Big Data Technologies

Agenda

- Hive

ORC format

- Optimized Row Columnar format
- Designed by Hive team for efficient execution of hive queries
- Hadoop has ORCInputFormat and ORCOutputFormat.
- Hive queries when work on ORC tables, they internally use ORCInputFormat to read data into mapper and use ORCOutputFormat to write reducer output data into HDFS.

Vectorization

- To perform math operations on primitive types, hive uses vectorization.
- Typically Hive data is stored in ORC format, that is further divided into the blocks and arranged in columns.
- This helps speeding up math operations on primitive types (block by block).

  ```
  set hive.vectorized.execution.enabled=true;
  ```

Hive Architecture

- Refer slides

Hive Compaction

- Minor compaction: Smaller changes files (delta) are mereged together to reduce the number of delta files. This will save space in HDFS and also speed-up further queries. In minor compaction, data in main file is not modified (e.g. older records are still present there). Minor compaction process takes relatively less time.

- Major compaction: All delta files and main data files are mereged to created a new data file. All older versions (delta files and older main file) are deleted. This will significantly improve speed to further queries. Major compaction needs more time.

```
ALTER TABLE ncdc_orc COMPACT 'major';
```

Hive scripts

- Set of hive commands.
- Example 1: Compaction process

  ```
  -- compact.hql
  ALTER TABLE ncdc_orc COMPACT 'major';
  ```

- Example 2: Rebuild correlation table

  ```
  -- rebuild-correlation.hql
  SET mapreduce.reduce.memory.mb = 5120;
  SET mapreduce.reduce.java.opts = -Xmx4096m;

  INSERT INTO movies_orc
  SELECT * FROM movies_staging WHERE condition_to_get_new_records_only;

  INSERT INTO ratings_orc
  SELECT * FROM ratings_staging WHERE condition_to_get_new_records_only;

  ALTER MATERIALIZED VIEW mv_user_movies REBUILD;

  DROP TABLE IF EXISTS movies_corr;
  ```

```
CREATE TABLE movies_corr AS
SELECT m1, m2, COUNT(rt1) cnt, CORR(rt1,rt2) cor
FROM mv_user_movies
GROUP BY m1, m2;
```

- To run the script:
  - beeline> !run /path/of/script.hql
  - OR
  - terminal> beeline -u jdbc:hive2://localhost:10000/dbda -n $USER -f /path/of/script.hql

Managed Table vs External Table

- Managed Table
  - CREATE TABLE tablename ...
  - Created in HDFS "warehouse" directory and metadata is stored in metastore.
  - To load the data in the table -- LOAD DATA or INSERT.
  - DROP TABLE tablename ... -- drop the data from HDFS and metadata from metastore.
- External Table
  - CREATE EXTERNAL TABLE tablename ... LOCATION '/path/of/data/directory';
  - Data is already present in HDFS (not necessarily in "warehouse") and metadata is stored in metastore.
  - The data is already present -- LOCATION '...' -- given file table creation. To load the additional data in the table -- LOAD DATA or INSERT.
  - DROP TABLE tablename ... -- drop metadata from metastore. Data from HDFS is not deleted.

```
CREATE EXTERNAL TABLE emp_staging(
empno INT,
ename STRING,
job STRING,
mgr INT,
hire STRING,
sal DOUBLE,
comm DOUBLE,
deptno INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/nilesh/emp/input';

SELECT * FROM emp_staging
LIMIT 10;
```

Hive Functions

- Hive function types
  - Single row funcions/Scalar functions
    - "n" rows --> Function --> "n" rows
  - Multi row functions/Group functions/Aggregated functions
    - "n" rows --> Function --> "m" rows (m < n)
  - Table valued functions
    - "n" rows --> Function --> "m" rows (m > n)
- Hive user-defined functions
  - UDF -- User Defined Function
  - UDAF -- User Defined Aggregate Function
  - UDTF -- User Defined Table Function
  - Implemented in Java.

```
DESCRIBE movies_orc;

SELECT * FROM  movies_orc LIMIT 5;

SELECT id, title, SPLIT(genres, '\\|') FROM movies_orc
LIMIT 5;

CREATE TABLE movies(id INT, title STRING, genres ARRAY<STRING>)
```

```
STORED AS ORC;

INSERT INTO movies
SELECT id, title, SPLIT(genres, '\\|') FROM movies_orc;

SELECT * FROM movies
LIMIT 5;

SELECT COUNT(id) FROM movies
WHERE ARRAY_CONTAINS(genres, 'Romance');
```

