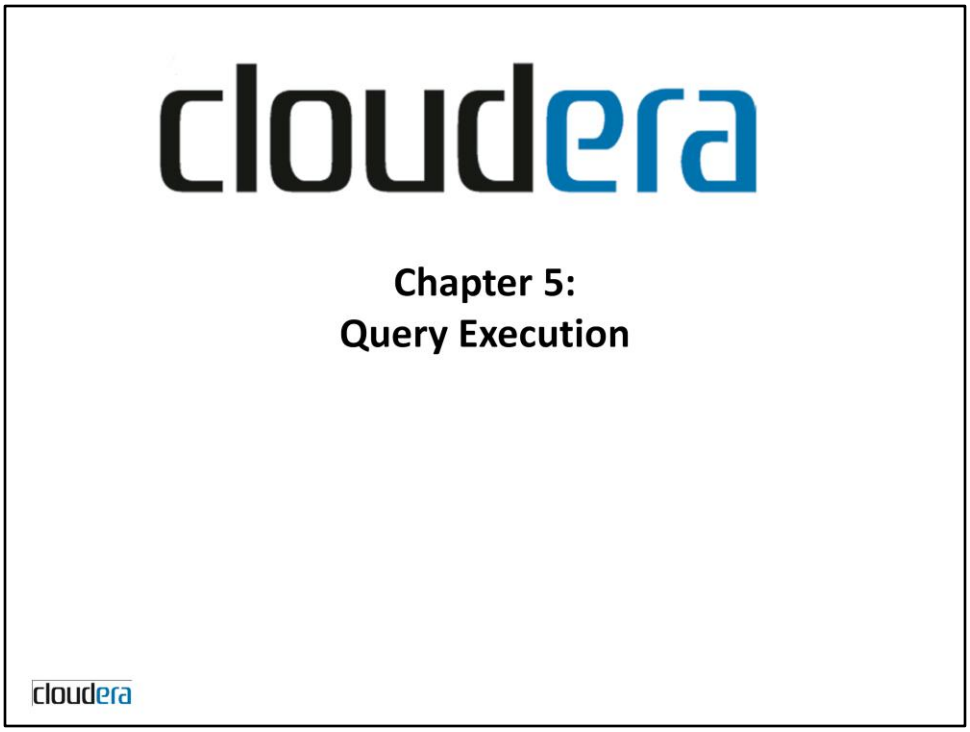
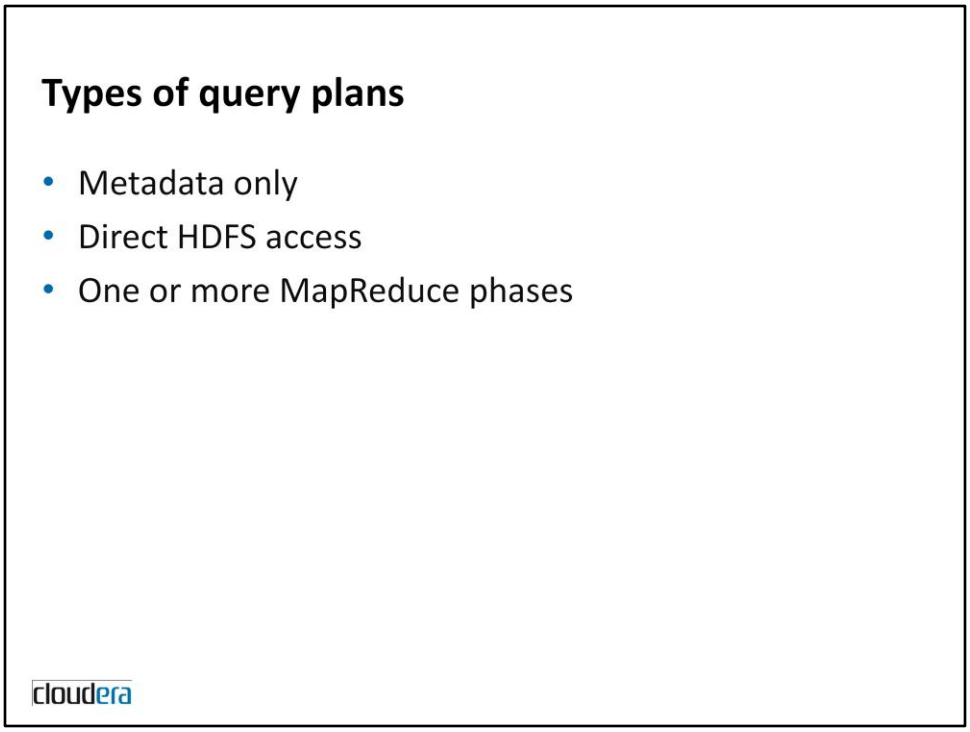


