Chapter

01 하둡 에코시스템 - 아파치 하이브

●소개 및 구조이해

➢ HIVE는 하둡의 최상위층에 있는 하부프로젝트 패키지로서 대량의 로그 데이터 처리를 위해 페이스북에서 먼저 개발되었다.

➢HIVE는 SQL에 익숙한 사용자들을 위해 HIVE QL이라는 SQL과 유사한 언어를 사용하여 데이터를 처리하도록 하고 있다.

1. 하둡기반의 분석도구
2. 정형화된 데이터 관리
3. SQL과 유사한 H-SQL 사용
4. 메타스토어 사용

● 메타스토어
➢테이블, 행, 컬럼, 스키마등 관계형 데이터베이스와 유사한 개념으로 HDFS의 데이터를 조회하고, 시스템을 관리하기 위한 기능을 제공.

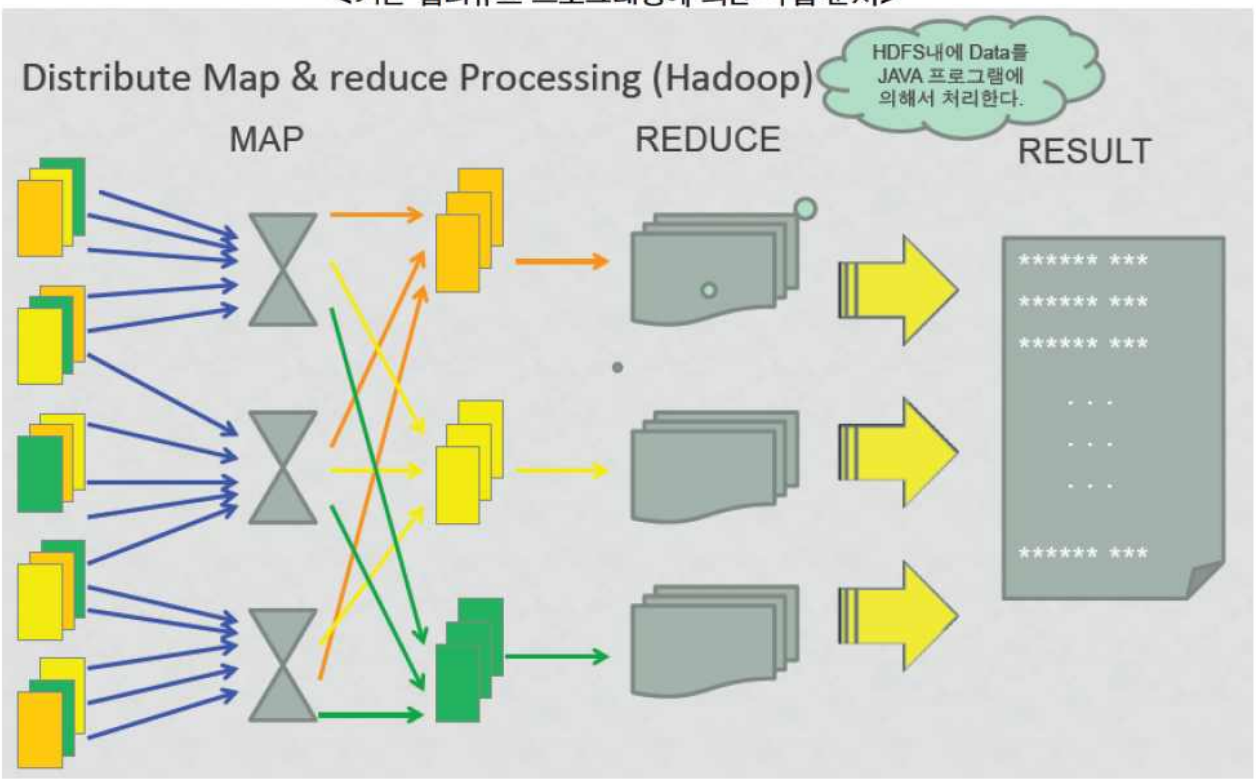
● H-SQL 쿼리
➢다소 복잡한 맵리듀스 프로그램 대신 HIVE QL을 사용해서 데이터를 처리.

