | FAIL | Following are the four types of action for bad records: FORCE: Auto-corrects the data by storing the bad records as NULL. REDIRECT: Bad records are written to the raw CSV instead of being loaded. IGNORE: Bad records are neither loaded nor written to the raw CSV. FAIL: Data loading fails if any bad records are found. NOTE: In loaded data, if all records are bad records, the BAD_RECORDS_ACTION is invalid and the load operation fails. | | IS_EMPTY_DATA_BAD_RECORD | false | If false, then empty ("" or ’’ or ,,) data will not be considered as bad record and vice versa. | | BAD_RECORD_PATH | - | Specifies the HDFS path where bad records are stored. By default the value is Null. This path must to be configured by the user if bad record logger is enabled or bad record action redirect. |

6.3 INSERT DATA INTO A CARBONDATA TABLE

This command inserts data into a CarbonData table. It is defined as a combination of two queries Insert and Select query respectively. It inserts records from a source table into a target CarbonData table. The source table can be a Hive table, Parquet table or a CarbonData table itself. It comes with the functionality to aggregate the records of a table by performing Select query on source table and load its corresponding resultant records into a CarbonData table.

NOTE : The client node where the INSERT command is executing, must be part of the cluster.

6.3.1 Syntax

```
INSERT INTO TABLE <CARBONDATA TABLE> SELECT * FROM sourceTableName
[ WHERE { <filter_condition> } ];
```

You can also omit the `table` keyword and write your query as:

```
INSERT INTO <CARBONDATA TABLE> SELECT * FROM sourceTableName
[ WHERE { <filter_condition> } ];
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

6.3.2 Parameter Description

Parameter	Description
CARBON TABLE	The name of the Carbon table in which you want to perform the insert operation.
sourceTableName	The table from which the records are read and inserted into destination CarbonData table.

6.3.3 Usage Guidelines

The following condition must be met for successful insert operation :

- The source table and the CarbonData table must have the same table schema.
- The table must be created.
- Overwrite is not supported for CarbonData table.
- The data type of source and destination table columns should be same, else the data from source table will be treated as bad records and the INSERT command fails.
- INSERT INTO command does not support partial success if bad records are found, it will fail.
- Data cannot be loaded or updated in source table while insert from source table to target table is in progress.

To enable data load or update during insert operation, configure the following property to true.

```
carbon.insert.persist.enable=true
```

By default the above configuration will be false.

NOTE: Enabling this property will reduce the performance.

6.3.4 Examples

```
INSERT INTO table1 SELECT item1 ,sum(item2 + 1000) as result FROM
table2 group by item1;
```

```
INSERT INTO table1 SELECT item1, item2, item3 FROM table2
where item2='xyz';
```

```
INSERT INTO table1 SELECT * FROM table2
where exists (select * from table3
where table2.item1 = table3.item1);
```

The Status Success/Failure shall be captured in the driver log.

6.4 SHOW SEGMENTS

This command is used to get the segments of CarbonData table.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
SHOW SEGMENTS FOR TABLE [db_name.]table_name
LIMIT number_of_segments;
```

6.4.1 Parameter Description

Parameter	Description	Optional
db_name	Database name, if it is not specified then it uses the current database.	YES
table_name	The name of the table in provided database.	NO
number_of_segments	Limit the output to this number.	YES

6.4.2 Example:

```
SHOW SEGMENTS FOR TABLE CarbonDatabase.CarbonTable LIMIT 4;
```

6.5 DELETE SEGMENT BY ID

This command is used to delete segment by using the segment ID. Each segment has a unique segment ID associated with it. Using this segment ID, you can remove the segment.

The following command will get the segmentID.

```
SHOW SEGMENTS FOR Table [db_name.]table_name LIMIT number_of_segments
```

After you retrieve the segment ID of the segment that you want to delete, execute the following command to delete the selected segment.

```
DELETE FROM TABLE [db_name.]table_name WHERE SEGMENT.ID IN (segment_id1, segments_i
```

6.5.1 Parameter Description

Parameter	Description	Optional
segment_id	Segment Id of the load.	NO
db_name	Database name, if it is not specified then it uses the current database.	YES
table_name	The name of the table in provided database.	NO

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

6.5.2 Example:

```
DELETE FROM TABLE CarbonDatabase.CarbonTable WHERE SEGMENT.ID IN (0);
DELETE FROM TABLE CarbonDatabase.CarbonTable WHERE SEGMENT.ID IN (0,5,8);
```

NOTE: Here 0.1 is compacted segment sequence id.

6.6 DELETE SEGMENT BY DATE

This command will allow to delete the CarbonData segment(s) from the store based on the date provided by the user in the DML command. The segment created before the particular date will be removed from the specific stores.

```
DELETE FROM TABLE [db_name.]table_name
WHERE SEGMENT.STARTTIME BEFORE DATE_VALUE
```

6.6.1 Parameter Description

Parameter	Description	Optional
DATE_VALUE	Valid segment load start time value. All the segments before this specified date will be deleted.	NO
db_name	Database name, if it is not specified then it uses the current database.	YES
table_name	The name of the table in provided database.	NO

6.6.2 Example:

```
DELETE FROM TABLE CarbonDatabase.CarbonTable
WHERE SEGMENT.STARTTIME BEFORE '2017-06-01 12:05:06';
```

6.7 Update CarbonData Table

This command will allow to update the carbon table based on the column expression and optional filter conditions.