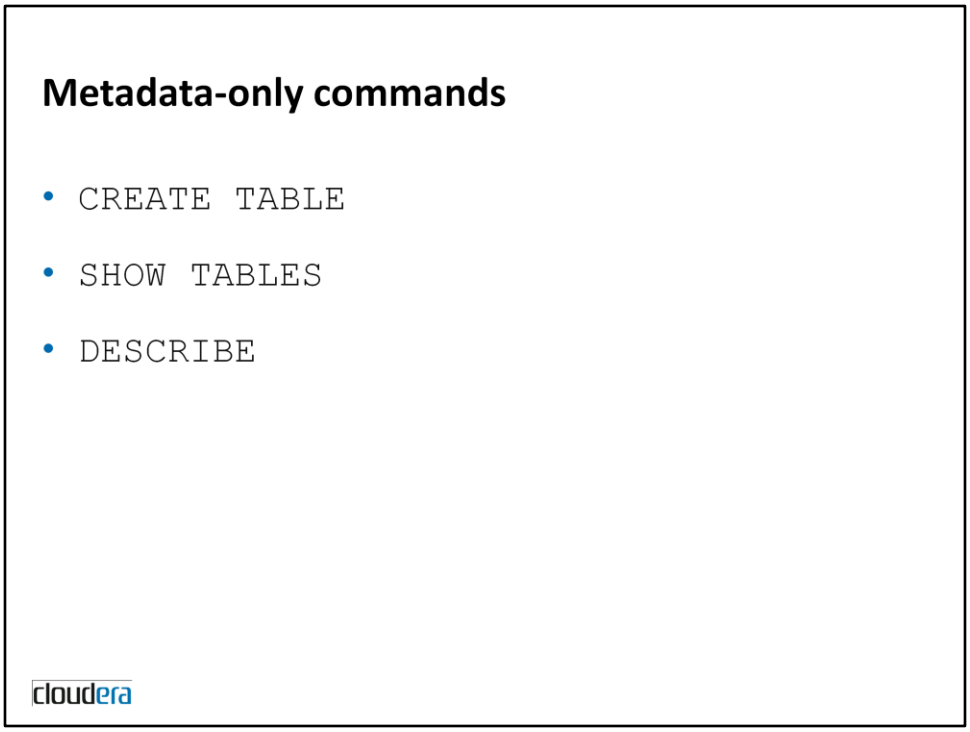
In this chapter, you will learn

- The various types of query plans
- How to use EXPLAIN
- How Hive executes joins
- How to use hints

