Blog

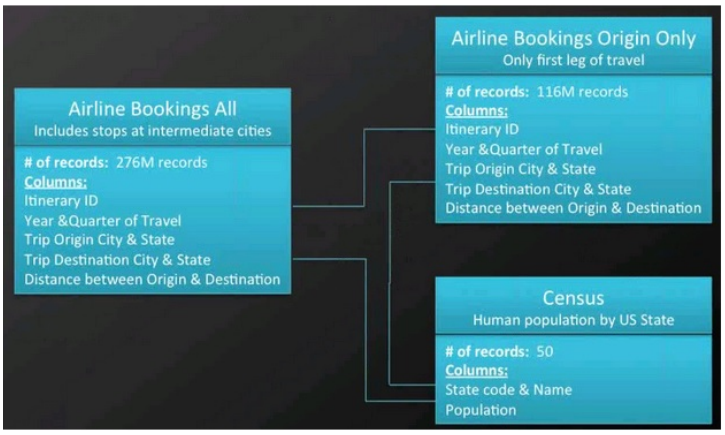
5 Tips for efficient Hive queries with Hive Query Language

By sshaik@qubole.com | October 18, 2013 | Hive Query (https://www.qubole.com/tag/hive-query/)

Hive on Hadoop makes data processing so straightforward and scalable that we can easily forget to optimize our Hive queries. Well designed tables and queries can greatly improve your query speed and reduce processing cost. This article includes five tips, which are valuable for ad-hoc queries (http://www.qubole.com/resources/articles/ad-hoc-analysis/), to save time, as much as for regular ETL (Extract, Transform, Load) workloads, to save money. The three areas in which we can optimize our Hive utilization are:

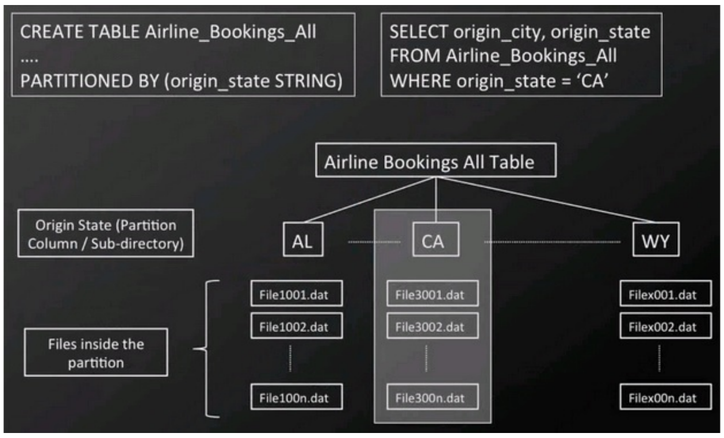
- Data Layout (Partitions and Buckets)
- Data Sampling (Bucket and Block sampling)
- Data Processing (Bucket Map Join and Parallel execution)

We will discuss these areas in detail in this article, you can if like, also watch our webinar on the topic given by Ashish Thusoo, co-founder of Apache Hive, and Sadiq Sid Shaik, Director of Product at Qubole. Example Data Set We can illustrate the improvements best on an example data set we use at Qubole. The data consists of three tables. The table Airline Bookings All contains 276 million records of complete air travel trips from an origin to a destination with an itinerary identifier as key. The second table Airline Bookings Origin Only contains the data for the first leg of an itinerary only and also has the itinerary's identifier as a key. The last table is 'Census' containing population information for each US state.


Airline Bookings All
Includes stops at intermediate cities
# of records: 276M records
Columns:
Itinerary ID
Year &Quarter of Travel
Trip Origin City & State
Trip Destination City & State
Distance between Origin & Destination
Airline Bookings Origin Only
Only first leg of travel
# of records: 116M records
Columns:
Itinerary ID
Year &Quarter of Travel
Trip Origin City & State
Trip Destination City & State
Distance between Origin & Destination
Census
Human population by US State
# of records: 50
Columns:
State code & Name
Population

The example data set to demonstrate Hive query language optimization (https://www.qubole.com/resources/webinars/5-tips-creating-effective-hive-queries-secrets-pros/)