MEMO

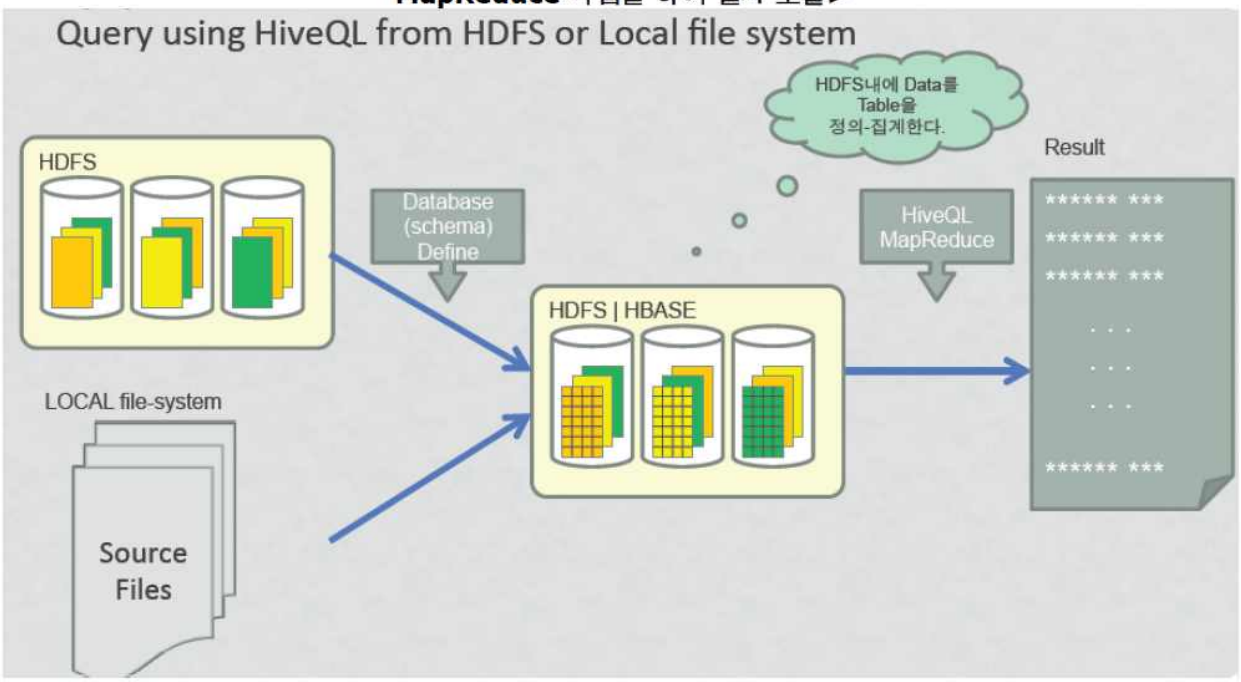
❖

● 맵리듀스 프로그램과 하이브

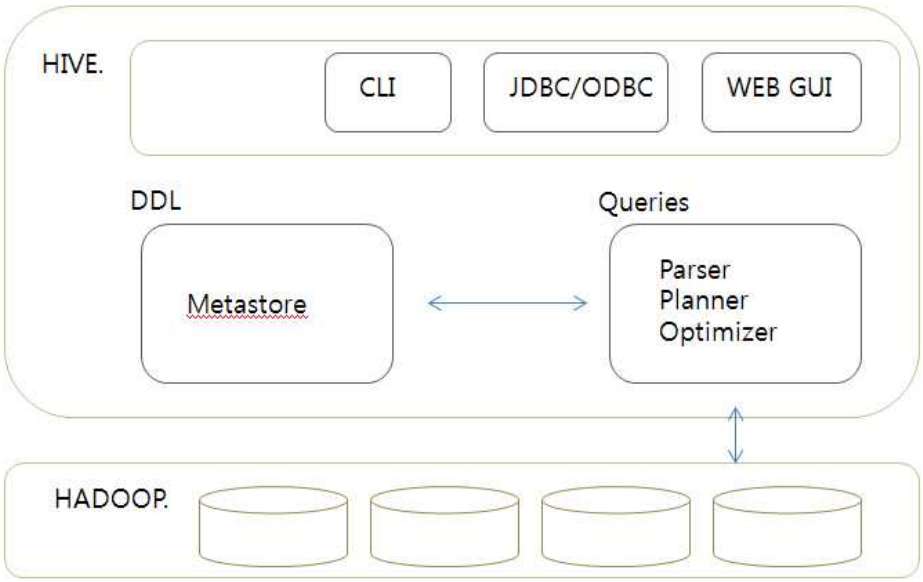
<기존 맵리듀스 프로그래밍에 의한 작업 순서>



<HDFS또는 LOCAL FILE SYSTME의 내용을 HIVE QL을 이용하여 MapReduce 작업을 하여 결과 도출>



● 아키텍처



#CONNECTER : CLI 와 JDBC/ODBC, WEB GUI 를 제공해주고 있습니다.
-CLI : command line interface 의 약자, hive 의 명령 프롬프트를 제공
-JDBC/ODBC : 데이터베이스를 연동할 수 있습니다.
-WEB GUI : 웹페이지

