Hive Indexes

HIVE INDEXES

What is an Index ?

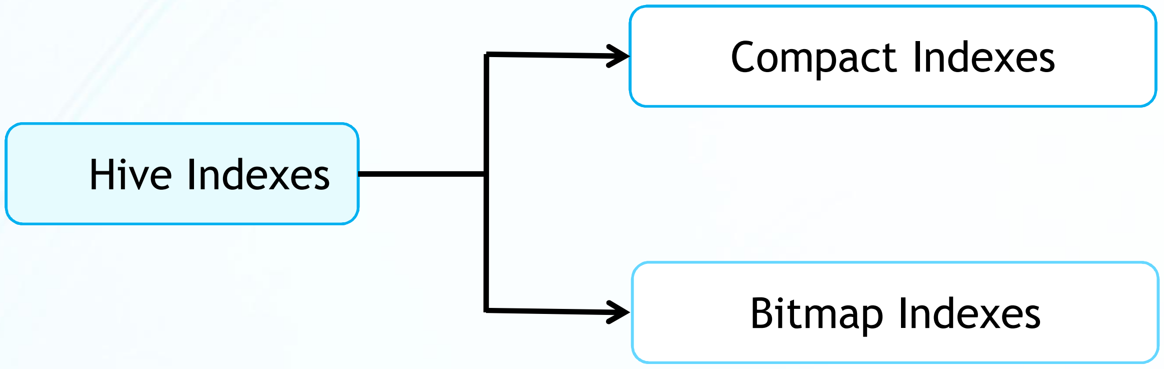
- An index is a reference of records inside a table.
- Instead of searching all the records, we can refer to the index to search for a particular record. This usually speeds up the searching of data on the indexed column with minimal overhead.

Advantage of Indexes in Hive

- As Hive deals with Big Data, the size of files tend to be very big. If we want to perform any query on this huge amount of data, it may take lot of time as the queries has to scan all the data files to fetch the data.
- In a Hive table, there are many numbers of rows and columns. If we want to perform queries only on some columns without indexing, it will take large amount of time because queries will be executed on all the columns present in the table.
- If we maintain indexes, it will be easier for Hive query to look into the indexes first and then perform the needed operations within less amount of time.

Hive Indexes

- Hive has limited indexing capabilities.
- There are no keys in the usual relational database sense, but you can build an index on columns to speed some operations.
- The index data for a table is stored in another table.



Hive Indexes

- Indexing is also a good alternative to partitioning when the logical partitions would actually be too numerous and small to be useful.
- Indexing can aid in pruning some blocks from a table as input for a MapReduce job.
- Not all queries can benefit from an index—the EXPLAIN syntax and Hive can be used to determine if a given query is aided by an index.

Indexes in Hive need to be evaluated carefully. Maintaining an index requires extra disk space and building an index has a processing cost. The user must weigh these costs against the benefits they offer when querying a table.

Compact Vs. Bitmap Indexes

- Compact index stores the pair of indexed column’s value and its block-id.
- Bitmap index stores the combination of indexed column value and list of rows as a bitmap. **Bitmap indexing** is a standard technique for indexing columns with few distinct values.

- The main difference is the storing of the mapped values of the rows.
- As the data inside a Hive table is stored in HDFS, they are distributed across the nodes in a cluster. There needs to be a proper identification of the data, like the data in block indexing.
- This data will be able to identity which row is present in which block, so that when a query is triggered it can go directly into that block. So, while performing a query, it will first check the index and then go directly into that block.

Create Compact Index

```
CREATE INDEX employees_index
ON TABLE employees (country)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD
```

```
CREATE INDEX employees_index
ON TABLE employees (country)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD
IDXPROPERTIES ('creator = 'me', 'created_at' = 'some_time')
IN TABLE employees_index_table
PARTITIONED BY (country, name)
COMMENT 'Employees indexed by country and name.';
```

Create Bitmap Index

```
CREATE INDEX users_index_bitmap
ON TABLE users (age)
AS 'BITMAP'
WITH DEFERRED REBUILD;
```

Rebuilding the Index

- If you specified **`WITH DEFERRED REBUILD`**, the new index starts empty. At any time, the index can be built the first time or rebuilt using the **`ALTER INDEX`** statement.