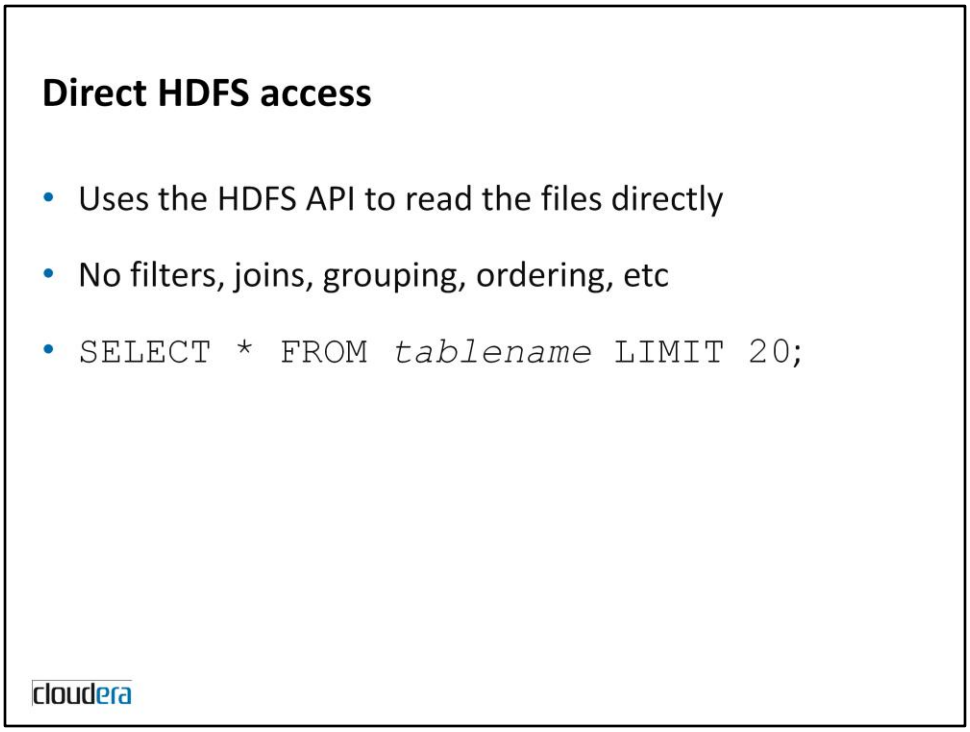
cloudera



There are three possible ways Hive could execute a query. For commands that only need metadata (e.g., `SHOW TABLES`), Hive simply needs to access the metastore. Some statements require data, but can get that data directly from the Hadoop Distributed File System (HDFS). The majority of queries will fall into the third category: one or more MapReduce phases will be executed to perform the filters, joins, grouping, ordering, etc.

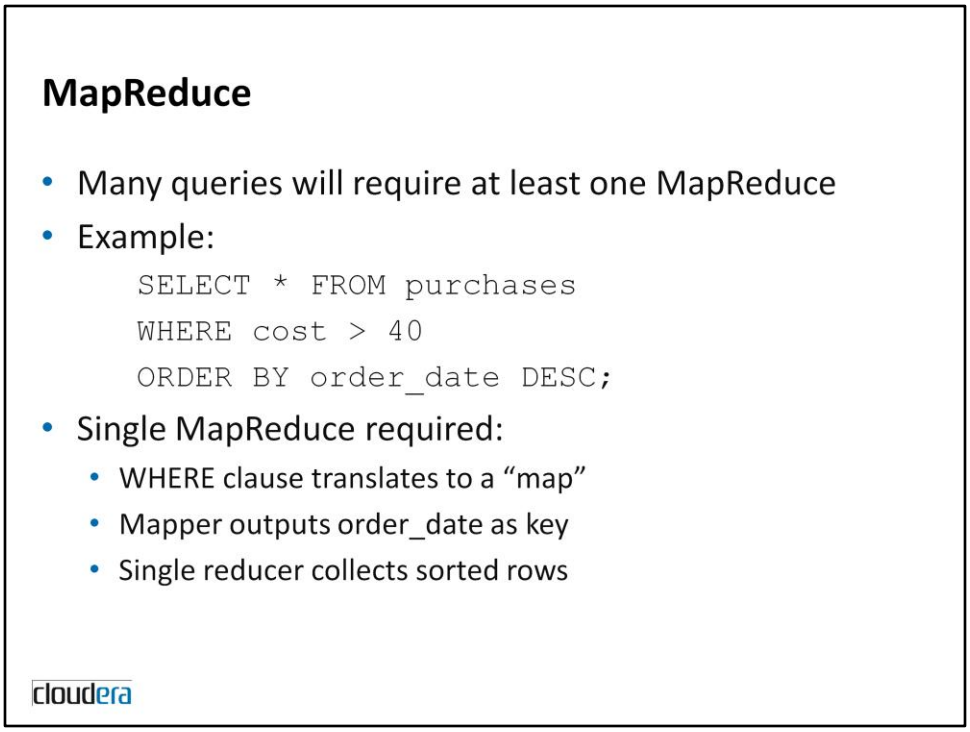


Some commands only use metadata (from the metastore). For example, `CREATE TABLE` needs to write to the metastore. Listing all the tables using `SHOW TABLES` does not require accessing the data in the tables, but merely getting a list of tables from the metastore. Likewise, describing the structure of a table only requires information about the table definition. To execute these commands, Hive will query the Hive metastore and return the results. These commands are typical very fast.