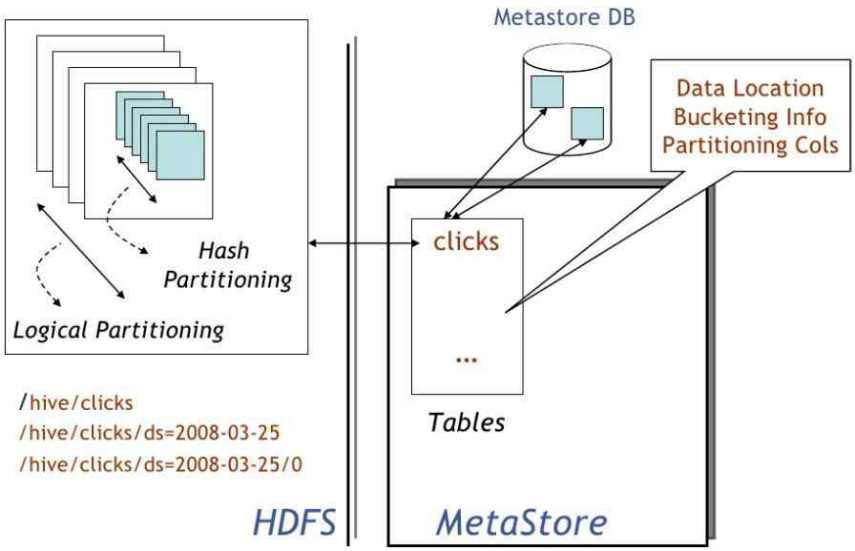
#DDL : metastore를 통하여 스키마 정보를 관리

#Queries : HIVE 는 HIVE QL 이라는 SQL 과 상당히 유사한 언어 사용.

MEMO

❖

●메타스토어



1. 테이블
컬럼 타입 (int, float, string, boolean)
리스트 : map (JSON과 유사)

2. 파티션
테이블은 하나 이상의 파티션 키를 가짐
데이터 접근을 위한 테이블 구조의 최적화(인덱스)
Ex) 날짜별 파티션 테이블

3. 버킷
파티션의 데이터는 특정 해쉬 함수에 의해서 버킷으로 분할
특정 범위의 해쉬 파티션 (샘플링에 이용)

MEMO

❖ 물리적 레이아웃
1. HDFS에 위치한 Hive 홈 디렉토리
=>/user/hive/warehouse

2. 테이블은 Hive 홈의 하위 디렉토리에 생성
=>/user/hive/warehouse/clicks

3. 파티션은 해당 테이블의 하위 디렉토리에 생성
=>/user/hive/warehouse/clicks/ds=2014

●H-SQL

➢HIVE는 SQL에 익숙한 사용자들을 위해 HIVE QL이라는 SQL과 유사한 언어를 사용하여 데이터를 처리하도록 하고 있다.

1. SQL-92 표준
2. 오픈소스 성격
3. 확장된 기능. MULTI TABLE INSERT 등
4. 맵리듀스의 영향. TRANSFORM, MAP, REDUCE 등
5. http://wiki.apache.org/hadoop/Hive/LanguageManual

●H-SQL 수행

➢실행 단계
1단계, 구문 분석
2단계, 의미 분석
3단계, 실행계획 (논리적)
4단계, MapReduce Job
5단계, 수행 (물리적)

➢실행 계획
1단계, DDL : 메타 데이터 작업
2단계, DML : LOAD -> HDFS 작업
3단계, DML : INSERT와 H-SQL -> MapReduce Job

MEMO

❖

●데이터 타입

➢ Atomic Number

TINYINT	: 8비트 부호화 정수	ex) 1
SMALLINT	: 16비트 부호화 정수	ex) 1
INT	: 32비트 부호화 정수	ex) 1
BIGINT	: 64비트 부호화 정수	ex) 1
FLOAT	: 32비트 부동 소수점	ex) 1.0
DOUBLE	: 64비트 부동 소수점	ex) 1.0

➢ Atomic Character

STRING	: UTF-8 형식의 문자 배열	ex) 'A' 'a'