6.7.1 Syntax

```
UPDATE <table_name>
SET (column_name1, column_name2, ... column_name n) =
(column1_expression , column2_expression, ... column n_expression )
[ WHERE { <filter_condition> } ];
```

alternatively the following the command can also be used for updating the CarbonData Table :

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
UPDATE <table_name>
SET (column_name1, column_name2) =
(select sourceColumn1, sourceColumn2 from sourceTable
[ WHERE { <filter_condition> } ] )
[ WHERE { <filter_condition> } ];
```

6.7.2 Parameter Description

Parameter	Description
table_name	The name of the Carbon table in which you want to perform the update operation.
column_name	The destination columns to be updated.
sourceColumn	The source table column values to be updated in destination table.
sourceTable	The table from which the records are updated into destination Carbon table.

NOTE: This functionality is currently not supported in Spark 2.x and will support soon.

6.7.3 Usage Guidelines

The following conditions must be met for successful updation :

- The update command fails if multiple input rows in source table are matched with single row in destination table.
- If the source table generates empty records, the update operation will complete successfully without updating the table.
- If a source table row does not correspond to any of the existing rows in a destination table, the update operation will complete successfully without updating the table.
- In sub-query, if the source table and the target table are same, then the update operation fails.
- If the sub-query used in UPDATE statement contains aggregate method or group by query, then the UPDATE operation fails.

6.7.4 Examples

Update is not supported for queries that contain aggregate or group by.

```
UPDATE t_carbn01 a
SET (a.item_type_code, a.profit) = ( SELECT b.item_type_cd,
sum(b.profit) from t_carbn01b b
WHERE item_type_cd =2 group by item_type_code);
```

Here the Update Operation fails as the query contains aggregate function sum(b.profit) and group by clause in the sub-query.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
UPDATE carbonTable1 d
SET(d.column3,d.column5 ) = (SELECT s.c33 ,s.c55
FROM sourceTable1 s WHERE d.column1 = s.c11)
WHERE d.column1 = 'china' EXISTS( SELECT * from table3 o where o.c2 > 1);
```

```
UPDATE carbonTable1 d SET (c3) = (SELECT s.c33 from sourceTable1 s
WHERE d.column1 = s.c11)
WHERE exists( select * from iud.other o where o.c2 > 1);
```

```
UPDATE carbonTable1 SET (c2, c5 ) = (c2 + 1, concat(c5 , "y" ));
```

```
UPDATE carbonTable1 d SET (c2, c5 ) = (c2 + 1, "xyx")
WHERE d.column1 = 'india';
```

```
UPDATE carbonTable1 d SET (c2, c5 ) = (c2 + 1, "xyx")
WHERE d.column1 = 'india'
and EXISTS( SELECT * FROM table3 o WHERE o.column2 > 1);
```

The Status Success/Failure shall be captured in the driver log and the client.

6.8 Delete Records from CarbonData Table

This command allows us to delete records from CarbonData table.

6.8.1 Syntax

```
DELETE FROM table_name [WHERE expression];
```

6.8.2 Parameter Description

Parameter	Description
table_name	The name of the Carbon table in which you want to perform the delete.

NOTE: This functionality is currently not supported in Spark 2.x and will support soon.

6.8.3 Examples

```
DELETE FROM columncarbonTable1 d WHERE d.column1  = 'china';
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
DELETE FROM dest WHERE column1 IN ('china', 'USA');
```

```
DELETE FROM columncarbonTable1
WHERE column1 IN (SELECT column11 FROM sourceTable2);
```

```
DELETE FROM columncarbonTable1
WHERE column1 IN (SELECT column11 FROM sourceTable2 WHERE
column1 = 'USA');
```

```
DELETE FROM columncarbonTable1 WHERE column2 >= 4;
```

The Status Success/Failure shall be captured in the driver log and the client.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

7 Installation

Installation Guide

This tutorial guides you through the installation and configuration of CarbonData in the following two modes :

- Installing and Configuring CarbonData on Standalone Spark Cluster
- Installing and Configuring CarbonData on “Spark on YARN” Cluster

followed by :

- Query Execution using CarbonData Thrift Server

7.1 Installing and Configuring CarbonData on Standalone Spark Cluster

7.1.1 Prerequisites

- Hadoop HDFS and Yarn should be installed and running.
- Spark should be installed and running on all the cluster nodes.
- CarbonData user should have permission to access HDFS.

7.1.2 Procedure

1. Build the CarbonData project and get the assembly jar from `./assembly/target/scala-2.1x/carbondata_xxx.jar`.
2. Copy `./assembly/target/scala-2.1x/carbondata_xxx.jar` to `$SPARK_HOME/carbonlib` folder.

   NOTE: Create the carbonlib folder if it does not exist inside `$SPARK_HOME` path.
3. Add the carbonlib folder path in the Spark classpath. (Edit `$SPARK_HOME/conf/spark-env.sh` file and modify the value of `SPARK_CLASSPATH` by appending `$SPARK_HOME/carbonlib/*` to the existing value)
4. Copy the `./conf/carbon.properties.template` file from CarbonData repository to `$SPARK_HOME/conf/` folder and rename the file to `carbon.properties`.
5. Repeat Step 2 to Step 5 in all the nodes of the cluster.
6. In Spark node[master], configure the properties mentioned in the following table in `$SPARK_HOME/conf/spark-defaults.conf` file.

Property	Value	Description
spark.driver.extraJavaOptions	`-Dcarbon.properties.filepa = $SPARK_HOME/conf/carbon.properties`	A string of extra JVM options to pass to the driver. For instance, GC settings or other logging.
spark.executor.extraJavaOptions	`-Dcarbon.properties.filepa = $SPARK_HOME/conf/carbon.properties`	A string of extra JVM options to pass to executors. For instance, GC settings or other logging. **NOTE**: You can enter multiple values separated by space.

1. Add the following properties in `$SPARK_HOME/conf/carbon.properties` file:

Property	Required	Description	Example	Remark

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.storelocation	NO	Location where data CarbonData will create the store and write the data in its own format.	hdfs://HOSTNAME:PORT/Opt/CarbonStore	Propose to set HDFS directory

1. Verify the installation. For example:

```
./spark-shell --master spark://HOSTNAME:PORT --total-executor-cores 2
--executor-memory 2G
```

NOTE: Make sure you have permissions for CarbonData JARs and files through which driver and executor will start.

To get started with CarbonData : Quick Start, DDL Operations on CarbonData

7.2 Installing and Configuring CarbonData on “Spark on YARN” Cluster

This section provides the procedure to install CarbonData on “Spark on YARN” cluster.