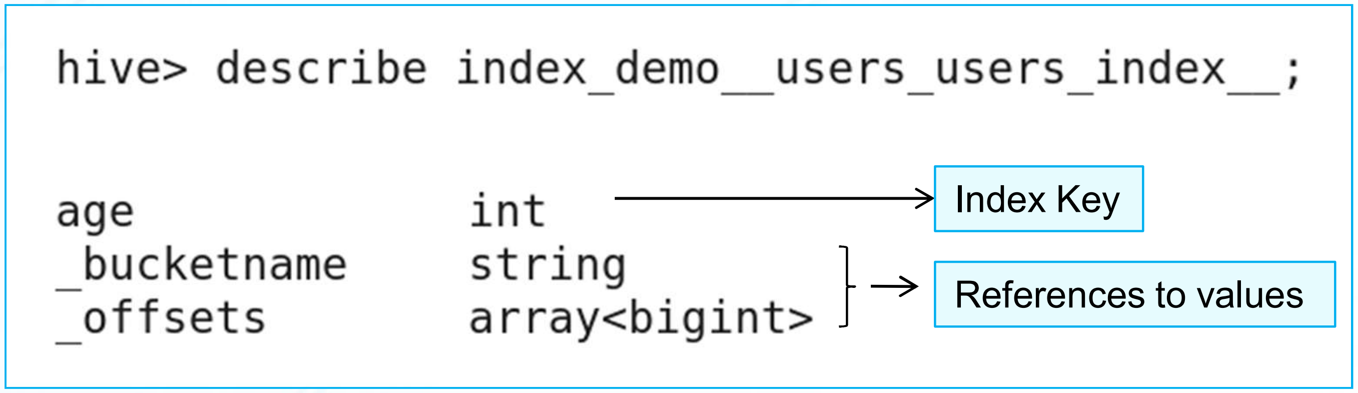
  - **`ALTER INDEX users_index_bitmap on users REBUILD`**;

- There is no built-in mechanism to trigger an automatic rebuild of the index if the underlying table or a particular partition changes.

Rebuilding the Index

- If `WITH DEFERRED REBUILD` is specified on `CREATE INDEX`, then the newly created index is initially empty (regardless of whether the table contains any data).
- The `ALTER INDEX ... REBUILD` command can be used to build the index structure for all partitions or a single partition.
- If data in the base table changes, then the `REBUILD` command must be used to bring the index up to date.
- This is an atomic operation, so if the table was previously indexed, and a rebuild fails, then the stale index remains untouched.

How does an index look like?

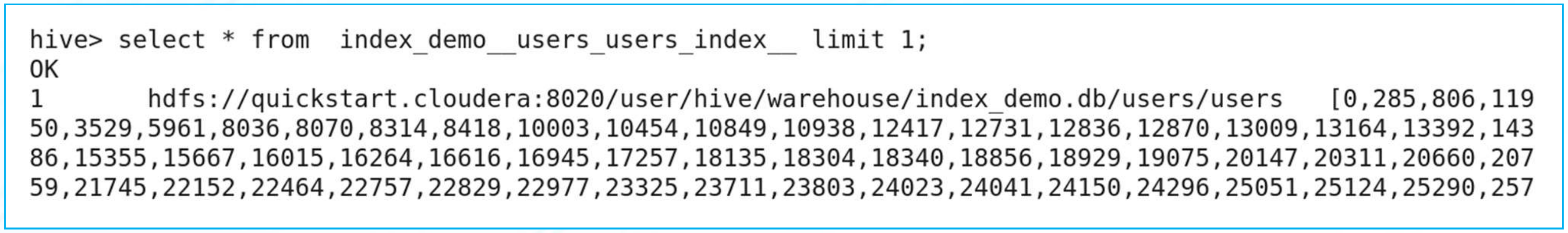


```
hive> describe index_demo__users_users_index__;

age                int
_bucketname        string
_offsets           array<bigint>
```

In the above schema, *bucketname* refers to the file(s) in which we have a given index key and *offsets* refer to the offsets of the matching rows in that bucket.

How does an index look like?



```
hive> select * from  index_demo__users_users_index__  limit 1;
OK
1       hdfs://quickstart.cloudera:8020/user/hive/warehouse/index_demo.db/users/users   [0,285,806,119
50,3529,5961,8036,8070,8314,8418,10003,10454,10849,10938,12417,12731,12836,12870,13009,13164,13392,143
86,15355,15667,16015,16264,16616,16945,17257,18135,18304,18340,18856,18929,19075,20147,20311,20660,207
59,21745,22152,22464,22757,22829,22977,23325,23711,23803,24023,24041,24150,24296,25051,25124,25290,257
```

In the above picture, the data of the index table shows, the index key (i.e age = 1), the bucket (file location) and offsets with in that file where all matching records for that key are found.

List indexes of a table

```
•  show formatted index on users
```

- The above command lists all the indexes created for a given table (‘users’ in this case)
- With different types (compact,bitmap) of indexes on the same columns, for the same table, the index which is created first is taken as the index for that table on the specified columns.

Drop an Index

```
• DROP INDEX IF EXISTS users_index ON users;
```

THANK YOU