Tip 1: Partitioning Hive Tables Hive is a powerful tool to perform queries on large data sets and it is particularly good at queries that require full table scans. Yet many queries run on Hive have filtering where clauses limiting the data to be retrieved and processed, e.g. SELECT * WHERE state='CA'. Hive users tend to have or develop a domain knowledge, understand the data they work with and the queries commonly executed or scheduled. With this knowledge we can identify common data structures that surface in queries. This enables us to identify columns with a (relatively) low cardinality like geographies or dates and high relevance to key queries. For example, common approaches to slice the airline data may be by origin state for reporting purposes. We can utilize this knowledge to organise our data by this information and tell Hive about it. Hive can utilize this knowledge to exclude data from queries before even reading it. Hive tables are linked to directories on HDFS or S3 with files in them interpreted by the meta data stored with Hive. Without partitioning Hive reads all the data in the directory and applies the query filters on it. This is slow and expensive since all data has to be read. In our example a common reports and queries might be generated on an origin state basis. This enables us to define at creation time of the table the state column to be a partition. Consequently, when we write data to the table the data will be written in sub-directories named by state (abbreviations). Subsequently, queries filtering by origin state, e.g. SELECT * FROM Airline_Bookings_All WHERE origin_state = 'CA', allow Hive to skip all but the relevant sub-directories and data files. This can lead to tremendous reduction in data required to read and filter in the initial map stage. This reduces the number of mappers, IO operations, and time to answer the query.

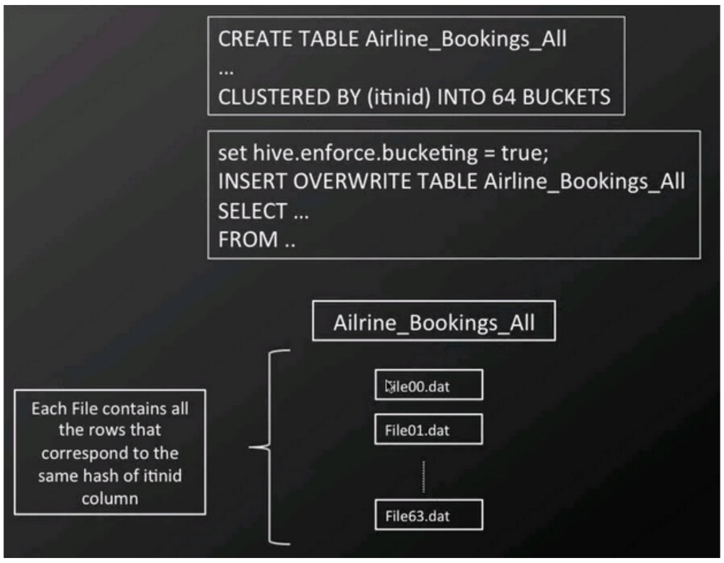


```
CREATE TABLE Airline_Bookings_All
....
PARTITIONED BY (origin_state STRING)
```

```
SELECT origin_city, origin_state
FROM Airline_Bookings_All
WHERE origin_state = 'CA'
```

Example Hive table partitioning It is important to consider the cardinality of a potential partition column and avoid fragmenting the data too much. Itinerary ID would be a very poor choice for partitioning. Queries for single itineraries by ID would be very fast but any other query would require to parse a huge amount of directories and files incurring serious overheads. Additionally, HDFS uses a very large block size of usually 64 MB or more which means that each file, even with only a few bytes of data, will have to allocate that block size on HDFS. This can potentially fill the file system up with large number of files carrying barely any actual data.

Tip 2: Bucketing Hive Tables Itinerary ID is unsuitable for partitioning as we learned but it is used frequently for join operations. We can optimize joins by bucketing 'similar' IDs so Hive can minimise the processing steps, and reduce the data needed to parse and compare for join operations. Itinerary IDs, of course, have no real similarity and we only need to achieve that the same itinerary IDs from two tables end up in the same processing bucket. A simple trick to do this is to hash the data and store it by hash results, which is what bucketing does.