- If input movie:

```
| 4          | Waiting to Exhale (1995)            | ["Comedy","Drama","Romance"]
```

- Output records

```
| 4          | Waiting to Exhale (1995)            | "Comedy"
| 4          | Waiting to Exhale (1995)            | "Drama"
| 4          | Waiting to Exhale (1995)            | "Romance"
```

```
SELECT id, title, EXPLODE(genres) FROM movies;
-- works in spark (not in hive)

SELECT id, title, genre FROM movies
LATERAL VIEW EXPLODE(genres) v_genres AS genre
LIMIT 20;

SELECT genre, COUNT(id) FROM movies
```

```
LATERAL VIEW EXPLODE(genres) v_genres AS genre
GROUP BY genre;
```

```
SELECT * FROM ratings_orc
LIMIT 5;

SELECT userid,movieid,rating,FROM_UNIXTIME(rtime) FROM ratings_orc
LIMIT 5;

CREATE TABLE ratings(
userid INT,
movieid INT,
rating DOUBLE,
rtime TIMESTAMP
)
STORED AS ORC;

INSERT INTO ratings
SELECT userid,movieid,rating,FROM_UNIXTIME(rtime) FROM ratings_orc;

SELECT * FROM ratings
LIMIT 5;

SELECT YEAR(rtime) yr, COUNT(rating) FROM ratings
GROUP BY YEAR(rtime);
```

Partitioning

Static partitioning

```
CREATE TABLE emp_part_dept(
empno INT,
ename STRING,
job STRING,
mgr INT,
hire STRING,
sal DOUBLE,
comm DOUBLE
)
PARTITIONED BY (deptno INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

DESCRIBE emp_part_dept;

LOAD DATA LOCAL
INPATH '/home/nilesh/sep22/dbda/bigdata/data/emp10.csv'
INTO TABLE emp_part_dept
PARTITION (deptno=10);

LOAD DATA LOCAL
INPATH '/home/nilesh/sep22/dbda/bigdata/data/emp20.csv'
INTO TABLE emp_part_dept
PARTITION (deptno=20);

LOAD DATA LOCAL
INPATH '/home/nilesh/sep22/dbda/bigdata/data/emp30.csv'
INTO TABLE emp_part_dept
PARTITION (deptno=30);

EXPLAIN
SELECT SUM(sal) FROM emp_staging
WHERE deptno=20;
```

```
EXPLAIN
SELECT SUM(sal) FROM emp_part_dept
WHERE deptno=20;

DROP TABLE emp_part_dept;
```

Dynamic partitioning

```
CREATE TABLE emp_part_dept(
empno INT,
ename STRING,
job STRING,
mgr INT,
hire STRING,
sal DOUBLE,
comm DOUBLE
)
PARTITIONED BY (deptno INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

DESCRIBE emp_part_dept;

-- load emp.csv into a staging table -- emp_staging

INSERT INTO emp_part_dept PARTITION(deptno)
SELECT empno,ename,job,mgr,hire,sal,comm,deptno FROM emp_staging;
```

```
CREATE TABLE emp_part_dept_job(
empno INT,
ename STRING,
mgr INT,
hire STRING,
sal DOUBLE,
comm DOUBLE
)
PARTITIONED BY (deptno INT, job STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

DESCRIBE emp_part_dept_job;

INSERT INTO emp_part_dept_job PARTITION(deptno,job)
SELECT empno,ename,mgr,hire,sal,comm,deptno,job FROM emp_staging;
```

Assignment

- External tables
  - Upload contacts.csv into HDFS.
  - Table contacts1: `id INT, name STRING, emails STRING, addr STRING, phone STRING`
    - SELECT * FROM contacts1 WHERE emails LIKE '%nilesh@sunbeaminfo.com%';
  - Table contacts2: `id INT, name STRING, emails ARRAY<STRING>, addr STRUCT<...>, phone MAP<STRING,STRING>`
    - SELECT * FROM contacts1 WHERE ARRAY_CONATINS(emails, 'nilesh@sunbeaminfo.com');
  - Table contacts3: `line STRING`
  - DROP TABLE contacts1;
- EXPLODE()
  - Load Hadoop license file into a Managed table `line STRING`.
  - Count number of occurrences for each word.
  - Count number of occurrences for each word other than stop words -- WHERE word NOT IN ('a', 'an', 'the', 'is', 'you', 'i', ...).
  - Find top 20 most frequent words.
- Dynamic Partitioning
  - Create emp table partitioned by job.

  - Create another emp table partitioned by job and deptno.