Chapter17. HIVE

17.1 하이브 아키텍처

- 하이브
    - 하둡에 저장된 데이터를 쉽게 처리할 수 있는 데이터웨어하우스 패키지.

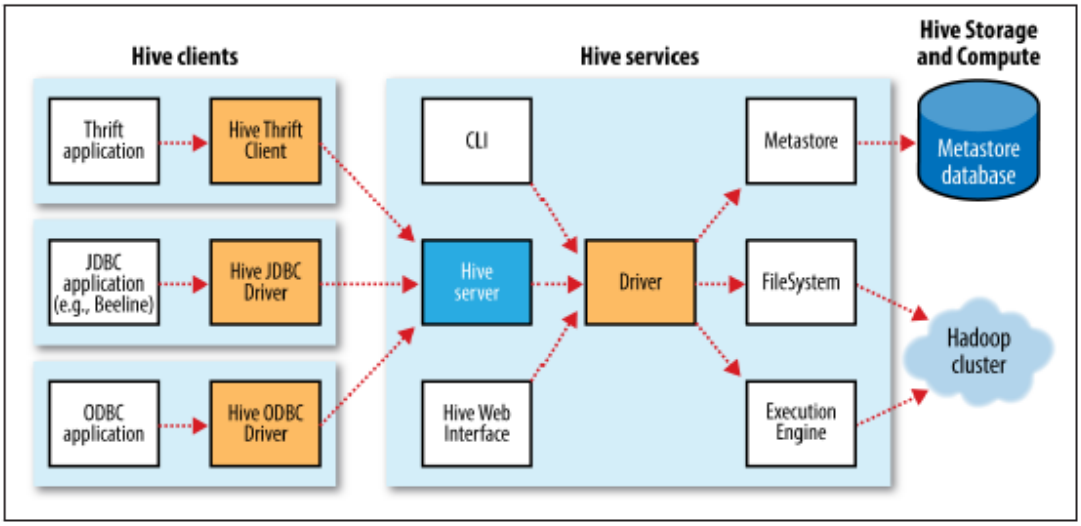


Figure 17-1. Hive architecture

- 하이브 클라이언트
  - Command Line Interface(CLI), 하이브 서버, 웹 인터페이스로 구성됨.
  - 하이브 서버
    - JDBC, ODBC, Thrift로 개발된 클라이언트가 하이브 서비스를 이용할 수 있게 thrift 서비스를 제공
- 메타스토어(Metastore)
  - 하둡에서 처리된 메타데이터의 구조를 메타스토어에 저장.
  - 스터디/기능 테스트의 경우 경량 DB인 아파치 더비를 로컬 클라이언트에 설치해 메타스토어를 구성.
  - 여러 명의 사용자가 동일한 하둡 클러스터에서 하이브로 작업할 경우에는 RDBMS로 메타스토어를 구성해야 함.
    - 각 사용자의 로컬에 메타스토어가 생성된다면 메타데이터가 전혀 관리되지 않을 것이기 때문.
  - 오라클, MySQL 등 JDBC를 지원하는 모든 데이터베이스를 이용해 메타스토어를 구축할 수 있음
- 드라이버
  - 사용자가 입력한 하이브QL문을 해석. 하둡과 연결되어 하이브QL문을 실행하고, 하이브QL은 하이브QL문의 실행 계획을 작성하고, 최적화 작업까지 함께 수행

17.2 하이브QL

- 하이브는 하이브QL이라는 SQL문과 매우 유사한 언어를 제공. 대부분의 기능은 SQL과 유사하지만 다소 차이점이 있음
- 하이브QL은 대소문자를 구분하지 않음
- SQL과 하이브QL의 차이점

1. 'UPDATE', 'DELETE', 'INSERT' 에서의 차이점
   - 하이브에서 사용하는 데이터가 HDFS에 저장되는데, HDFS가 한 번 저장한 파일은 수정할 수 없기 때문에 UPDATE와 DELETE를 사용할 수 없음.
   - 같은 이유로 INSERT도 비어 있는 테이블에 입력하거나, 이미 입력된 데이터를 덮어 쓰는 경우에만 가능. 따라서 하이브QL은 "INSERT OVERWRITE"라는 키워드를 사용

2. 'FROM' 에서의 차이점
    - SQL은 어떠한 절에서도 서브쿼리를 사용할 수 있지만 하이브QL은 FROM 절에서만 서브 쿼리를 사용할 수 있음
3. '뷰' 에서의 차이점
    - SQL의 뷰는 업데이트할 수 있고, 구체화된 뷰 또는 비구체화된 뷰를 지원.
    - 하이브QL의 뷰는 읽기 전용이며, 비구체화된 뷰만 지원
4. 'SELECT' 에서의 차이점
    - SELECT 문을 사용할 때 HAVING 절을 사용할 수 없음
5. '저장 프로시저(stored procedure)' 에서의 차이점
    - 저장 프로시저(stored procedure)를 지원하지 않음. 대신 맵리듀스 스크립트를 실행할 수 있음

서브쿼리란?
- 쿼리문 안에 쿼리문을 사용하는 것
- 서브쿼리는 괄호로 감싸서 사용
- 서브쿼리의 칼럼을 메인쿼리에서 참조할 수 없음.
- 메인쿼리의 칼럼은 서브쿼리에서 참조할 수 있음.