Hive stores data in HDFS. Like a regular file system, Hive can perform a “cat” operation on files in HDFS using the HDFS API. For example, "SELECT * FROM *tablename* LIMIT 20" is very similar to

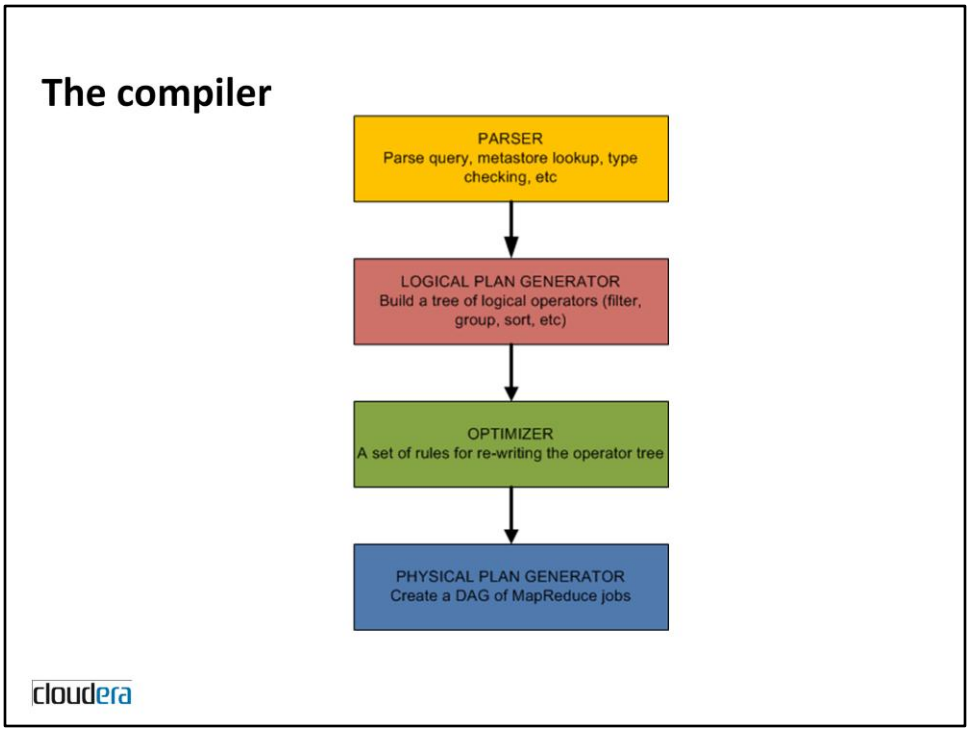
```
hadoop fs -ls /user/hive/warehouse/tablename/file/ | head
-n 20
```



Many queries will require Hive to compile and execute at least one MapReduce job. For example, take this query:

```
SELECT * FROM purchases
WHERE cost > 40
ORDER BY order_date DESC;
```

The "`WHERE`" clause will be executed as a map. The benefit of MapReduce is that scanning a table is done in parallel, where each map task reads a small chunk of data (usually 64 or 128MB). The map will output the matching rows (`cost>40`) with the `order_date` as the key and the rest of the data as the values. This allows the MapReduce framework to sort the data by `order_date`. Hive will set the number of reducers to one to guarantee a complete ordering.