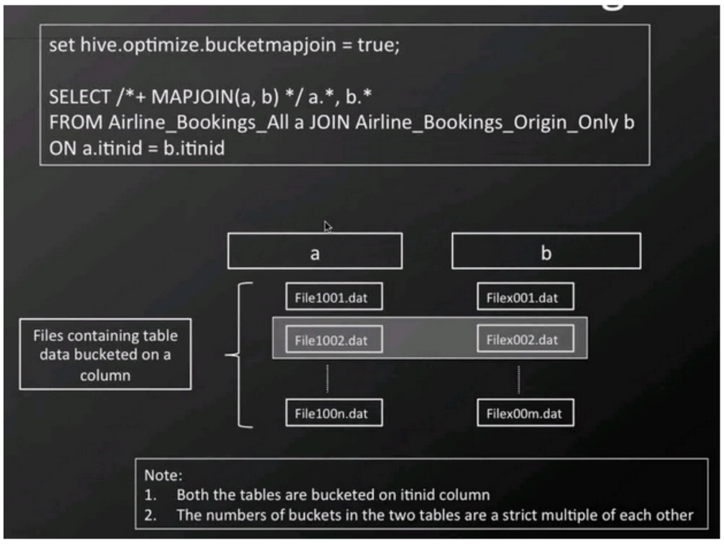


```
CREATE TABLE Airline_Bookings_All
...
CLUSTERED BY (itinid) INTO 64 BUCKETS
```

```
set hive.enforce.bucketing = true;
INSERT OVERWRITE TABLE Airline_Bookings_All
SELECT ...
FROM ..
```

Ailrine_Bookings_All
Each File contains all the rows that correspond to the same hash of itinid column
File00.dat
File01.dat
File63.dat

Example Hive query table bucketing Bucketing requires us to tell Hive at table creation time by which column to cluster by and into how many buckets. We also have to ensure the bucketing flag is set (SET hive.enforce.bucketing=true;) every time before we write data to the bucketed table. Importantly, the corresponding tables we want to join on have to be set up in the same manner with the joining columns bucketed and the bucket sizes being multiples of each other to work. The second part is the optimized query for which we have to set a flag to hint to Hive that we want to take advantage of the bucketing in the join (SET hive.optimize.bucketmapjoin=true;). The SELECT statement then can include a MAPJOIN statement to ensure that the join operation is executed at the map stage by combining only the few relevant files in each mapper task in a distributed fashion from the two tables instead of parsing the full tables. Example Hive MAPJOIN with bucketing

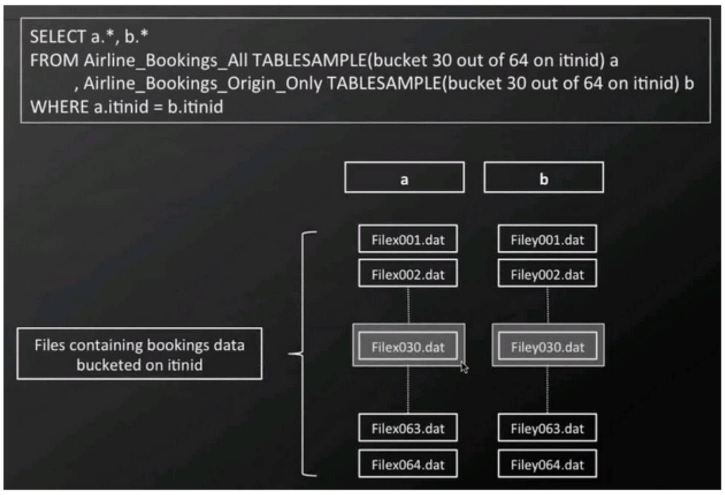


```
set hive.optimize.bucketmapjoin = true;

SELECT /*+ MAPJOIN(a, b) */ a.*, b.*
FROM Airline_Bookings_All a JOIN Airline_Bookings_Origin_Only b
ON a.itinid = b.itinid
```

Note:
1. Both the tables are bucketed on itinid column
2. The numbers of buckets in the two tables are a strict multiple of each other

Tip 3: Bucket Sampling Once our tables are setup with these buckets we can address another important use-case. We often want to query large table joins for a sample. We may want to try out complex queries or explore the data, and we want to do this iteratively, swiftly, and not process the whole data set. This is particularly difficult because of the joining of the tables since only very little data may overlap on independent samples from two tables. Ideally we would want to sample the relevant data on both tables and join it, i.e. ensure that we sample the same itinerary IDs from both tables and not sets with no or little overlap. The bucketing on the join column enables us to join specific buckets from two tables with data overlapping on the join column. Effectively, we execute exactly one part of the complete join operation and only incur the cost of it. The hashing function on the ID has the additional benefit of a (somewhat) random nature providing a representative sample.