17.2.1 테이블 생성

- 하둡은 HDFS에 저장된 파일에 직접 접근해서 처리하지만 하이브는 메타스토어에 저장된 테이블을 분석
- iris 테이블 생성

```
CREATE TABLE iris(Id Int, SepalLength FLOAT, SepalWidth FLOAT, PetalLength FLOAT, PetalWidth FLOAT)
COMMENT 'It is data about iris (mad flower) data. A piece of data sorted into 150 star records.'
PARTITIONED BY (Species STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

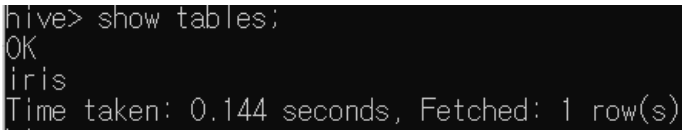
하이브 칼럼 타입

Aa 타입	내용	열
TINYINT	1바이트 정수	
SMALLINT	2바이트 정수	
INT	4바이트 정수	
BIGINT	8바이트 정수	
BOOLEAN	TRUE/FALSE	
FLOAT	단정밀도 부동 소수점	
DOUBLE	배정밀도 부동 소수점	
STRING	문자열	

- CREATE TABLE 테이블(칼럼명 칼럼 타입, ...)
    - 테이블 생성의 주 구문.
    - 이후 구문은 테이블에 대한 부가적인 정보를 설정하는 부분

- COMMENT '~~~'
  - 테이블의 설명을 참고용으로 등록하는 부분
- PARTITIONED BY ()
  - 파티션을 설정
  - 하이브는 쿼리문의 수행 속도를 향상시키기 위해 파티션을 설정할 수 있음. 파티션을 설정하면 해당 테이블의 데이터를 파티션별로 디렉터리를 생성해서 저장.
  - 파티션키는 해당 테이블의 새로운 칼럼으로 추가됨
- ROW FORMAT, FIELS TERMINATED BY, LINES TERMINATED BY
  - 해당 테이블 내의 데이터가 어떠한 형식으로 저장되는지 설정
  - 여기에서는 필드를 콤마 기준으로 구분하고, 행과 행은 \n값으로 구분
- STORED AS
  - 데이터 저장 파일 포맷을 의미
  - 하이브는 텍스트 파일을 위한 TEXTFILE과 시퀀스파일을 저장하기 위한 SEQUENCEFILE을 지원
- 생성한 데이터 확인

```
hive > SHOW TABLES;
```



- 외부 테이블 생성 방식
  - 외부테이블은 hive.metastore.warehouse.dir 속성이 가리키는 디렉터리에 데이터를 저장하지 않고, 테이블 생성 시 설정한 경로로 데이터를 저장.
  - 사용자가 실수로 테이블을 DROP했더라도 데이터가 보존된다는 장점
  - 기존 코드와의 차이점
    - CREATE TABLE → CREATE EXTERNAL TABLE
    - 마지막줄에 LOCATION + '데이터를 저장할 경로'; 추가

17.2.2 데이터 수정

- 생성된 테이블은 ALTER TABLE을 이용해 수정할 수 있음.
- 테이블 이름 변경
  - ALTER TABLE의 RENAME 옵션

```
ALTER TABLE iris RENAME TO iris_150;
```

- 기존 테이블의 칼럼을 추가
  - ALTER TABLE의 ADD COLUMNS 옵션

```
ALTER TABLE iris ADD COLUMNS (predicted STRING)
```

- 테이블을 삭제
  - DROP TABLE
  - 메타스토어 데이터베이스에 저장된 테이블과 HDFS에 저장된 데이터가 모두 삭제됨.
  - EXTERNAL 키워드를 통해 외부 테이블을 생성했다면 데이터는 남아 있고 메타데이터만 삭제됨

```
DROP TABLE iris;
```

17.2.3 데이터 업로드

17.2.3.1 window 데이터 ubuntu server 로 전송

- window의 데이터를 ubuntu server로 파일 전송
- 윈도우 명령 프롬포트에서 실행
- scp C:윈도우/파일/경로 우분투계정@ip 주소: /home/우분투계정
  - ex) scp C:/Users/이혜림/Downloads/iris.csv hduser@127.0.0.1:/usr/local/hadoop
- 만약 Permission denied (public) 에러가 발생하면 sshd_config 파일 수정 이후 재실행

  >> sudo vi /etc/ssh/sshd_config

```
#Port 22 -> Port 22
PermitRootLogin Yes
PasswordAuthentication Yes
PubkeyAuthentication No
```

```
sudo service ssh reshart
```

17.2.3.2 데이터 업로드

- 앞서 생성한 iris 테이블에 데이터를 업로드
- 하이브는 로컬 파일 시스템에 있는 데이터와 HDFS에 저장된 데이터를 모두 업로드 할 수 있음

```
LOAD DATA LOCAL INPATH '/usr/local/hadoop/iris1.csv'
OVERWRITE INTO TABLE iris
PARTITION (Species='setosa');