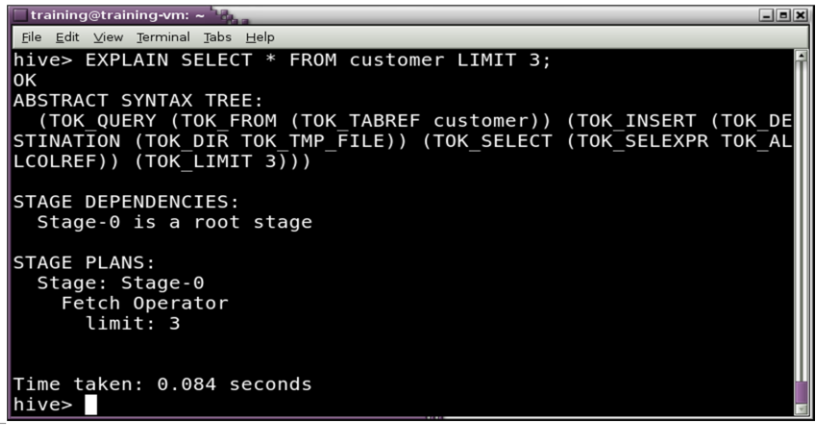
When a query is submitted to the Hive shell, several things happen before communicating to the Hadoop cluster.

1. The Parser and Semantic Analyzer convert the query to an internal representation. A metastore lookup is performed to verify the column names, expand "SELECT *", and do type-checking.
2. The Logical Plan Generator takes the internal representation and builds a tree of logical operators. For each query (include subqueries), the resulting tree will have this form:
   FROM->WHERE->GROUPBY->ORDERBY->SELECT
3. The Optimizer runs a set of rules that re-write aspects of the logical plan. An example rule is whether tables in a join should be re-ordered or if the join can be performed in a map instead of reduce (a "map-side" join).
4. The Physical Plan Generator creates a directed acyclic graph (DAG) of any map-reduce phases that are required.

Finally, this workflow is submitted to the Hadoop cluster and executed.

EXPLAIN

- EXPLAIN SELECT…
- Displays the syntax tree and DAG in textual form



cloudera

The `EXPLAIN` keyword can be used before any `SELECT` statement. This invokes the parser, logical plan generator, optimizer and physical plan generator (but does not execute the query). Example:

```
hive> EXPLAIN SELECT * FROM customer LIMIT 3;
```

As expected, this query does not require any MapReduce jobs. Instead, Hive invokes a "Fetch Operator" which uses the HDFS API to read the data for this table.

EXPLAIN – the map

- EXPLAIN SELECT * FROM purchases
  WHERE cost > 40
  ORDER BY order_date DESC;
- …
  STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> **Map** Operator Tree:
        purchases
          TableScan
            alias: purchases
            Filter Operator
              predicate:
                  expr: **(cost > UDFToDouble(40))**
                  type: boolean

cloudera

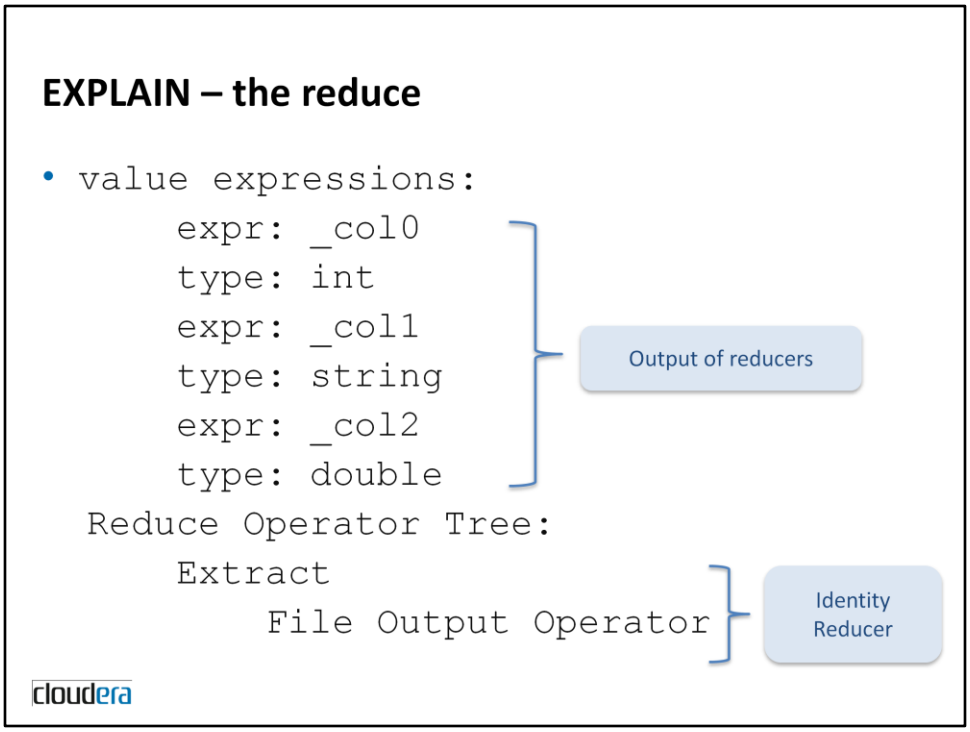
When a MapReduce job is required, the `EXPLAIN` output shows what is being done in the map and reduce (if a reduce task is required). In this example, the map is scanning the purchases table and filtering for cost > 40.