* 형변환 : CAST('1' AS INT)

➢ Object

ARRAY	: 순서화된 필드의 집합	ex) array(1,2)
MAP	: 키/값 쌍의 비 순서화된 집합	ex) map('a',1 , 'b', 2)
STRUCT	: 이름있는 필드의 집합	ex) struct('a', 1, 1.0)

Ex)

```
CREATE TABLE complex (
col1 ARRAY <INT>,
col2 MAP <STRING,INT>,
col3 STRUCT <a:STRING, b:INT, c:DOUBLE>
);

> SELECT col1[0], col2['b'], col3.c FROM complex;
  (1   2   1.0)
```

MEMO

❖

●스크립트

CREATE

➢테이블
Expression

```
CREATE EXTERNAL TABLE target ( dumm STRING )
        LOCATION '/user/tom/target';
LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE target;
```

➢파티션
Expression

```
CREATE TABLE logs ( ts INT, line STRING )
PARTITIONED BY ( dt STRING, country STRING );
LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE target
PARTITION (dt='2010-01-01', country='GB');
```

➢버킷
Expression

```
CREATE TABLE bucketed ( id INT, name STRING)
CLUSTERED BY (id) STORED BY (id ASC) INTO 4 BUCKETS;
```

MEMO

❖

●스크립트

ALTER

➢변경
Expression

ALTER TABLE source RENAME target;

ALTER TABLE target ADD COLUMNS (col3 STRING);

ALTER TABLE target DROP PARTITION (dt='2010-0101-');

➢삭제
Expression

DROP TABLE source;

➢복사
Expression

CREATE TABLE new_table LIKE exsting_table;

MEMO

❖

●스크립트

ROW FORMAT

➢**FIELDS**
CSV : 콤마 구분 (‘,’) : defualt
MR : Tab 구분 (‘₩t’) : mapreduce

Expression

DELIMITED FIELDS TERMINATED BY < output format >

➢**LINE**
Defualt : Enter 구분 (‘₩n’)

Expression

LINES TERMINATED BY < output format >

➢**FILE**
순차파일 : seq 확장자 <SEQUENCEFILE>
텍스트파일 : txt 확장자 <TEXTFILE>

Expression

STORED AS < storage format >;

MEMO

❖

●SELECT Syntax

1. SELECT문은 조인 또는 서브쿼리의 일부가 될 수 있습니다.
2. 관계는 테이블, 뷰 , 조인 , 서브쿼리 등으로 나타냅니다.
3. 간단한 예를 들어 테이블 t1의 모든 열과 모든 행을 검색합니다.

Expression

```
SELECT * FROM t1
```

●WHERE Clause

➢WHERE 절은 조건 Boolean[expression] 입니다.
➢간단한 예를 들어 sales테이블에서 amount보다 작고 region값이 'US'인 경우를 검색합니다.

Expression

```
SELECT * FROM sales WHERE amount > 10 AND region = "US"
```

MEMO

❖

●ALL and DISTINCE Clause

➢ALL과 DISTINCE 옵션은 중복 행을 반환해야 하는지 여부를 확인합니다.
➢기본값은 ALL입니다.
➢DISTINCT는 결과 행에서 중복행을 제거하도록 지정합니다.

Expression

```
hive> SELECT col1, col2 FROM t1   << ALL
1 3
1 3
1 4
2 5
hive> SELECT DISTINCT col1 FROM t1  << DISTINCT [ col1 ]
1
2
```

●LIMIT Clause

➢결과가 반환되는 행 수를 지정합니다. 반환되는 행은 무작위로 선택 됩니다.
➢다음 예로 t1에서 5행을 무작위로 반환 시켜보겠습니다.

Expression

```
SELECT * FROM t1 LIMIT 5
```

MEMO

❖

●Group By Syntax

➢컬럼 리스트의 중복을 제거하고 정렬 후 결과를 반환한다.

Expression

```
SELECT pv_users.gender, count(DISTINCT pv_users.userid),
        count(*), sum(DISTINCT pv_users.userid)
FROM pv_users GROUP BY pv_users.gender
```

●Multi Inserts

➢집합체 도는 단순 선정한 출력은 여러 테이블과 하둡 DFS 파일에 수행 결과를 한번에 보낼 수 있다. 다음과 같은 예를 들어 보자.