LOAD DATA LOCAL INPATH '/usr/local/hadoop/iris2.csv'
OVERWRITE INTO TABLE iris
PARTITION (Species='versicolor');

LOAD DATA LOCAL INPATH '/usr/local/hadoop/iris3.csv'
OVERWRITE INTO TABLE iris
PARTITION (Species='virginica');
```

- OVERWRITE INFO
  - 중복된 레코드가 있어도 무시하고 입력
- PARTITION

- 테이블에는 파티션을 설정했는데, 테이블을 등록할 때 PARTITION을 등록하지 않으면 LOAD DATA 실행시 오류 발생
- 이때 PARTITION (칼럼='해당 칼럼 값') 에서 전체 데이터를 해당 칼럼 값에 따라 나눠서 넣어주는 것이 아닌 해당 파티션의 칼럼값이 '해당 칼럼 값'이다 라고 이름을 붙여주는 것.
- 따라서 전체 데이터를 해당 칼럼 값에 따라 나누어 파티션하고 싶다면 전체 데이터를 질의해서 해주어야 함.
- 해당 방법에선 Species에 따른 각각의 데이터가 존재해야 함

```
Found 3 items
drwxrwxr-x   - hduser supergroup          0 2021-07-14 14:15 /user/hive/warehouse/iris/species=setosa
drwxrwxr-x   - hduser supergroup          0 2021-07-14 14:18 /user/hive/warehouse/iris/species=versicolor
drwxrwxr-x   - hduser supergroup          0 2021-07-14 14:18 /user/hive/warehouse/iris/species=virginica
```

- 헤더 출력 설정

```
ET hive.cli.print.header = TRUE;
```

- 상위 10개 중 출력
  - 원래 하이브는 질의를 실행하면 맵리듀스 잡을 실행. 그러나 최근 버전의 하이브는 하나의 테이블을 limit 조건으로 조회하면 맵리듀스 잡을 실행하지 않고 파일을 직접 조회해서 출력

```
SELECT * from iris
LIMIT 10;
```

```
OK
1       5.1     3.5     1.4     0.2     setosa
2       4.9     3.0     1.4     0.2     setosa
3       4.7     3.2     1.3     0.2     setosa
4       4.6     3.1     1.5     0.2     setosa
5       5.0     3.6     1.4     0.2     setosa
6       5.4     3.9     1.7     0.4     setosa
7       4.6     3.4     1.4     0.3     setosa
8       5.0     3.4     1.5     0.2     setosa
9       4.4     2.9     1.4     0.2     setosa
10      4.9     3.1     1.5     0.1     setosa
```

17.2.4 집계 함수

하이브에서 지원하는 집계 함수

함수	내용
COUNT(1), COUNT(*)	전체 데이터 건수를 반환
COUNT(DISTINCT 칼럼)	유일한 칼럼값의 건수를 반환
SUM(칼럼)	칼럼값의 합계를 반환
SUM(DISTINCT 칼럼)	유일한 칼럼값의 합계를 반환
AVG(칼럼)	칼럼값의 평균을 반환
AVG(DISTINCT 칼럼)	유일한 칼럼값의 평균을 반환
MAX(칼럼)	칼럼의 최댓값을 반환
MIN(칼럼)	칼럼의 최솟값을 반환

- COUNT(1)

```
SELECT COUNT(1) FROM iris
```

```
Total jobs = 1
Launching Job 1 out of 1s, Fetched: 151 row(s)
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1626222563964_0003, Tracking URL = http://0.0.0.0:8089/pr
oxy/application_1626222563964_0003/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1626222563964_000
3
Hadoop job information for Stage-1: number of mappers: 1; number of reducers
: 1
2021-07-14 16:18:30,060 Stage-1 map = 0%,  reduce = 0%
2021-07-14 16:18:38,688 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.2
8 sec
2021-07-14 16:18:48,321 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1
0.38 sec
MapReduce Total cumulative CPU time: 10 seconds 380 msec
Ended Job = job_1626222563964_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.38 sec   HDFS Read: 22
230 HDFS Write: 103 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 380 msec
OK
450
Time taken: 32.675 seconds, Fetched: 1 row(s)
```

- 이 쿼리문은 하나의 잡으로 구성됨
- 1개의 맵 태스크와 1개의 리듀스 태스크를 실행
- 매퍼와 리듀서의 수행 단계가 퍼센트로 표시되며, 작업이 완료되면 몇 개의 맵 태스크와 리듀스 태스크가 실행됐고 HDFS와 CPU 자원은 얼마나 소모했는지 표시
- GOUPBY

```
SELECT Species, COUNT(*) AS count_by_species FROM IRIS
GROUP BY Species;
```

- AVG