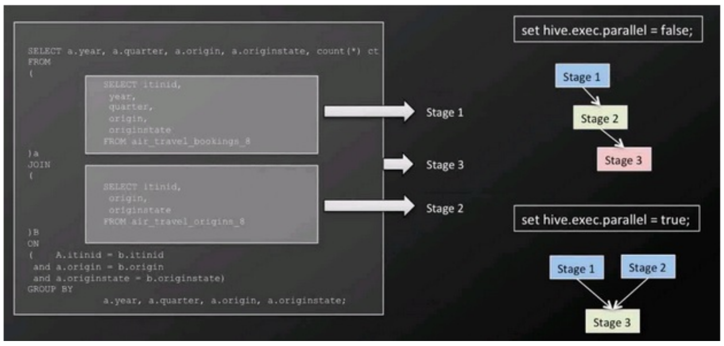


```
SELECT a.*, b.*
FROM Airline_Bookings_All TABLESAMPLE(bucket 30 out of 64 on itinid) a
      , Airline_Bookings_Origin_Only TABLESAMPLE(bucket 30 out of 64 on itinid) b
WHERE a.itinid = b.itinid
```

Example Hive TABLESAMPLE on bucketed tables

Tip 4: Block Sampling Similarly, to the previous tip, we often want to sample data from only one table to explore queries and data. In these cases we may not want to go through bucketing the table or we have the need to sample the data more randomly (independent from the hashing of a bucketing column) or at decreasing granularity. Block sampling provides a powerful syntax to define various ways of sampling the data in a table with the TABLESAMPLE statement. We can use it to sample a certain percentage, number of bytes, or rows of the data. We can use the sampling to approximate information like average distance between origin and destination of our itineraries. A query using 1% of the data using TABLESAMPLE(1 PERCENT) on a large table will give us a near perfect answer and use up to only a hundredth of the resource and return the result one to two magnitudes faster. In exploratory work or for metrics this approach can be extremely efficient and effective alternative to processing all of the data. The beauty of this solution is that we can scale the sample size with our data size. If we were to explore at Tera- or Petabytes of data we could sample a fraction of percent and get the same actionable information in minutes or less which would otherwise take hours to receive.

Tip 5: Parallel Execution Hadoop can execute map reduce (http://www.qubole.com/mapreduce-as-a-service/) jobs in parallel and several queries executed on Hive make automatically use of this parallelism. However, single, complex Hive queries commonly are translated to a number of map reduce jobs that are executed by default sequentially. Often though some of a query's map reduce stages are not interdependent and could be executed in parallel. They then can take advantage of spare capacity on a cluster and improve cluster utilization while at the same time reduce the overall query executions time. The configuration in Hive to change this behaviour is a merely switching a single flag SET hive.exce.parallel=true;.



```
SELECT a.year, a.quarter, a.origin, a.originstate, count(*) ct
FROM
(
            SELECT itinid,
             year,
             quarter,
             origin,
             originstate
            FROM air_travel_bookings_8
)a
JOIN
(
            SELECT itinid,
             origin,
             originstate
            FROM air_travel_origins_8
)B
ON
(    A.itinid = b.itinid
 and a.origin = b.origin
 and a.originstate = b.originstate)
GROUP BY
            a.year, a.quarter, a.origin, a.originstate;
```

```
set hive.exec.parallel = false;
```

```
set hive.exec.parallel = true;
```

Example of Hive parallel stage execution of a query In our example in the image above we can see that the two sub-queries are independent and when we enable parallel execution are processed at the same time. In our example this reduced the execution time by 50%! Conclusion The five presented tips in this article can easily be applied by anyone using Hive to improve processing and query speed and reduce resource consumption. You can, of course, use them with the Qubole platform (http://www.qubole.com/features/) and are welcome to try it out.

BIG DATA IN THE CLOUD
SUCCESS SHEET
BIG DATA IN THE CLOUD
SUCCESS SHEET
Considering big data in the cloud? Review these six critical factors for success with your project.
DOWNLOAD NOW

(https://cta-service-cms2.hubspot.com/ctas/v2/public/cs/c/?cta_guid=ab572ee2-9164-4ba2-952a-5f010751bde7&placement_guid=ba736405-a440-4614-9156-701730a2fe3b&portal_id=284677&redirect_url=APefjpG-4WCBUV6uVKF0kfuW_lqTz0NWHznwcCYOw5eLXcewu8EfYobIY_bmeXGLa88qAaPSE8INARc1NAczxXs5u6kCJJ5HfE7Db2jpJRKvpanFu20439QZOeMbqCmL0ysnfgxQLdQ7LUKA&hsutk=&canon=https%3A%2F%2Fwww.qubole.com%2Fblog%2Fbig-data%2F5-tips-for-efficient-hive-queries%2F&__hstc=98673089.272f90663bdddd7f6065478c05172a4f.1472846865663.1472846865663.1472846865663.1&__hssc=98673089.1.1472846865664