7.2.1 Prerequisites

- Hadoop HDFS and Yarn should be installed and running.
- Spark should be installed and running in all the clients.
- CarbonData user should have permission to access HDFS.

7.2.2 Procedure

The following steps are only for Driver Nodes. (Driver nodes are the one which starts the spark context.)

1. Build the CarbonData project and get the assembly jar from `./assembly/target/scala-2.1x/carbondata_xxx.jar` and copy to `$SPARK_HOME/carbonlib` folder.

   NOTE: Create the carbonlib folder if it does not exists inside `$SPARK_HOME` path.
2. Copy the `./conf/carbon.properties.template` file from CarbonData repository to `$SPARK_HOME/conf/` folder and rename the file to `carbon.properties`.
3. Create `tar.gz` file of carbonlib folder and move it inside the carbonlib folder.

```
cd $SPARK_HOME
tar -zcvf carbondata.tar.gz carbonlib/
mv carbondata.tar.gz carbonlib/
```

1. Configure the properties mentioned in the following table in `$SPARK_HOME/conf/spark-defaults.conf` file.

Property	Description	Value
spark.master	Set this value to run the Spark in yarn cluster mode.	Set yarn-client to run the Spark in yarn cluster mode.
spark.yarn.dist.files	Comma-separated list of files to be placed in the working directory of each executor.	`$SPARK_HOME/conf/carbon.properties`

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

spark.yarn.dist.archives	Comma-separated list of archives to be extracted into the working directory of each executor.	`$SPARK_HOME/carbonlib/carbondata.tar.gz`
spark.executor.extraJavaOptions	A string of extra JVM options to pass to executors. For instance **NOTE**: You can enter multiple values separated by space.	`-Dcarbon.properties.filepath = carbon.properties`
spark.executor.extraClassPath	Extra classpath entries to prepend to the classpath of executors. **NOTE**: If SPARK_CLASSPATH is defined in spark-env.sh, then comment it and append the values in below parameter spark.driver.extraClassPath	`carbondata.tar.gz/carbonlib/*`
spark.driver.extraClassPath	Extra classpath entries to prepend to the classpath of the driver. **NOTE**: If SPARK_CLASSPATH is defined in spark-env.sh, then comment it and append the value in below parameter spark.driver.extraClassPath.	`$SPARK_HOME/carbonlib/*`
spark.driver.extraJavaOptions	A string of extra JVM options to pass to the driver. For instance, GC settings or other logging.	`-Dcarbon.properties.filepath = $SPARK_HOME/conf/carbon.properties`

1. Add the following properties in `$SPARK_HOME/conf/carbon.properties`:

Property	Required	Description	Example	Default Value
carbon.storelocation	NO	Location where CarbonData will create the store and write the data in its own format.	hdfs://HOSTNAME:PORT/Opt/CarbonStore	Propose to set HDFS directory

1. Verify the installation.

```
./bin/spark-shell --master yarn-client --driver-memory 1g
--executor-cores 2 --executor-memory 2G
```

NOTE: Make sure you have permissions for CarbonData JARs and files through which driver and executor will start.

Getting started with CarbonData : Quick Start, DDL Operations on CarbonData

7.3 Query Execution Using CarbonData Thrift Server

7.3.1 Starting CarbonData Thrift Server.

a. cd `$SPARK_HOME`

b. Run the following command to start the CarbonData thrift server.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
./bin/spark-submit
--conf spark.sql.hive.thriftServer.singleSession=true
--class org.apache.carbondata.spark.thriftserver.CarbonThriftServer
$SPARK_HOME/carbonlib/$CARBON_ASSEMBLY_JAR <carbon_store_path>
```

Parameter	Description	Example
CARBON_ASSEMBLY_JAR	CarbonData assembly jar name present in the $SPARK_HOME/carbonlib/ folder.	carbondata_2.xx-x.x.x-SNAPSHOT-shade-hadoop2.7.2.jar
carbon_store_path	This is a parameter to the CarbonThriftServer class. This a HDFS path where CarbonData files will be kept. Strongly Recommended to put same as carbon.storelocation parameter of carbon.properties.	hdfs://<host_name>:port/user/hive/warehouse/carbon.store

Examples

- Start with default memory and executors.

```
./bin/spark-submit
--conf spark.sql.hive.thriftServer.singleSession=true
--class org.apache.carbondata.spark.thriftserver.CarbonThriftServer
$SPARK_HOME/carbonlib
/carbondata_2.xx-x.x.x-SNAPSHOT-shade-hadoop2.7.2.jar
hdfs://<host_name>:port/user/hive/warehouse/carbon.store
```

- Start with Fixed executors and resources.

```
./bin/spark-submit --conf spark.sql.hive.thriftServer.singleSession=true
--class org.apache.carbondata.spark.thriftserver.CarbonThriftServer
--num-executors 3 --driver-memory 20g --executor-memory 250g
--executor-cores 32
/srv/OSCON/BigData/HACluster/install/spark/sparkJdbc/lib
/carbondata_2.xx-x.x.x-SNAPSHOT-shade-hadoop2.7.2.jar
hdfs://<host_name>:port/user/hive/warehouse/carbon.store
```

7.3.2 Connecting to CarbonData Thrift Server Using Beeline.

```
    cd $SPARK_HOME
    ./bin/beeline jdbc:hive2://<thriftserver_host>:port

    Example
    ./bin/beeline jdbc:hive2://10.10.10.10:10000
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

8 Configuring CarbonData

Configuring CarbonData

This tutorial guides you through the advanced configurations of CarbonData :

- System Configuration
- Performance Configuration
- Miscellaneous Configuration
- Spark Configuration

8.1 System Configuration

This section provides the details of all the configurations required for the CarbonData System.