```
SELECT Species, AVG(SepalLength) AS AVG_SepalLength, AVG(SepalWidth) AS AVG_SepalWidth
FROM iris
where PetalLength>2 AND PetalWidth<6
GROUP BY Species;
```

하이브의 내장 함수

Aa 함수	내용

Aa 함수	내용
concat(칼럼a, 칼럼b)	각 칼럼a의 값 뒤에 각 칼럼b의 값을 붙여서 반환. 예를 들어 칼럼a의 1행이 facebook이고, 칼럼b의 1행이 hive라면 facebookhive를 반환
substr(칼럼a, int start_index)	칼럼a의 start_index에서 마지막 인덱스까지 잘라낸 문자열을 반환. 예를 들어,칼럼a의 어떤 행이 hadoop이고 start_index가 4라면 oop를 반환. 즉, index는 1부터 시작
substr(칼럼a, int start_index, int length)	칼럼a의 start_index에서 설정한 length만큼 잘라낸 문자열을 반환. 예를 들어,칼럼a의 어떤 행이 hadoop이고 start_index가 4, length가 2라면 oo를 반환. 즉, index는 1부터 시작
upper(칼럼)	문자열을 대문자로 변환해서 반환. 예를 들어 칼럼의 값이 hive라면 HIVE를 반환
ucase(칼럼)	upper 함수와 동일
lower(칼럼)	문자열을 소문자로 변환해서 반환. 예를 들어 칼럼의 값이 HIVE라면 hive를 반환
lcase(칼럼)	lower 함수와 동일
trim(칼럼)	문자열 양쪽에 있는 공백을 제거. 예를 들어 칼럼의 값이 ' hive ' 라면 'hive'반환
ltrim(칼럼)	문자열 왼쪽에 있는 공백을 제거. 예를 들어 칼럼의 값이 ' hive ' 라면 'hive '반환
rtirm(칼럼)	문자열 양쪽에 있는 공백을 제거. 예를 들어 칼럼의 값이 ' hive ' 라면 ' hive'반환
regexp_replace(칼럼, string regex, string replacement)	문자열 str의 정규 표현식 regex와 일치하는 모든 문자열을 replacement로 변경해서 반환. 예를 들어 칼럼의 어떤 값이 'hadoop'이고 regex가 'op', replacement가 ''라면, 'had'반환
from_unixtime(칼럼)	유닉스 시간 문자열(1970-01-01 00:00:00 UTC)를 현재 시스템의 시간대로 변경해서 반환
to_date(칼럼)	타임스태프 문자열에서 시간을 제외한 날짜값만 반환. 예를 들어, "2012-09-01 00:00:00"은 "2012-09-01"을 반환
round(칼럼)	값에 대한 반올림 정수값(BIGINT)을 반환
floor(칼럼)	값보다 작거나 같은 최대 정수값(BIGINT)를 반환
ceil(칼럼)	값보다 크거나 같은 최소 정수값(BIGINT) 반환
rand(). rand(int seed)	랜덤값을 반환. seed 파라미터로 랜덤값의 범위를 설정할 수 있음
year(칼럼)	날짜 혹은 타임스태프 문자열에서 연도만 반환. 예를 들어, "2012-09-11 00:00:00"은 "2012"를 반환
month(칼럼)	날짜 혹은 타임스태프 문자열에서 월만 반환. 예를 들어, "2012-09-11 00:00:00"은 "09"를 반환
day(칼럼)	날짜 혹은 타임스태프 문자열에서 일만 반환. 예를 들어, "2012-09-11 00:00:00"은 "11"를 반환
get_json_object(string json_string, string path)	디렉터리 path에서 문자열 json_string으로부터 json 객체를 추출하고 json 문자열로 반환. 만약 json이 유효하지 않으면 null 반환
size(MAP<K.V>)	맵 타입의 엘리먼트 개수를 반환
size(Array<t>)	배열 타입의 엘리먼트 개수를 반환
cast(칼럼 as<type>)	정규 표현식 expr을 type으로 타입을 변환. 예를 들어, cast('100' as BIGINT)는 '100'을 BIGINT로 변환해서 반환. 변환에 실패할 경우 null 값 반환

17.2.5 조인

- 맵리듀스로 조인을 구현하려면 수십 줄 이상의 클래스를 작성해야 함.
- 하지만 하이브를 이용하면 간단하게 조인 수행 가능
- 하이브 조인 제약사항
  1. 하이브는 EQ 조인만 지원.

> 📝 EQ 조인이란?
> 두 테이블을 대상으로 동일성을 비교한 후, 그 결과를 기준으로 조인.
> 조인 서술자로 등호(=)만 사용 가능(SQL은 ==가 아닌 =)

2. 하이브는 FROM 절에 테이블 하나만 지정할 수 있고, ON 키워드를 사용해 조인을 처리해야 함.

17.2.5.1 내부 조인(INNER JOIN)

- iris 데이터와 내부 조인하기 위해 iris의 species를 예측한 데이터인 predicted_iris 테이블 생성 및 데이터 업로드
- 테이블 생성

```
CREATE TABLE predicted_iris(Id int, Predicted string)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

- 데이터 업로드
  - scp C:/Users/이혜림/Downloads/predicted_iris.csv hduser@127.0.0.1:/usr/local/hadoop

```
LOAD DATA LOCAL INPATH '/usr/local/hadoop/predicted_iris.csv'
OVERWRITE INTO TABLE predicted_iris;
```

- 예측값과 실제값 조인
  - 각 테이블에 별칭 사용

```
SELECT A.Id, A.Species, B.Predicted
FROM iris A
JOIN predicted_iris B ON (A.Id=B.Id);
```

```
SELECT A.Species, B.Predicted, Count(*)
FROM iris A
JOIN predicted_iris B ON (A.Id=B.Id)
WHERE A.Id='setosa'
GROUP BY A.Species, B.Predicted
LIMIT 10;
```

