Agenda

- Apache Hive

Apache Hive

- https://cwiki.apache.org/confluence/display/Hive/LanguageManual

Hive Joins

- Hive joins are similar to RDBMS joins
    - Map Side Joins
    - Reduce Side Joins
    - LEFT SEMI JOIN *
    - Implicit Join syntax
- Salient points to consider when writing join queries:
    - More than 2 tables can be joined in the same query.

    ```
    SELECT e.ename, m.topic FROM emp_staging e
    INNER JOIN emp_staging_meetings em ON e.empno = em.empno
    INNER JOIN meetings m ON m.meetingno = em.meetingno;
    ```

    - Hive converts joins over multiple tables into a single map/reduce job if for every table the same column is used in the join clauses.

    ```
    SELECT c.cname, o.amt, s.sname FROM salespeople s
    INNER JOIN orders o ON s.snum = o.snum
    INNER JOIN customers c ON s.snum = c.snum;
    ```

- Joins occur before WHERE clauses. So, if you want to restrict the OUTPUT of a join, a requirement (condition) should be in the WHERE clause, otherwise it should be in the JOIN clause. Following queries produces different results.

```
SELECT d.dname, e.ename FROM dept_staging d
LEFT JOIN emp_staging e ON e.deptno = d.deptno
WHERE d.deptno = 40;
```

```
SELECT d.dname, e.ename FROM dept_staging d
LEFT JOIN emp_staging e ON e.deptno = d.deptno AND d.deptno = 40;
```

- LEFT SEMI JOIN implements the uncorrelated IN/EXISTS subquery semantics in an efficient way. As of Hive 0.13 the IN/NOT IN/EXISTS/NOT EXISTS operators are supported using subqueries so most of these JOINs don't have to be performed manually anymore. The restrictions of using LEFT SEMI JOIN are that the right-hand-side table should only be referenced in the join condition (ON-clause), but not in WHERE- or SELECT-clauses etc.

```
SELECT d.deptno, d.dname FROM dept_staging d
WHERE EXISTS (SELECT e.deptno FROM emp_staging e WHERE e.deptno = d.deptno);
```

```
SELECT d.deptno, d.dname FROM dept_staging d
WHERE d.deptno IN (SELECT e.deptno FROM emp_staging e);
```

```
SELECT d.deptno, d.dname FROM dept_staging d
LEFT SEMI JOIN emp_staging e ON (d.deptno = e.deptno);
```

- If one of the tables being joined are small, the join can be performed as a map only job. Example below, doesn't need reducer. For each mapper of emp_staging, dept_staging table is read completely. The restriction is that a FULL/RIGHT OUTER JOIN d (small table) cannot be performed.

```
SET hive.auto.convert.join = true;

SELECT /*+ MAPJOIN(d) */ e.ename, d.dname FROM emp_staging e
INNER JOIN dept_staging d ON e.deptno = d.deptno;
```

- The configuration variable hive.auto.convert.join (if set to true) automatically converts the joins to mapjoins at runtime, if possible, and it should be used instead of the mapjoin hint.

```
SET hive.auto.convert.join = true;
SET hive.auto.convert.join.noconditionaltask = true;
SET hive.auto.convert.join.noconditionaltask.size = 10000000; -- small table size to fit in memory
```

- If the tables being joined are bucketized on the join columns, and the number of buckets in one table is a multiple of the number of buckets in the other table, the buckets can be joined with each other. If table emp_staging has 4 buckets and table dept_staging has 4 buckets, the following join can be done on the mapper only. Instead of fetching dept_staging completely for each mapper of emp_staging, only the required buckets are fetched. For the query above, the mapper processing bucket 1 for emp_staging will only fetch bucket 1 of dept_staging.

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

SELECT /*+ MAPJOIN(d) */ e.ename, d.dname FROM emp_staging e
INNER JOIN dept_staging d ON e.deptno = d.deptno;
```

- In every map/reduce stage of the join, the last table in the sequence is streamed through the reducers where as the others are buffered. Therefore, it helps to reduce the memory needed in the reducer for buffering the rows for a particular value of the join key by organizing the tables such that the largest tables appear last in the sequence. In example below, there is single map-reduce stage and the last table customers is streamed while earlier are buffered i.e. salespeople and orders table data will get into the memory of reducer and for each row of customers table join is computed with in-memory tables (orders and salespeople).

```
SELECT c.cname, o.amt, s.sname FROM salespeople s
INNER JOIN orders o ON s.snum = o.snum
INNER JOIN customers c ON s.snum = c.snum;
```

- In every map/reduce stage of the join, the table to be streamed can be specified via a hint. By default, last table is streamed. In example below, orders and customers tables will be buffered in reducer; while salespeople table will be streamed.

```
SELECT /*+ STREAMTABLE(s) */ c.cname, o.amt, s.sname FROM salespeople s
INNER JOIN orders o ON s.snum = o.snum
INNER JOIN customers c ON s.snum = c.snum;
```

Hive Indexes

- Supported in Hive 2.x. Not supported in Hive 3.x.
- Similar to RDBMS index.
- To speed up SELECT queries (searching & grouping).
- Indexes internally store addresses of records for given column values.
- Creating index is time-taking job (for huge data). If indexing is done under load, then clients query performance is too low.
- In Hive indexes are created, but deferred for build (using ALTER statement).
- CREATE INDEX query doesn't create index, rather keep ready for building later.

```
CREATE INDEX idx_deptno ON TABLE emp_staging(deptno) AS 'COMPACT' WITH DEFERRED REBUILD;

SHOW INDEX ON emp_staging;
```

- Index building should be triggered explicitly, when server is less loaded.

```
ALTER INDEX idx_deptno ON emp_staging REBUILD;
```

- In hive indexes are stored in HDFS (as hive tables).
- These indexes are build by different index handlers e.g. BITMAP, CompactHandler, ...
    - Compact: Stores combination of indexed column value & its HDFS block id.
    - Bitmap: Stores combination of indexed column value & list of rows as bitmap. Bitmap indexes work faster than Compact.