System Configuration in carbon.properties

Property	Default Value	Description
carbon.storelocation	/user/hive/warehouse/carbon.store	Location where CarbonData will create the store, and write the data in its own format. NOTE: Store location should be in HDFS.
carbon.ddl.base.hdfs.url	hdfs://hacluster/opt/data	This property is used to configure the HDFS relative path, the path configured in carbon.ddl.base.hdfs.url will be appended to the HDFS path configured in fs.defaultFS. If this path is configured, then user need not pass the complete path while dataload. For example: If absolute path of the csv file is hdfs://10.18.101.155:54310/data/cnbc/2016/xyz.csv, the path " hdfs://10.18.101.155:54310" will come from property fs.defaultFS and user can configure the /data/cnbc/ as carbon.ddl.base.hdfs.url. Now while dataload user can specify the csv path as /2016/xyz.csv.
carbon.badRecords.location	/opt/Carbon/Spark/badrecords	Path where the bad records are stored.
carbon.data.file.version	2	If this parameter value is set to 1, then CarbonData will support the data load which is in old format(0.x version). If the value is set to 2(1.x onwards version), then CarbonData will support the data load of new format only.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

8.2 Performance Configuration

This section provides the details of all the configurations required for CarbonData Performance Optimization.

Performance Configuration in carbon.properties

- **Data Loading Configuration**

Parameter	Default Value	Description	Range
carbon.sort.file.buffer.size	20	File read buffer size used during sorting. This value is expressed in MB.	Min=1 and Max=100
carbon.graph.rowset.size	100000	Rowset size exchanged between data load graph steps.	Min=500 and Max=1000000
carbon.number.of.cores.whi	6	Number of cores to be used while loading data.	
carbon.sort.size	500000	Record count to sort and write intermediate files to temp.	
carbon.enableXXHash	true	Algorithm for hashmap for hashkey calculation.	
carbon.number.of.cores.blo	7	Number of cores to use for block sort while loading data.	
carbon.max.driver.lru.cache	-1	Max LRU cache size upto which data will be loaded at the driver side. This value is expressed in MB. Default value of -1 means there is no memory limit for caching. Only integer values greater than 0 are accepted.	
carbon.max.executor.lru.ca	-1	Max LRU cache size upto which data will be loaded at the executor side. This value is expressed in MB. Default value of -1 means there is no memory limit for caching. Only integer values greater than 0 are accepted. If this parameter is not configured, then the carbon.max.driver.lru.cache value will be considered.	
carbon.merge.sort.prefetch	true	Enable prefetch of data during merge sort while reading data from sort temp files in data loading.	

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.update.persist.enabl	true	Enabling this parameter considers persistent data. Enabling this will reduce the execution time of UPDATE operation.
carbon.load.global.sort.parti	0	The Number of partitions to use when shuffling data for sort. If user don't configurate or configurate it less than 1, it uses the number of map tasks as reduce tasks. In general, we recommend 2-3 tasks per CPU core in your cluster.
carbon.options.bad.records.	false	Whether to create logs with details about bad records.
carbon.bad.records.action	fail	This property can have four types of actions for bad records FORCE, REDIRECT, IGNORE and FAIL. If set to FORCE then it auto-corrects the data by storing the bad records as NULL. If set to REDIRECT then bad records are written to the raw CSV instead of being loaded. If set to IGNORE then bad records are neither loaded nor written to the raw CSV. If set to FAIL then data loading fails if any bad records are found.
carbon.options.is.empty.dat	false	If false, then empty ("" or '' or ,,) data will not be considered as bad record and vice versa.
carbon.options.bad.record.p		Specifies the HDFS path where bad records are stored. By default the value is Null. This path must to be configured by the user if bad record logger is enabled or bad record action redirect.
carbon.enable.vector.reade	true	This parameter increases the performance of select queries as it fetch columnar batch of size 4*1024 rows instead of fetching data row by row.

- **Compaction Configuration**

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

Parameter	Default Value	Description	Range
carbon.number.of.cores.whi	2	Number of cores which are used to write data during compaction.	
carbon.compaction.level.thr	4, 3	This property is for minor compaction which decides how many segments to be merged. Example: If it is set as 2, 3 then minor compaction will be triggered for every 2 segments. 3 is the number of level 1 compacted segment which is further compacted to new segment.	Valid values are from 0-100.
carbon.major.compaction.si	1024	Major compaction size can be configured using this parameter. Sum of the segments which is below this threshold will be merged. This value is expressed in MB.	
carbon.horizontal.compactic	true	This property is used to turn ON/OFF horizontal compaction. After every DELETE and UPDATE statement, horizontal compaction may occur in case the delta (DELETE/ UPDATE) files becomes more than specified threshold.	
carbon.horizontal.UPDATE.	1	This property specifies the threshold limit on number of UPDATE delta files within a segment. In case the number of delta files goes beyond the threshold, the UPDATE delta files within the segment becomes eligible for horizontal compaction and compacted into single UPDATE delta file.	Values between 1 to 10000.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.horizontal.DELETE.(	1	This property specifies the threshold limit on number of DELETE delta files within a block of a segment. In case the number of delta files goes beyond the threshold, the DELETE delta files for the particular block of the segment becomes eligible for horizontal compaction and compacted into single DELETE delta file.	Values between 1 to 10000.
carbon.update.segment.par	1	This property specifies the parallelism for each segment during update. If there are segments that contain too many records to update and the spark job encounter data-spill related errors, it is better to increase this property value. It is recommended to set this value to a multiple of the number of executors for balance.	Values between 1 to 1000.

- **Query Configuration**

Parameter	Default Value	Description	Range
carbon.number.of.cores	4	Number of cores to be used while querying.	
carbon.inmemory.record.siz	120000	Number of records to be in memory while querying.	Min=100000 and Max=240000
carbon.enable.quick.filter	false	Improves the performance of filter query.	
no.of.cores.to.load.blocks.in	10	Number of core to load the blocks in driver.	

8.3 Miscellaneous Configuration

Extra Configuration in carbon.properties

- **Time format for CarbonData**

Parameter	Default Format	Description
carbon.timestamp.format	yyyy-MM-dd HH:mm:ss	Timestamp format of input data used for timestamp data type.