- 3개 테이블 조인 예시

```
SELECT A.Id, A.Species, B.Predicted, C.Predicted2
FROM iris A
JOIN predicted_iris B ON (A.Id=B.Id)
JOIN predicted_iris2 C ON (B.Id=C.Id);
```

17.2.5.2 외부 조인(Outer Join)

- 왼쪽 외부 조인 쿼리
  - 왼쪽 테이블에는 존재하지만 오른쪽 테이블에는 존재하지 않으면, 오른쪽 테이블의 칼럼 값에 NULL 값이 반환됨

```
SELECT A.Id, A.Species, B.Predicted
FROM iris A
LEFT OUTER JOIN predicted_iris B ON (A.Id=B.Id)
WHERE A.Species='setosa'
LIMIT 10;
```

17.2.6 버킷(Bucket) 활용

- 하이브는 효율적인 쿼리문 수행을 위해 버킷이라는 데이터 모델을 제공.
- 버킷은 버킷 칼럼 해시를 기준으로 데이터를 지정된 개수의 파일로 분리해서 저장.
  - 버킷을 사용하면 쿼리의 성능을 향상시킬 수 있음. 예를 들어, 조인키로 버킷을 생성해 두면 생성된 버킷 중 필요한 버킷만 조회하면 되기 때문에 디렉터리 전체를 풀스캔하는 것보다 훨씬 빠르게 작업을 처리할 수 있음.
  - 버킷을 이용하면 데이터 샘플링을 이용한 다양한 쿼리를 수행할 수 있음
- 테이블을 생성할 때 다음과 같은 형태로 선언.
  - 이때 버킷을 너무 작은 크기로 만들지 않게 주의해야 함. 하둡에서 너무 작은 파일을 많이 처리하게 되면 부하가 발생하기 때문
  - 버킷은 사용자가 생각하는 샘플 데이터와 크기가 같거나 작아야 함.

```
CLUSTERED BY (칼럼) INTO 버킷 개수 BUCKETS;
```

- 버킷 테이블 생성(3)개

```
CREATE TABLE iris_bucket(Id INT, Species STRING)
CLUSTERED BY (Species) INTO 3 BUCKETS;
```

- 데이터 업로드

```
INSERT OVERWRITE TABLE iris_bucket
SELECT Id, Species
FROM iris
WHERE iris.Species='setosa';
```

- 파티션 생성 확인
  - 0000nn_0파일은 모두 버킷을 나타냄

```
hive> dfs -ls /user/hive/warehouse/iris_bucket;
```

```
hive> dfs -ls /user/hive/warehouse/iris_bucket;
Found 3 items
-rwxrwxr-x   1 hduser supergroup          0 2021-07-15 11:54 /user/hive/warehouse/iris_bucket/000000_0
-rwxrwxr-x   1 hduser supergroup          0 2021-07-15 11:54 /user/hive/warehouse/iris_bucket/000001_0
-rwxrwxr-x   1 hduser supergroup       1542 2021-07-15 11:54 /user/hive/warehouse/iris_bucket/000002_0
```

- 쿼리문에서 버킷 데이터 샘플을 사용하는 방법 : TABLESAMPLE
  - 3개의 버킷 중 첫 번째 버킷에서 샘플을 조회

```
SELECT Species
FROM iris_bucket
TABLESAMPLE(BUCKET 3 OUT OF 3);
```

17.3 파티션 테이블

- 하이브 테이블에 데이터를 로딩하는 3가지 방법

1. LOAD DATA 명령어 : 로컬 파일 혹은 HDFS 파일을 로딩. (data loading)
2. INSERT INTO TABLE 구문 : 기존 데이터를 덮어쓰지 않고 데이터를 입력. (adding data)
3. INSERT OVERWRITE 구문 : 데이터를 입력할 때 기존 데이터를 덮어씀. (replacing data)

- 1(data loading) 예시

```
LOAD DATA LOCAL INPATH '/usr/local/hadoop/iris1.csv'
OVERWRITE INTO TABLE iris
PARTITION (Species='setosa');
```

- 2(adding data) 예시

```
INSERT INTO TABLE iris2
SELECT Id, Species
FROM iris
WHERE iris.Species='setosa'
```

- 3(replacing data) 예시

```
INSERT OVERWRITE TABLE iris_bucket
SELECT Id, Species
FROM iris
WHERE iris.Species='setosa';
```

- 파티션 파일에 데이터를 로딩하기 위해서는 3 구문에 PARTITION 절이 추가되어 있어야 함.
  - PARATION (파티셔닝할 칼럼 이름 = '칼럼값')
- 파티션 칼럼이 한개일 때 예시

```
INSERT OVERWRITE TABLE iris2
PARTITION (Species = 'setosa')
SELECT Id, Species
From iris
WHERE iris.Species='setosa';
```

- 파티션 칼럼이 두개일 때 예시

```
INSERT OVERWRITE TABLE iris2
PARTITION (Species = 'setosa', SepalLength = '2')
SELECT Id, Species, SepalLength
from iris
WHERE iris.Species='setosa' AND SepalLength = 2
```