EXPLAIN – the shuffle and sort

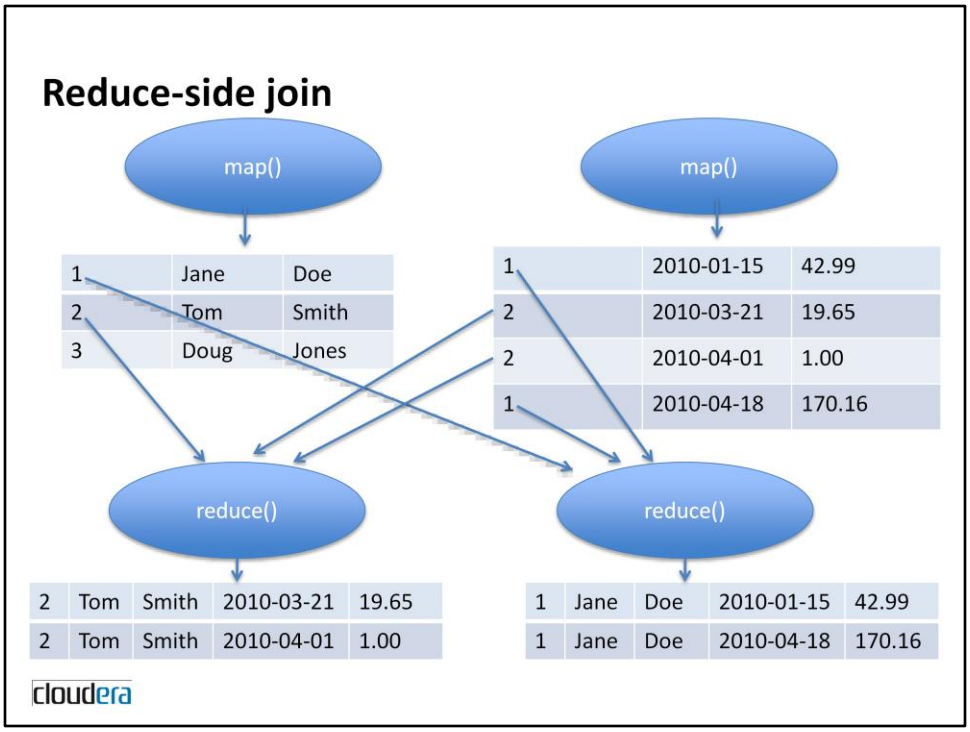
```
• Select Operator
        expressions:
              expr: custid
              type: int
              expr: order_date
              type: string
              expr: cost
              type: double
         outputColumnNames: _col0, _col1, _col2
         Reduce Output Operator
           key expressions:
             expr: _col1
             type: string
```