Expression

```
FROM pv_users

INSERT OVERWRITE TABLE pv_gender_sum
SELECT pv_users.gender, count(DISTINCT pv_users.userid)
GROUP BY pv_users.gender

INSERT OVERWRITE DIRECTORY '/user/facebook/tmp/pv_age_sum'
SELECT pv_users.age, count(DISTINCT pv_users.userid)
GROUP BY pv_users.age;
```

MEMO

❖

●Union Syntax

➢Union은 여러 SELECT 결과를 하나의 집합으로 결합하는데 사용합니다.

Expression

```
SELECT u.id, actions.date
FROM (
        SELECT av.uid AS uid  FROM action_video av
    UNION ALL
        SELECT ac.uid AS uid  FROM action_comment ac
        WHERE ac.date = '2008-06-03'
) actions
```

●Subquery Syntax

➢HIVE는 FROM 절에 하위 쿼리를 지원합니다.
➢하위쿼리는 FROM절의 모든 테이블 이름이 있어야 하기 때문에 이름을 부여해야 합니다.

Expression

```
SELECT t3.col
FROM (
  SELECT a+b AS col FROM t1
UNION ALL
  SELECT c+d AS col FROM t2
) t3
```

MEMO

❖

●JOIN Syntax

➢일반적으로 SELECT 쿼리는 전체 테이블을 검색하지만, 테이블이 조항에 의해 분할 사용하여 작성된 경우 쿼리는 파티션 나누기 작업을 수행하고 지정된 파티션과 관련된 테이블의 일부만 검색할 수 있습니다.
➢파티션을 동시에 검색하는 경우 WHERE 조건절을 사용하지 않고 JOIN에 ON 절에 명시하여 수행합니다.

Expression

```
SELECT join_tab.*, join_tab2.*
FROM join_tab JOIN join_tab2
ON (
    join_tab.id = join_tab2.id
   AND join_tab.date >= '2008-03-01'
   AND join_tab.date <= '2008-03-31'
);
```

●사용자 정의 함수

➢UDF (user-defined function)
: 단일 행을 처리하고 결과 또한 단일 행으로 내준다.
: 수학 함수와 문자열 함수와 같은 대부분의 함수가 이 종류에 해당한다.

➢UDAF (user-defined aggregate function)
: 다중 입력 행을 처리하고 단일 출력 행을 생성한다.
: 집계 함수는 COUNT와 MAX같은 함수를 포함한다.

➢UDTF (user-defined tabe-generating function)
: 단일 행을 처리하고 결과로 다중 행(테이블)을 생성한다.

●JOIN Syntax

➢일반적으로 SELECT 쿼리는 전체 테이블을 검색하지만, 테이블이 조항에 의해 분할 사용하여 작성된 경우 쿼리는 파티션 나누기 작업을 수행하고 지정된 파티션과 관련된 테이블의 일부만 검색할 수 있습니다.
➢파티션을 동시에 검색하는 경우 WHERE 조건절을 사용하지 않고 JOIN에 ON 절에 명시하여 수행합니다.

Expression

```
SELECT join_tab.*, join_tab2.*
FROM join_tab JOIN join_tab2
ON (
    join_tab.id = join_tab2.id
   AND join_tab.date >= '2008-03-01'
   AND join_tab.date <= '2008-03-31'
);
```

● 산술연산자

연산자	설명
A+B	A와 B를 더한다
A-B	A에서 B를 뺀다
A*B	A와 B를 곱한다
A/B	A를 B로 나눈다. 만약 피연산자가 정수형이면 나누기를 몫만 반환된다.
A%B	A를 B를 나눈 나머지를 반환한다.
A&B	A와 B를 비트 AND 연산한다.
A|B	A와 B를 비트 OR 연산한다.
A^B	A와 B를 비트 XOR 연산한다.
~A	A의 비트 NOT 연산한다.