- **Dataload Configuration**

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

Parameter	Default Value	Description
carbon.sort.file.write.buffer.size	10485760	File write buffer size used during sorting.
carbon.lock.type	LOCALLOCK	This configuration specifies the type of lock to be acquired during concurrent operations on table. There are following types of lock implementation: - LOCALLOCK: Lock is created on local file system as file. This lock is useful when only one spark driver (thrift server) runs on a machine and no other CarbonData spark application is launched concurrently. - HDFSLOCK: Lock is created on HDFS file system as file. This lock is useful when multiple CarbonData spark applications are launched and no ZooKeeper is running on cluster and HDFS supports file based locking.
carbon.sort.intermediate.files.limit	20	Minimum number of intermediate files after which merged sort can be started.
carbon.block.meta.size.reserved.perc	10	Space reserved in percentage for writing block meta data in CarbonData file.
carbon.csv.read.buffersize.byte	1048576	csv reading buffer size.
high.cardinality.value	100000	To identify and apply compression for non-high cardinality columns.
carbon.merge.sort.reader.thread	3	Maximum no of threads used for reading intermediate files for final merging.
carbon.load.metadata.lock.retries	3	Maximum number of retries to get the metadata lock for loading data to table.
carbon.load.metadata.lock.retry.timec	5	Interval between the retries to get the lock.
carbon.tempstore.location	/opt/Carbon/TempStoreLoc	Temporary store location. By default it takes System.getProperty(“java.io.tmpdir”).
carbon.load.log.counter	500000	Data loading records count logger.

- **Compaction Configuration**

Parameter	Default Value	Description

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.numberof.preserve.segments	0	If the user wants to preserve some number of segments from being compacted then he can set this property. Example: carbon.numberof.preserve.segments = 2 then 2 latest segments will always be excluded from the compaction. No segments will be preserved by default.
carbon.allowed.compaction.days	0	Compaction will merge the segments which are loaded with in the specific number of days configured. Example: If the configuration is 2, then the segments which are loaded in the time frame of 2 days only will get merged. Segments which are loaded 2 days apart will not be merged. This is disabled by default.
carbon.enable.auto.load.merge	false	To enable compaction while data loading.

- **Query Configuration**

Parameter	Default Value	Description
max.query.execution.time	60	Maximum time allowed for one query to be executed. The value is in minutes.
carbon.enableMinMax	true	Min max is feature added to enhance query performance. To disable this feature, set it false.

- **Global Dictionary Configurations**

Parameter	Default Value	Description
high.cardinality.identify.enable	true	If the parameter is true, the high cardinality columns of the dictionary code are automatically recognized and these columns will not be used as global dictionary encoding. If the parameter is false, all dictionary encoding columns are used as dictionary encoding. The high cardinality column must meet the following requirements: value of cardinality > configured value of high.cardinality. **Note:** If SINGLE_PASS is used during data load, then this property will be disabled.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

high.cardinality.threshold	1000000	It is a threshold to identify high cardinality of the columns.If the value of columns' cardinality > the configured value, then the columns are excluded from dictionary encoding.
carbon.cutOffTimestamp	1970-01-01 05:30:00	Sets the start date for calculating the timestamp. Java counts the number of milliseconds from start of "1970-01-01 00:00:00". This property is used to customize the start of position. For example "2000-01-01 00:00:00". The date must be in the form "carbon.timestamp.format". NOTE: The CarbonData supports data store up to 68 years from the cut-off time defined. For example, if the cut-off time is 1970-01-01 05:30:00, then the data can be stored up to 2038-01-01 05:30:00.
carbon.timegranularity	SECOND	The property used to set the data granularity level DAY, HOUR, MINUTE, or SECOND.

8.4 Spark Configuration

Spark Configuration Reference in spark-defaults.conf

Parameter	Default Value	Description
spark.driver.memory	1g	Amount of memory to be used by the driver process.
spark.executor.memory	1g	Amount of memory to be used per executor process.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

9 FAQs

FAQs

- What are Bad Records?
- Where are Bad Records Stored in CarbonData?
- How to enable Bad Record Logging?
- How to ignore the Bad Records?
- How to specify store location while creating carbon session?
- What is Carbon Lock Type?
- How to resolve Abstract Method Error?
- How Carbon will behave when execute insert operation in abnormal scenarios?

9.1 What are Bad Records?

Records that fail to get loaded into the CarbonData due to data type incompatibility or are empty or have incompatible format are classified as Bad Records.

9.2 Where are Bad Records Stored in CarbonData?

The bad records are stored at the location set in carbon.badRecords.location in carbon.properties file. By default **carbon.badRecords.location** specifies the following location `/opt/Carbon/Spark/badrecords`.

9.3 How to enable Bad Record Logging?

While loading data we can specify the approach to handle Bad Records. In order to analyse the cause of the Bad Records the parameter `BAD_RECORDS_LOGGER_ENABLE` must be set to value `TRUE`. There are multiple approaches to handle Bad Records which can be specified by the parameter `BAD_RECORDS_ACTION`.

- To pad the incorrect values of the csv rows with NULL value and load the data in CarbonData, set the following in the query : `'BAD_RECORDS_ACTION'='FORCE'`
- To write the Bad Records without padding incorrect values with NULL in the raw csv (set in the parameter **carbon.badRecords.location**), set the following in the query : `'BAD_RECORDS_ACTION'='REDIRECT'`

9.4 How to ignore the Bad Records?

To ignore the Bad Records from getting stored in the raw csv, we need to set the following in the query : `'BAD_RECORDS_ACTION'='IGNORE'`

9.5 How to specify store location while creating carbon session?

The store location specified while creating carbon session is used by the CarbonData to store the meta data like the schema, dictionary files, dictionary meta data and sort indexes.

