Roadmap

Before adding to the list below, please check JIRA to see if a ticket has already been opened for the feature. If not, please open a ticket on the Hive JIRA and also update the following list.

Features to be added

Major Recent Changes

- Table Statistics
- Archiving
- Indexing First Cut
- Concurrency
- Conversion to Map-Join at Runtime
- Support for Multiple Distincts
- Remove Partition Filtering Conditions
- INSERT INTO statement
- Block-level merge
- HAVING clause support
- Cross-database queries
- Bitmap Index
- Use Filter Pushdown for Automatically Accessing Indexes
- Remove Duplicate Filters
- Authentication
- Authorization

Current Projects

- Bloom Filters
- TIMESTAMP data type

Up For Grabs

Priorities are denoted as P0 > P1 > ...

Query Optimization

- P0 Optimizing JOIN followed by GROUP BY
  - A lot of analytics queries are JOINs followed by GROUP BY (join keys and group by keys may or may not be the same or related). We need a better optimization for this kind of query (optimize number of MapReduce Jobs vs. optimize data transfer size etc.)
- P0 Optimize JOINs using Bloom Filters
  - This is to optimize the case where two big tables are joined but the results are small.
- P1 Column-level statistics
  - We already have UDAFs for percentile, histogram etc. We need to figure out a smart way to compute column-level (approximate) stats in a streaming fashion (during/piggy-backing the scan/loading data process) without firing a new query.
- P1 Determining whether to use skew for a count distinct in group by
  - Need column level stats or a sample task before the real query.
- P1 Exploit clustering information in the input data
  - Some data are generated using 'GROUP BY', 'CLUSTER BY', 'ORDER BY' etc. that implicit ordering in the data. We should exploit this metadata to do better join in choosing joins (e.g., map-side sort-merge join).
- Cost-based Optimization

Query Execution

- P0 Native Support for types of numbers, datetime, and IP addresses
  - Currently most numbers, datetimes, and IP addresses are treated as strings. If we know their data types we should store these data types in a more effective manner (see also "Binary Storage and SerDe").
- P0 Create a URL data type that is more friendly to data compression.
  - URLs are very long and can contribute to large portion of storage. They are treated as generic strings and use string compression algorithms. We should investigate if there are ways to treate URL data types in a clever way so that the compression algorithm (may be a customized compression algorithm) can compress the data better in small enough overhead.
- P1 Binary storage format and SerDe
  - The idea is to store data in their native format (rather than converting to UTF-8 string type) before stored on disk. This can potentially save a lot of CPU/IO cost in data conversion and object creations.
  - Also investigating the compression technics used in other column-stores that does not require decompression before query (filtering).
  - Revisit the LazyBinarySerDe and see if they can be reused or extended to this storage format.
  - Make this storage format amenable to mmap() (or FileChannel in Java) so that the system can skip I/O if memory random access can skip part of data (e.g., columns)
- Support for IN, exists and correlated subqueries
- More native types - Enums, timestamp
- Persistent UDF's
- SQL/OLAP
- Storage handler improvements

- System views
- JDBC/ODBC improvements
- ~~mapred to mapreduce transition~~ (no longer needed since mapred got undeprecated)

Metadata Management

- P0 Reducing the size of the metastore ‚Äì whatever it takes: some ideas were to not store COLUMNS etc.
    - One idea here is to introduce something like a schema object that contains the column objects. Partitions would inherit the schema object, which would only change when the table schema changes. Also work would be involved to migrate the existing setup. This would greatly reduce the number of rows in columns (current in the billions).
- P1 View improvements

Test, Error Messages and Debugging

- P0 Heavy-duty test infrastructure
- Automated code coverage reports
- Hive CLI improvement/Error messages
- HiveServer robustness
- Debuggability / Resumability:
    - Show users the last portion of the data that caused the task to fail
    - Restart a job with a particular mapper (that failed earlier, for debugging purposes)
    - Resume at map-reduce job level.