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Are you human? *

three × = **twenty seven**

Post Comment

Live Session

(http://bit.ly/2cbRkXh)

On-demand Webinar

ON-DEMAND WEBINAR
Learn How Qubole Can Help Manage & Analyze Your Big Data Faster
powered by amazon web services
Qubole
Register Now

(http://bit.ly/2aDEMS6)

Cloudera Migration Program

CONSIDERING MOVING TO THE CLOUD FROM CLOUDERA

Qubole's partnered with WANdisco to create a seamless migration progress.

LEARN MORE

(http://bit.ly/2arN94p)

Get Blog Updates

Subscribe to Big Data Articles by Email

Email *

Subscribe

Search Blog

Enter keywords ...

Featured Blogs

Tags

Analysis (https://www.qubole.com/tag/analysis/) Apache Hadoop (https://www.qubole.com/tag/apache-hadoop/)

Apache Hive (https://www.qubole.com/tag/apache-hive/) Auto Scaling (https://www.qubole.com/tag/auto-scaling/)

AWS (https://www.qubole.com/tag/aws/) Benchmarks (https://www.qubole.com/tag/benchmarks/)

Big Data Uses (https://www.qubole.com/tag/big-data-uses/) Business Intelligence (https://www.qubole.com/tag/business-intelligence/)

Canonicalization (https://www.qubole.com/tag/canonicalization/) Case Studies (https://www.qubole.com/tag/case-studies/)

Cloud (https://www.qubole.com/tag/cloud/) Dashboard (https://www.qubole.com/tag/dashboard/)

Data Application (https://www.qubole.com/tag/data-app/) EBS (https://www.qubole.com/tag/ebs/) Engine (https://www.qubole.com/tag/engine/)

Events (https://www.qubole.com/tag/events/) Google (https://www.qubole.com/tag/google/) Hadoop (https://www.qubole.com/tag/hadoop/)

Hadoop Happenings (https://www.qubole.com/tag/hadoop-happenings/) Hadoop Influencer (https://www.qubole.com/tag/hadoop-influencer-posts/)

HDFS (https://www.qubole.com/tag/hdfs/) Hive (https://www.qubole.com/tag/hive/) Hive Query (https://www.qubole.com/tag/hive-query/)

Multi-cluster (https://www.qubole.com/tag/multi-cluster/) Performance (https://www.qubole.com/tag/performance/)

Press (https://www.qubole.com/tag/press-2/) qubole (https://www.qubole.com/tag/qubole/) Reserved Disks (https://www.qubole.com/tag/reserved-disks/)

Social Media (https://www.qubole.com/tag/social-media/) Status Dashboard (https://www.qubole.com/tag/status-dashboard/)

Technology (https://www.qubole.com/tag/technology/) User Interface (https://www.qubole.com/tag/user-interface/)

Contact Us (https://www.qubole.com/contact-us/)

Support (https://qubole.zendesk.com/hc/en-us)

Free Trial (https://api.qubole.com/users/sign_up?
__hstc=1833966.a514a1766ec1829f4d5fb27b4de63bd5.1438716957867.1443463805425.1443676478084.12&__hssc=1833966.38.

About Us (https://www.qubole.com/about-us/)
Press Releases (https://www.qubole.com/press-releases/)
In The News (https://www.qubole.com/in-the-news/)
Career (https://www.qubole.com/career/)

Big Data Use Cases (/resources/solution/best-use-cases-for-big-data-analytics/)
Webinars (https://www.qubole.com/webinars/)
Case Studies (https://www.qubole.com/case-studies/)
Articles (https://www.qubole.com/articles/)

Hadoop Oozie (https://www.qubole.com/cp/hadoop-oozie)
Big Data Analytics For Marketers (https://www.qubole.com/cp/big-data-analytics-for-marketers)
Hadoop Presto (https://www.qubole.com/cp/hadoop-presto)
Hadoop Kafka (https://www.qubole.com/cp/hadoop-kafka)
Hadoop Airflow (https://www.qubole.com/cp/hadoop-airflow)
Hadoop YARN (https://www.qubole.com/cp/hadoop-yarn)

(https://www.facebook.com/qubole) (http://www.linkedin.com/company/qubole) (https://twitter.com/@qubole)