Try creating `carbonsession` with `storepath` specified in the following manner :

```
val carbon = SparkSession.builder().config(sc.getConf)
             .getOrCreateCarbonSession(<store_path>)
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

Example:

```
val carbon = SparkSession.builder().config(sc.getConf)
             .getOrCreateCarbonSession("hdfs://localhost:9000/carbon/store")
```

9.6 What is Carbon Lock Type?

The Apache CarbonData acquires lock on the files to prevent concurrent operation from modifying the same files. The lock can be of the following types depending on the storage location, for HDFS we specify it to be of type HDFSLOCK. By default it is set to type LOCALLOCK. The property carbon.lock.type configuration specifies the type of lock to be acquired during concurrent operations on table. This property can be set with the following values : - **LOCALLOCK** : This Lock is created on local file system as file. This lock is useful when only one spark driver (thrift server) runs on a machine and no other CarbonData spark application is launched concurrently. - **HDFSLOCK** : This Lock is created on HDFS file system as file. This lock is useful when multiple CarbonData spark applications are launched and no ZooKeeper is running on cluster and the HDFS supports, file based locking.

9.7 How to resolve Abstract Method Error?

In order to build CarbonData project it is necessary to specify the spark profile. The spark profile sets the Spark Version. You need to specify the `spark version` while using Maven to build project.

9.8 How Carbon will behave when execute insert operation in abnormal scenarios?

Carbon support insert operation, you can refer to the syntax mentioned in DML Operations on CarbonData. First, create a soucre table in spark-sql and load data into this created table.

```
CREATE TABLE source_table(
id String,
name String,
city String)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ",";
```

```
SELECT * FROM source_table;
id  name    city
1   jack    beijing
2   erlu    hangzhou
3   davi    shenzhen
```

Scenario 1 :

Suppose, the column order in carbon table is different from source table, use script “SELECT * FROM carbon table” to query, will get the column order similar as source table, rather than in carbon table’s column order as expected.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
CREATE TABLE IF NOT EXISTS carbon_table(
id String,
city String,
name String)
STORED BY 'carbondata';
```

```
INSERT INTO TABLE carbon_table SELECT * FROM source_table;
```

```
SELECT * FROM carbon_table;
id  city    name
1   jack    beijing
2   erlu    hangzhou
3   davi    shenzhen
```

As result shows, the second column is city in carbon table, but what inside is name, such as jack. This phenomenon is same with insert data into hive table.

If you want to insert data into corresponding column in carbon table, you have to specify the column order same in insert statment.

```
INSERT INTO TABLE carbon_table SELECT id, city, name FROM source_table;
```

Scenario 2 :

Insert operation will be failed when the number of column in carbon table is different from the column specified in select statement. The following insert operation will be failed.

```
INSERT INTO TABLE carbon_table SELECT id, city FROM source_table;
```

Scenario 3 :

When the column type in carbon table is different from the column specified in select statement. The insert operation will still success, but you may get NULL in result, because NULL will be substitute value when conversion type failed.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

10 Troubleshooting

Troubleshooting

This tutorial is designed to provide troubleshooting for end users and developers who are building, deploying, and using CarbonData.

10.1 Failed to load thrift libraries

Symptom

Thrift throws following exception :

```
thrift: error while loading shared libraries: libthriftc.so.0: cannot open
shared object file: No such file or directory
```

Possible Cause

The complete path to the directory containing the libraries is not configured correctly.

Procedure

Follow the Apache thrift docs at https://thrift.apache.org/docs/install to install thrift correctly.

10.2 Failed to launch the Spark Shell

Symptom

The shell prompts the following error :

```
org.apache.spark.sql.CarbonContext$$anon$$apache$spark$sql$catalyst
$analysis $OverrideCatalog$_setter_$org$apache$spark$sql$catalyst$analysis
$OverrideCatalog$$overrides_$e
```

Possible Cause

The Spark Version and the selected Spark Profile do not match.

Procedure

1. Ensure your spark version and selected profile for spark are correct.
2. Use the following command :

```
"mvn -Pspark-2.1 -Dspark.version {yourSparkVersion} clean package"
```

Note : Refrain from using “mvn clean package” without specifying the profile.

10.3 Failed to execute load query on cluster.

Symptom

Load query failed with the following exception:

```
Dictionary file is locked for updation.
```

Possible Cause

The carbon.properties file is not identical in all the nodes of the cluster.

Procedure

Follow the steps to ensure the carbon.properties file is consistent across all the nodes:

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

1. Copy the carbon.properties file from the master node to all the other nodes in the cluster. For example, you can use ssh to copy this file to all the nodes.
2. For the changes to take effect, restart the Spark cluster.

10.4 Failed to execute insert query on cluster.

Symptom

Load query failed with the following exception:

```
Dictionary file is locked for updation.
```

Possible Cause

The carbon.properties file is not identical in all the nodes of the cluster.

Procedure

Follow the steps to ensure the carbon.properties file is consistent across all the nodes:

1. Copy the carbon.properties file from the master node to all the other nodes in the cluster. For example, you can use scp to copy this file to all the nodes.
2. For the changes to take effect, restart the Spark cluster.

10.5 Failed to connect to hiveuser with thrift

Symptom

We get the following exception :

```
Cannot connect to hiveuser.
```

Possible Cause

The external process does not have permission to access.

Procedure

Ensure that the Hiveuser in mysql must allow its access to the external processes.

10.6 Failed to read the metastore db during table creation.

Symptom

We get the following exception on trying to connect :

```
Cannot read the metastore db
```

Possible Cause

The metastore db is dysfunctional.

Procedure

Remove the metastore db from the carbon.metastore in the Spark Directory.

10.7 Failed to load data on the cluster

Symptom

Data loading fails with the following exception :

```
Data Load failure exeception
```

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

Possible Cause

The following issue can cause the failure :

1. The core-site.xml, hive-site.xml, yarn-site and carbon.properties are not consistent across all nodes of the cluster.
2. Path to hdfs ddl is not configured correctly in the carbon.properties.