- 만약 해당 데이터의 파티션 칼럼 값을 모른다면 다이내믹 파티션 기능을 사용
  - load data에서는 사용 불가능. insert 구문에서만(hive에 저장된 데이터에 한에서만) 사용 가능
  - 파티션 칼럼명만 설정하고 칼럼값은 설정하지 않으면 다이내믹 파티션 기능을 사용할 수 있음.
  - 하이브는 사용자의 실수로 너무 많은 파티션이 생성되는 것에 대비하기 위해 다이내믹 파티션을 비활성화. 따라서 다이나믹 파티션 설정 활성화 이후 가능

다이내믹 파티션 속성

Aa 속성	기본값	내용
hive.exec.dynamic.partition	false	다이내믹 파티션 사용 여부를 설정
hive.exec.dynamic.partition.mode	strict	모든 파티션을 동적으로 설정하고 싶다면 nonstrict으로 설정
hive.exec.max.dynamic.partitions.permode	100	각 매퍼 및 리듀서가 생성할 수 있는 최대 다이내믹 파티션 개수. 예를 들어, 하나의 질의에 매퍼 두 개가 실행 될 경우 각 매퍼는 100개의 파티션을 생성할 수 잇으며, 해당 질의에서는 최대 200개의 파티션이 생성될 수 있음
hive.exec.max.dynamic.partitions	1000	하나의 절에서 만들어질 수 있는 최대 다이내믹 파티션 개수
hive.exec.max.created.files	100000	하나의 질의에서 만들어질 수 있는 최대 다이내믹 파티션 개수

```
# 다이나믹 파티션 설정 활성화
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> set hive.exec.dynamic.partition=true;
```

```
# 파티셔닝
INSERT OVERWRITE TABLE iris2
PARTITION (Species, SepalLength)
SELECT Id, Species, SepalLength
from iris
WHERE iris.Species='setosa' AND SepalLength = 2
```

- 해당 테이블의 모든 파티션 정보를 출력

```
show partitions iris;
```

17.4 데이터 정렬

- 모든 정렬 질의문에 desc 를 붙이면 내림차순 정렬

17.4.1 ORDER BY

- 질의문 결과에 대해 전체 정렬을 수행하며, 이를 위해 마지막 맵리듀스 잡에서 하나의 리듀서만 실행.
- 따라서 정렬 대상 데이터가 클 경우 성능이 느려지는 단점이 존재.

```
select *
FROM iris
ORDER BY SepalLength
limit 10;
```

17.4.2 SORT BY

- 전체 정렬을 포기하는 대신 질의 성능을 높이는 데 초점을 맞춤. 즉, 부분데이터에 대한 정렬
- ORDER BY 절과는 다르게 여러 개의 리듀서를 실행하고, 리듀서별로 출력 결과를 정렬.
- 이 방법은 각 리듀서가 같은 키만 받는 것을 보장하지 않기 때문에 각 리듀서의 출력 결과에 동일 한 키가 생성될 수 있음.

```
select *
FROM iris
SORT BY SepalLength
limit 10;
```

17.4.3 DISTRIBUTED BY

- 같은 키를 가진 레코드(컬럼 값이 같은 레코드)가 같은 리듀서로 보내지는 것을 보장
- group by와 비슷한 기능. 같은 컬럼 값끼리 묶어서 같은 리듀서로 보냄
- SORT BY와 함께 사용할 경우, 각 리듀서는 키가 중복되지 않고, 정렬을 수행할 수 있음.

```
SELECT *
FROM iris
DISTRIBUTED BY SepalLength
SORT BY SepalLength DESC;
```

17.4.4 CLUSTERED BY

- CLUSTERED BY는 DISTRIBUTED BY와 SORT BY를 함께 사용하는 것과 동일한 기능

```
SELECT *
FROM iris
CLUSTERED BY SepalLength DESC;
```

17.5 데이터 저장 포맷

- 하이브의 데이터 저장 포맷은 파일 포맷과 레코드 포맷으로 구분
  - 파일 포맷 : 파일에 레코드를 저장할 때 인코딩되는 방식. 텍스트 파일, 시퀀스 파일 등이 파일 포맷에 해당
  - 레코드 포맷 : 칼럼 값이 레코드 내에서 인코딩되는 방식

17.5.1 SerDe

- 하이브는 레코드 내에 인코딩돼 있는 데이터를 저장, 처리하기 위한 SerDe(Serialize/Deserialize)를 제공. 하이브는 SerDe를 이용하여 데이터를 읽고 쓸 수 있음.
  - 직렬화 : ROW Object → Serialize → [key, value] → OutputFileFormat → HDFS Files
  - 비직렬화 : HDFS Files → InputFileFormat → [key, value] → Deserialize → Row Object
  - 하이브는 INSERT나 CTAS(CREATE TABLE AS SELECT) 구문을 실행할 때 SerDe 를 이용해 레코드의 직렬화(Serialize)를 수행. 또한 테이블에 저장된 데이터를 조회할 때는 SerDe를 통해 역직렬화(Deserialize)를 수행.
- 하이브는 다음과 같은 SerDe를 제공
  - LazySimpleSerDe : CSV 파일과 같은 단순 텍스트 파일용. 기본 SerDe로 사용