cloudera

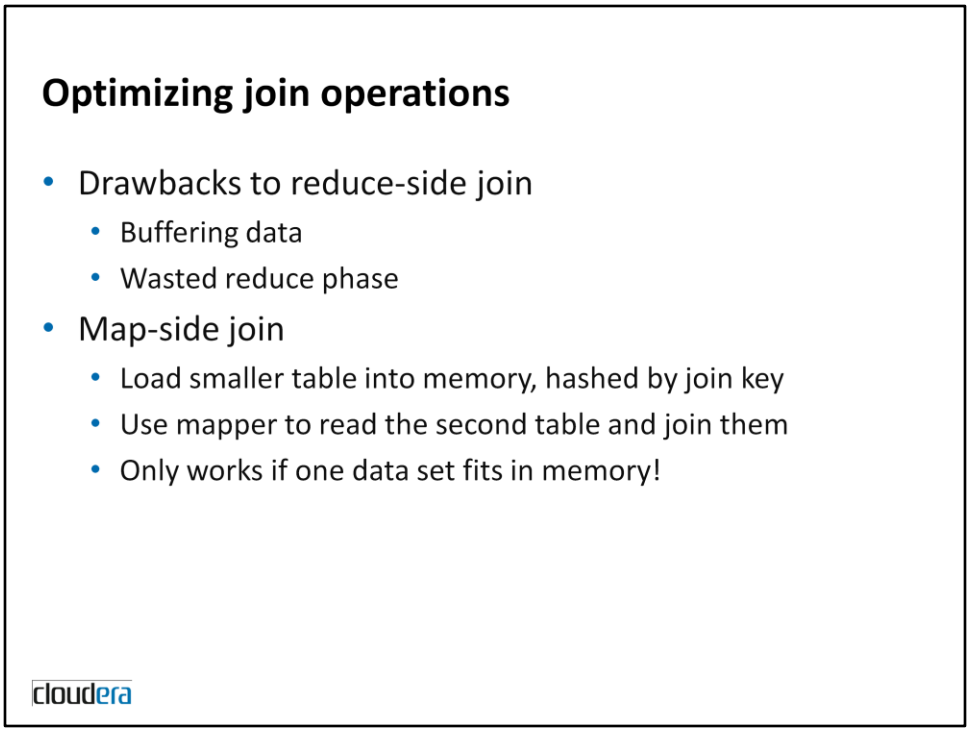
The MapReduce framework guarantees that the output of the map tasks will be sorted by key before being consumed by a reducer. The process is called the "shuffle and sort". Hive takes advantage of this by having the mappers output "order_date" as the key and setting the number of reducers to one. This accomplishes the `ORDER BY` clause of the query.



The work for filtering and sorting the result set has already been done by the map task and shuffle/sort process. Therefore the reducer just needs to output the results. This is called the IdentityReducer.



A reduce-side join is the normal technique used for joining multiple tables. The map tasks read the data from all tables and output the join key (primary key or foreign key) as the key for the reducers. The shuffle and sort groups the output by keys and sends the intermediate rows to reducers. Then the reducers joins the rows.



There are drawbacks to the reduce-side join, such as having to buffer data in the reducers and not being able to sort by a different key. For example, if a query had this pattern:

```
SELECT..FROM t1 JOIN t2 ON(t1.foo = t2.foo) ORDER BY
t1.bar
```

The above query would need 2 MapReduce jobs; the first would do the join and the second would sort.

A better technique could be a map-side join. A map-side join pre-loads the smaller table (or tables) into a memory structure such as a HashMap. Then the map tasks read the larger table, doing a lookup against the HashMap for each row. A reduce phase is not required for the join operation.

Using map-side join in Hive

- Reduce-side join is default
- Use a hint
  - SELECT **/*+ MAPJOIN(t2) */** t1.col, t2.col

    FROM t1 JOIN t2

    ON (t1.col = t2.col)

cloudera

The default join execution is a reduce-side join. A “hint” can be added to a query to request a map-side join. This should only be used if the table is small enough to fit in memory on each mapper.

Conclusion

In this chapter, you have learned:

- The various types of query plans
- How to use EXPLAIN
- How Hive executes joins
- How to use hints

cloudera