Procedure

Follow the steps to ensure the following configuration files are consistent across all the nodes:

1. Copy the core-site.xml, hive-site.xml, yarn-site,carbon.properties files from the master node to all the other nodes in the cluster. For example, you can use scp to copy this file to all the nodes.

   Note : Set the path to hdfs ddl in carbon.properties in the master node.
2. For the changes to take effect, restart the Spark cluster.

10.8 Failed to insert data on the cluster

Symptom

Insertion fails with the following exception :

```
Data Load failure exeception
```

Possible Cause

The following issue can cause the failure :

1. The core-site.xml, hive-site.xml, yarn-site and carbon.properties are not consistent across all nodes of the cluster.
2. Path to hdfs ddl is not configured correctly in the carbon.properties.

Procedure

Follow the steps to ensure the following configuration files are consistent across all the nodes:

1. Copy the core-site.xml, hive-site.xml, yarn-site,carbon.properties files from the master node to all the other nodes in the cluster. For example, you can use scp to copy this file to all the nodes.

   Note : Set the path to hdfs ddl in carbon.properties in the master node.
2. For the changes to take effect, restart the Spark cluster.

10.9 Failed to execute Concurrent Operations(Load,Insert,Update) on table by multiple workers.

Symptom

Execution fails with the following exception :

```
Table is locked for updation.
```

Possible Cause

Concurrency not supported.

Procedure

Worker must wait for the query execution to complete and the table to release the lock for another query execution to succeed.

10.10 Failed to create a table with a single numeric column.

Symptom

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

Execution fails with the following exception :

```
Table creation fails.
```

Possible Cause

Behaviour not supported.

Procedure

A single column that can be considered as dimension is mandatory for table creation.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

11 Useful Tips

Useful Tips

This tutorial guides you to create CarbonData Tables and optimize performance. The following sections will elaborate on the above topics :

- Suggestions to create CarbonData Table
- Configuration for Optimizing Data Loading performance for Massive Data
- Optimizing Mass Data Loading

11.1 Suggestions to Create CarbonData Table

Recently CarbonData was used to analyze performance of Telecommunication field. The results of the analysis for table creation with dimensions ranging from 10 thousand to 10 billion rows and 100 to 300 columns have been summarized below.

The following table describes some of the columns from the table used.

Table Column Description

Column Name	Data Type	Cardinality	Attribution
msisdn	String	30 million	Dimension
BEGIN_TIME	BigInt	10 Thousand	Dimension
HOST	String	1 million	Dimension
Dime_1	String	1 Thousand	Dimension
counter_1	Numeric(20,0)	NA	Measure
…	…	NA	Measure
counter_100	Numeric(20,0)	NA	Measure

CarbonData has more than 50 test cases, on the basis of these we have following suggestions to enhance the query performance :

- **Put the frequently-used column filter in the beginning**

For example, MSISDN filter is used in most of the query then we must put the MSISDN in the first column. The create table command can be modified as suggested below :

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
create table carbondata_table(
msisdn String,
...
)STORED BY 'org.apache.carbondata.format'
TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN,..',
'DICTIONARY_INCLUDE'='...');

Example:
create table carbondata_table(
  msisdn String,
  BEGIN_TIME bigint
  )STORED BY 'org.apache.carbondata.format'
  TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN',
  'DICTIONARY_INCLUDE'='BEGIN_TIME');
```

Now the query with MSISDN in the filter will be more efficient.

- **Put the frequently-used columns in the order of low to high cardinality**

If the table in the specified query has multiple columns which are frequently used to filter the results, it is suggested to put the columns in the order of cardinality low to high. This ordering of frequently used columns improves the compression ratio and enhances the performance of queries with filter on these columns.

For example if MSISDN, HOST and Dime_1 are frequently-used columns, then the column order of table is suggested as Dime_1>HOST>MSISDN as Dime_1 has the lowest cardinality. The create table command can be modified as suggested below :

```
create table carbondata_table(
Dime_1 String,
HOST String,
MSISDN String,
...
)STORED BY 'org.apache.carbondata.format'
TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN,HOST..',
'DICTIONARY_INCLUDE'='Dime_1..');

Example:
create table carbondata_table(
  Dime_1 String,
  HOST String,
  MSISDN String
  )STORED BY 'org.apache.carbondata.format'
  TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN,HOST',
  'DICTIONARY_INCLUDE'='Dime_1');
```

- **Put the Dimension type columns in order of low to high cardinality**

If the columns used to filter are not frequently used, then it is suggested to order all the columns of dimension type in order of low to high cardinality. The create table command can be modified as below :

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
create table carbondata_table(
  Dime_1 String,
  BEGIN_TIME bigint,
  END_TIME bigint,
  HOST String,
  MSISDN String
  ...
  )STORED BY 'org.apache.carbondata.format'
  TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN,HOST...',
  'DICTIONARY_INCLUDE'='Dime_1,END_TIME,BEGIN_TIME...');
```

- **For measure type columns with non high accuracy, replace Numeric(20,0) data type with Double data type**

For columns of measure type, not requiring high accuracy, it is suggested to replace Numeric data type with Double to enhance query performance. The create table command can be modified as below :

```
create table carbondata_table(
  Dime_1 String,
  BEGIN_TIME bigint,
  END_TIME bigint,
  HOST String,
  MSISDN String,
  counter_1 double,
  counter_2 double,
  ...
  counter_100 double
  )STORED BY 'org.apache.carbondata.format'
  TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN,HOST...',
  'DICTIONARY_INCLUDE'='Dime_1,END_TIME,BEGIN_TIME...');
```

The result of performance analysis of test-case shows reduction in query execution time from 15 to 3 seconds, thereby improving performance by nearly 5 times.

- **Columns of incremental character should be re-arranged at the end of dimensions**

Consider the following scenario where data is loaded each day and the begin_time is incremental for each load, it is suggested to put begin_time at the end of dimensions.