- LazyBinarySerDe : 텍스트 파일을 바이너리 형태로 저장할 때 사용
- ColumnarSerDe : RC 파일을 텍스트 형태로 저장할 때 사용
- LazyBinaryColumnarSerDe : RC 파일을 바이너리 형태로 저장할 때 사용
- OrcSerde : ORC 파일용 SerDe
- RegexSerDe : 정규표현식으로 설정된 칼럼이 포함된 텍스트 데이터를 조회할 때 사용
- ThriftByteStreamTypedSerDe : 쓰리프트 바이너리 데이터용 SerDe
- HBaseSerDe : H베이스 테이블에 데이터를 저장하고 조회할 때 사용

17.5.1.1 RegexSerDe를 이용해 아파치 웹 로그 조회용 테이블 생성

- 아파치 웹 로그 조회용 테이블 생성 스크립트

```
CREATE TABLE apache_access_logs(
remote_host STRING,
remote_logname STRING,
remote_userid STRING,
finish_time STRING,
request STRING,
status_code STRING,
size STRING,
referer STRING,
user_agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES(
"input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-:\\[[^\\]]*\\]) ([^ \"]*:\"[^\"]*\") (-:[0-9]*)
(-:[0-9]*)(?: ([^ \"]*:\".*\") ([^ \"]*:\".*\"))?",
"output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"
)
STORED AS TEXTFILE;
```

- 데이터 다운로드

```
cd /usr/local/hadoop
wget https://github.com/blrunner/hadoop-beginners-example/blob/master/src/main/resources/chapter17/apache_access.log
```

- 데이터 로드

```
load data local inpath '/usr/local/hadoop/apache_access.log'
overwrite into table apache_access_logs;
```

17.5.2 파일 포맷

- 하이브는 다양한 파일 포맷을 지원. 데이터의 크기 및 성능을 고려해 적절한 파일 포맷을 설정해야 함.
- 파일 포맷은 CREATE의 STORED AS 옵션으로 설정 가능
- 하이브는 텍스트 파일을 기본 파일 포맷으로 사용

하이브 지원 파일 포맷

Aa 항목	텍스트 파일	시퀀스 파일	RC 파일	ORC 파일	파케이
저장 기반	로우 기반	로우 기반	칼럼 기반	칼럼 기반	칼럼 기반
압축	파일 압축	레코드/블록 압축	블록 압축	블록 압축	블록 압축
스플릿 지원	지원	지원	지원	지원	지원

Aa 항목	텍스트 파일	시퀀스 파일	RC 파일	ORC 파일	파케이
압축 적용 시 스플릿 지원	미지원	지원	지원	지원	지원
하이브 키워드	TEXTFILE	SEQUENCEFILE	RCFILE	ORCFILE	PARQUET

- 시퀀스파일 테이블 생성 구문

```
// 텍스트 파일 SerDe
CREATE TABLE table1(id int, name string, score float, type string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS SEQUENCEFILE;

// 바이너리 SerDe
CREATE TABLE table2(id int, name string, score float, type string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazybinary.LazyBinarySerDe'
STORED AS SEQUENCEFILE;
```

17.5.2.1 RC 파일

- RC(Record-Colunmar) 파일은 칼럼 기반의 파일 포맷이며, 로우 그룹 단위로 레코드를 관리.
- RC 파일 장점
  - 칼럼 개수가 많은 테이블에 유리.
    - 로우 기반 테이블은 특정 칼럼만 조회할 경우에도 내부적으로 해당 레코드의 전체 칼럼을 조회하게 됨. 왜냐하면 각 레코드에 모든 칼럼이 저장돼 있기 때문. 하지만 RC 파일에서는 각 칼럼별로 분리되어 저장돼 있기 때문에 필요한 칼럼만 조회.
  - 데이터 조회 속도가 로우 기반 파일보다 뛰어남
  - RC 파일은 칼럼별로 압축하기 때문에 해당 칼럼이 필요한 시점에만 압축을 해제
- RC 파일 단점
  - 데이터 타입별로 효과적인 압축 방식을 적용할 수 없음
    - 칼럼 타입에 영향을 받고, 한 번에 하나의 레코드에만 SerDe를 적용할 수 있기 때문
  - Map, List와 같은 데이터 타입의 특정 값을 조회할 때 비효율적
    - 순차적인 레코드 접근을 위해 설계돼 있어서 Map과 List 내의 불필요한 데이터도 함께 조회하기 때문

