Remove segment when all elements are deleted

Objective

To mark the segment as ‘marked for delete’ instead of creating a delete delta file when all the elements in that segment is deleted.

Background

The present design is to add a delete delta file for any mutation operation for all the rows updated. The writing of a delete delta file for a delete operation can be avoided if all the elements in the segments are deleted. Instead, the whole segment can be marked for delete. The segement which are marked for delete will be deleted during the compaction process.

Design Approach

For each segment, the total number of rows can be matched with the deleted number of rows. The total number of rows in each segment can be fetched from the mapping of segment and block count in the CarbonTableInputFormat.

- CarbondataFileWriter.getRowCount(String segmentId)

```
private long getRowCount(String segmentId) throws IOException{
    long rowCount = 0;
    String tableName = carbonLoadModel.getTableName();
    String databaseName = carbonLoadModel.getDatabaseName();
    CarbonTable carbonTable = SchemaReader.readCarbonTableFromStore(AbsoluteTableIdentifier.from(tablePath,databaseName,tableName));
    partitionInfo = carbonTable.getPartitionInfo();
    LoadMetadataDetails[] loadMetadataDetails;
    List<PartitionSpec> partitionSpecs = null;
    if (partitionInfo != null && partitionInfo.getPartitionType() == PartitionType.NATIVE_HIVE) {
        loadMetadataDetails = SegmentStatusManager.readTableStatusFile(
                CarbonTablePath.getTableStatusFilePath(carbonTable.getTablePath()));
        partitionSpecs = new ArrayList<>();
        for (LoadMetadataDetails loadMetadataDetail : loadMetadataDetails) {
            SegmentFileStore segmentFileStore;
            segmentFileStore =
                    new SegmentFileStore(carbonTable.getTablePath(), loadMetadataDetail.getSegmentFile());
            partitionSpecs.addAll(segmentFileStore.getPartitionSpecs());
        }
    }
    configuration.set(CarbonTableInputFormat.INPUT_DIR,tablePath);
    configuration.set(CarbonTableInputFormat.DATABASE_NAME,databaseName);
    configuration.set(CarbonTableInputFormat.TABLE_NAME,tableName);
    Job job = new Job(configuration);
    CarbonTableInputFormat carbonInputFormat = createCarbonInputFormat(AbsoluteTableIdentifier.from(tablePath,databaseName,tableName));
    BlockMappingVO blockMappingVO = carbonInputFormat.getBlockRowCount(job,carbonTable,partitionSpecs, isUpdateFlow: true);
    segmentNoRowCountMapping = getSegmentNoRowCountMapping(blockMappingVO);
    rowCount = segmentNoRowCountMapping.get(segmentId);

    return rowCount;
}
```

If they are matching, we can mark the segment for delete and skip the write for delete delta file.

- CarbondataFileWriter.appendRows(Page dataPage, int position);

```
long rowCount = getRowCount(segmentId);
deleteSegmentMap.computeIfAbsent(deltaPath,
        v -> rowCount == deletedRows? true : false
);

if (rowCount == deletedRows && (HiveACIDWriteType.DELETE == acidWriteType || isHivePartitionedTable)) {
    segmentUpdateDetailMap.get(segmentBlockId).setSegmentStatus(SegmentStatus.MARKED_FOR_DELETE);
}
```

The check is done only for delete operation for non-partitioned tables. For other mutation operations like update the row count of the segment will be the same before and after the operation.

For a partitioned table, however, the new records will be added to a new segment. Hence the row count can be reduced with both delete and update operations. Therefore segments can be deleted for update and delete of a partitioned table.

- CarbondataFileWriter.commit()

```
@Override
public void commit()
{
    try {
        if (HiveACIDWriteType.isUpdateOrDelete(acidWriteType)) {
            deleteDeltaDetailsMap.forEach((k, v) -> {
                try {
                    if(!deleteSegmentMap.get(k)) {
                        (new CarbonDeleteDeltaWriterImpl(k)).write(v);
                    }
                }
                catch (IOException e) {
                    LOG.error( message: "Error while writing the deleteDeltas ", e
                    throw new PrestoException(GENERIC_INTERNAL_ERROR, "Error whi
                }
            });
        }
```