Incremental values are efficient in using min/max index. The create table command can be modified as below :

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
create table carbondata_table(
  Dime_1 String,
  HOST String,
  MSISDN String,
  counter_1 double,
  counter_2 double,
  BEGIN_TIME bigint,
  END_TIME bigint,
  ...
  counter_100 double
  )STORED BY 'org.apache.carbondata.format'
  TBLPROPERTIES ( 'DICTIONARY_EXCLUDE'='MSISDN,HOST...',
  'DICTIONARY_INCLUDE'='Dime_1,END_TIME,BEGIN_TIME....');
```

- **Avoid adding high cardinality columns to dictionary**

If the system has low memory configuration, then it is suggested to exclude high cardinality columns from the dictionary to enhance load performance. Creation of dictionary for high cardinality columns at time of load will degrade load performance due to excessive memory usage.

By default CarbonData determines the cardinality at the first data load and allows for dictionary creation only if the cardinality is less than 1 million.

11.2 Configuration for Optimizing Data Loading performance for Massive Data

CarbonData supports large data load, in this process sorting data while loading consumes a lot of memory and disk IO and this can result sometimes in “Out Of Memory” exception. If you do not have much memory to use, then you may prefer to slow the speed of data loading instead of data load failure. You can configure CarbonData by tuning following properties in carbon.properties file to get a better performance.

Parameter	Default Value	Description/Tuning
carbon.number.of.cores.while.loading	Default: 2.This value should be >= 2	Specifies the number of cores used for data processing during data loading in CarbonData.
carbon.sort.size	Default: 100000. The value should be >= 100.	Threshold to write local file in sort step when loading data
carbon.sort.file.write.buffer.size	Default: 50000.	DataOutputStream buffer.
carbon.number.of.cores.block.sort	Default: 7	If you have huge memory and cpus, increase it as you will
carbon.merge.sort.reader.thread	Default: 3	Specifies the number of cores used for temp file merging during data loading in CarbonData.
carbon.merge.sort.prefetch	Default: true	You may want set this value to false if you have not enough memory

For example, if there are 10 million records ,and i have only 16 cores ,64GB memory, will be loaded to CarbonData table. Using the default configuration always fail in sort step. Modify carbon.properties as suggested below:

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

```
carbon.number.of.cores.block.sort=1
carbon.merge.sort.reader.thread=1
carbon.sort.size=5000
carbon.sort.file.write.buffer.size=5000
carbon.merge.sort.prefetch=false
```

11.3 Configurations for Optimizing CarbonData Performance

Recently we did some performance POC on CarbonData for Finance and telecommunication Field. It involved detailed queries and aggregation scenarios. After the completion of POC, some of the configurations impacting the performance have been identified and tabulated below :

Parameter	Location	Used For	Description	Tuning
carbon.sort.intermedia	spark/carbonlib/ carbon.properties	Data loading	During the loading of data, local temp is used to sort the data. This number specifies the minimum number of intermediate files after which the merge sort has to be initiated.	Increasing the parameter to a higher value will improve the load performance. For example, when we increase the value from 20 to 100, it increases the data load performance from 35MB/S to more than 50MB/S. Higher values of this parameter consumes more memory during the load.
carbon.number.of.core	spark/carbonlib/ carbon.properties	Data loading	Specifies the number of cores used for data processing during data loading in CarbonData.	If you have more number of CPUs, then you can increase the number of CPUs, which will increase the performance. For example if we increase the value from 2 to 4 then the CSV reading performance can increase about 1 times

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.compaction.le\	spark/carbonlib/ carbon.properties	Data loading and Querying	For minor compaction, specifies the number of segments to be merged in stage 1 and number of compacted segments to be merged in stage 2.	Each CarbonData load will create one segment, if every load is small in size it will generate many small file over a period of time impacting the query performance. Configuring this parameter will merge the small segment to one big segment which will sort the data and improve the performance. For Example in one telecommunication scenario, the performance improves about 2 times after minor compaction.
spark.sql.shuffle.partit	spark/conf/spark-defaults.conf	Querying	The number of task started when spark shuffle.	The value can be 1 to 2 times as much as the executor cores. In an aggregation scenario, reducing the number from 200 to 32 reduced the query time from 17 to 9 seconds.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

spark.executor.instanc spark.executor.cores/ spark.executor.memo	spark/conf/spark-defaults.conf	Querying	The number of executors, CPU cores, and memory used for CarbonData query.	In the bank scenario, we provide the 4 CPUs cores and 15 GB for each executor which can get good performance. This 2 value does not mean more the better. It needs to be configured properly in case of limited resources. For example, In the bank scenario, it has enough CPU 32 cores each node but less memory 64 GB each node. So we cannot give more CPU but less memory. For example, when 4 cores and 12GB for each executor. It sometimes happens GC during the query which impact the query performance very much from the 3 second to more than 15 seconds. In this scenario need to increase the memory or decrease the CPU cores.
carbon.detail.batch.siz	spark/carbonlib/carbon.properties	Data loading	The buffer size to store records, returned from the block scan.	In limit scenario this parameter is very important. For example your query limit is 1000. But if we set this value to 3000 that means we get 3000 records from scan but spark will only take 1000 rows. So the 2000 remaining are useless. In one Finance test case after we set it to 100, in the limit 1000 scenario the performance increase about 2 times in comparison to if we set this value to 12000.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.

carbon.use.local.dir	spark/carbonlib/ carbon.properties	Data loading	Whether use YARN local directories for multi-table load disk load balance	If this is set it to true CarbonData will use YARN local directories for multi-table load disk load balance, that will improve the data load performance.
carbon.use.multiple.te	spark/carbonlib/ carbon.properties	Data loading	Whether to use multiple YARN local directories during table data loading for disk load balance	After enabling 'carbon.use.local.dir', if this is set to true, CarbonData will use all YARN local directories during data load for disk load balance, that will improve the data load performance. Please enable this property when you encounter disk hotspot problem during data loading.

Note: If your CarbonData instance is provided only for query, you may specify the property 'spark.speculation=true' which is in conf directory of spark.

©2017, The Apache Software Foundation • ALL RIGHTS RESERVED.