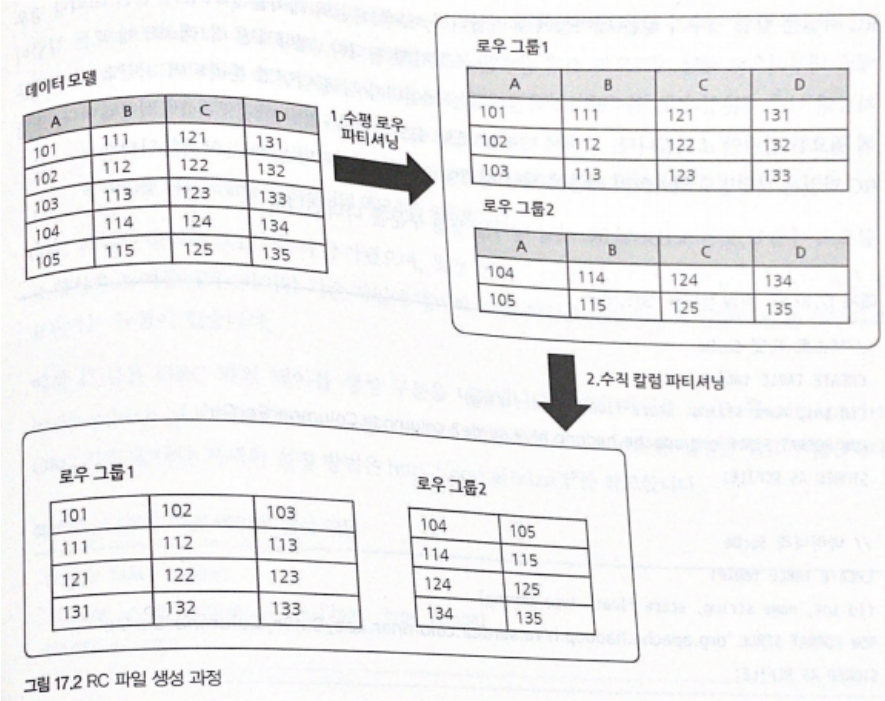


그림 17.2 RC 파일 생성 과정

1. 수평 로우 파티셔닝
    - 로우 데이터를 로우 그룹으로 수평 파티셔닝. 로우 그룹의 크기는 사용자가 설정할 수 있으며, 기본값은 HDFS의 블록 크기와 동일
2. 수직 칼럼 파티셔닝
    - 로우 그룹을 수직으로 칼럼 파티셔닝. 이때 로우 그룹의 동일한 칼럼끼리 분리되어 저장.
    - 로우 그룹은 3개의 포맷으로 구성
        - 싱크 마커(sync marker) : 각 로우 그룹을 구분하는 데 사용
        - 메타데이터 헤더 : 로우 그룹에 저장된 레코드 건수, 각 칼럼별 바이트 크기, 각 필드별 바이트 크기를 저장
        - 테이블 데이터 : 동일한 칼럼의 필드값이 연속적으로 저장됨. 예를 들어, 17.2의 경우 로우 그룹 1과 로우 그룹 2의 첫 번째 레코드에 연속적으로 저장됨

```
// 텍스트 파일 SerDe
CREATE TABLE table3(id int, name string, score float, type string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe'
STORED AS RCFILE;

// 바이너리 SerDe
CREATE TABLE table4(id int, name string, score float, type string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe'
STORED AS RCFILE;
```

17.5.2.2 ORC 파일

- 앞서 언급한 RC 파일의 모든 단점을 개선한 파일 포맷
- ORC 파일의 장점
    - 높은 성능 향상

    - 높은 압축률
- ORC 파일의 단점
    - 상당히 많은 시간 소요
    - 하이브 외에 다른 플랫폼에서는 사용할 수 없음
- ORC 파일 특징
    1. 하나의 파일에 칼럼을 JSON처럼 중첩(nested) 구조로 저장할 수 있음
    2. Map, Struct, List 등을 칼럼값 대신 사용할 수 있음
    3. 네임노드의 부하를 줄이기 위해 하나의 태스크는 하나의 출력 파일만 생성
    4. 빠른 조회를 위해 파일 내에 경량 인덱스를 저장
    5. 필터 조건에 해당되지 않는 로우 그룹울 제외함으로써 빠른 스캔이 가능
    6. 파일을 읽고 저장할 때 일정량의 메모리만 필요

```
CREATE TABLE table5(id int, name string, score float, type string)
STORED AS ORC;
```

17.5.2.3 파케이

- ORC 파일이 하이브 외에 다른 플랫폼에서 사용할 수 없는 단점을 극복
- 파케이(Parquet) 장점
    1. ORC 파일처럼 중첩 구조로 칼럼을 저장. 따라서 칼럼을 계층적으로 늘려갈 수 있음.(마치 SNS에 댓글을 다는 것처럼)
    2. 범용적으로 사용 가능
    3. 하나의 파일에 여러 칼럼을 저장하기 때문에 조인 작업을 최소화할 수 있음
    4. 다양한 압축과 인코딩 방식을 적용할 수 있음

```
CREATE TABLE table5(id int, name string, score float, type string,
mp MAP<STRING, STRING>,
1st ARRAY<STRING>
strct STRUCT<A:STRING, B:STRING>)
PARTITIONED BY (part string)
STORED AS ORC;
```

17.6 하이브에서 파일 저장 후 윈도우로 보내기

- 하이브에서 로컬로 파일 저장

```
INSERT OVERWRITE LOCAL DIRECTORY '/usr/local/hadoop'
ROW FORMAT DELIMITED
FIELS TERMINATED BY ','
SELECT * from iris;
```

- 리눅스에서 윈도우로 파일 전송
    - scp linux계정@서버주소:파일 windows경로

```
scp hduser@127.0.0.1:/usr/local/hadoop/000000_0 C:/Users/이혜림/Downloads
```