● 수학 함수

시그니쳐	설명
Round(d)	DOUBLE 데이터형 D의 반올림값을 BIGINT로 반환
Round(d,N)	DOUBLE 데이터형 D의 N개 소수점 이하의 값까지 반올림값을 DOUBLE 데이터유형으로 반환
Floor(d)	DOUBLE 데이터형 d보다 작거나 값은 값 중 가장 큰 BIGINT 를 반환한다
Ceil(d),Ceiling(DOUBLE d)	D보다 크거나 같은 값 중 가장 작은 BIGINT를 반환한다.
Rand(), Rand(seed)	각 로우마다 난수값이 바뀌면서 반환된다. 인자로 받은 정수 데이터형의 씨드는 반환되는 값을 결정한다.
Pow(d,p), Power(d,p)	D의 P거듭제곱한 값을 반환한다. D와 P는 DOUBLE데이터형이다.
Sqrt(d)	DOUBLE 데이터형 D의 제곱근을 반환한다.
Abs(d)	DOUBLE 데이터형 D의 절대값을 DOUBLE 데이터형으로 반환한다.
Sin(d)	DOUBLE 데이터형 D의 사인값을 DOUBLE 데이터형의 라디안 값으로 반환한다.
Cos(d)	DOUBLE 데이터형 D의 코사인값을 DOUBLE 데이터형의 라디안 값으로 반환한다.
Tan(d)	DOUBLE 데이터형 D의 탄젠트값을 DOUBLE 데이터형의 라디안 값으로 반환한다.
Pi()	파이 상수값을 DOUBLE데이터형으로 반환한다.

MEMO

❖

● 집계 함수

시그니쳐	설명
count(*), count(expr), count(distinct expr)	총 수를 구한다.
sum(col), sum(distinct col)	합을 구한다.
avg(col), avg(distinct col)	평균을 구한다.
min(col), max(col)	최소/최대를 구한다.
variance(col)	분산을 구한다.
stddev_pop(col)	표준 편차를 구한다.
covar_pop(col1, col2)	공분산을 구한다.
corr(col1, col2)	두 집합의 상관관계를 구한다.
collect(set(col)	중복된 요소를 제거한 집합을 구한다.
percentile(정수타입컬럼, p)	컬럼에서 백분위 p에 해당하는 값을 구한다.
percentile_approx(실수타입컬럼, p)	컬럼에서 백분위 p에 해당하는 값을 구한다.

MEMO

❖

● 기타 함수

시그니쳐	설명
length(s)	문자열의 길이를 구한다.
concat(s1, s2, ...), concat(sep, s2, s2)	문자열을 연결한다. sep는 구분자.
substring(s, start), substring(s, start, len)	문자열의 일부를 구한다.
upper(s), ucase(s), lower(s), lcase(s)	대문자/소문자로 변환한다.
trim(s), ltrim(s), rtrim(s)	끝의 공백을 제거한다.
regexp_replace(s, regex, replacement)	정규표현식과 일치하는 부분을 대치한다.
regexp_extract(s, regex, idx)	정규표현식과 일치하는 idx 번째 부분을 구한다.
parse_url(url, partname, key)	URL에서 일부를 구한다.
from_unixtime(int unixtime))	유닉스 기준시로부터 타임스탬프(yyyy-MM-dd HH:mm:ss)를 구한다.
to_date(ts타임스탬프)	타임스탬프에서 'yyyy-MM-dd' 부분을 구한다.
year(ts), month(ts), day(ts)	타임스탬프에서 연, 월, 일 부분을 구한다.

MEMO

❖

Chapter

02 아파치 하이브 활용 실습

● 파일시스템 리스트 조회

➢명령어 : dfs -ls /;

```
hive> dfs -ls /;
Found 3 items
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-13 20:10 /tmp
drwxr-xr-x   - root supergroup          0 2014-11-13 20:06 /user
hive>
```

●파일시스템에 파일 올리기

➢명령어 : dfs -put /home/root/hadoop-1.0.4/README.txt /;

```
hive> dfs -put /home/root/hadoop-1.0.4/README.txt /;
hive> dfs -ls /;
Found 4 items
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-13 20:10 /tmp
drwxr-xr-x   - root supergroup          0 2014-11-13 20:06 /user
hive>
```

로컬시스템에 있는 README.txt 파일을 분산 파일 시스템으로 옮긴다.
dfs -ls 명령어를 이용하여 분산 파일 시스템에 README.txt 파일이 생성된 것을 확인할 수 있다.

MEMO

❖

● 파일시스템 디렉토리 만들기

➢명령어 : dfs -ls /;

```
hive> dfs -ls /;
Found 4 items
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
drwxr-xr-x   - root supergroup          0 2014-11-13 20:06 /user
```

dfs -ls 명령어를 이용하여 분산 파일 시스템의 디렉토리와 파일을 조회 할수 있다.

➢명령어 : dfs -mkdir /data_dir;

```
hive> dfs -mkdir /data_dir;
hive> dfs -ls /;
Found 5 items
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-19 23:25 /data_dir
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
drwxr-xr-x   - root supergroup          0 2014-11-13 20:06 /user
```

dfs -mkdir 명령어로 /data_dir 디렉토리를 생성한 다음 dfs -ls 명령어를 이용하여 분산 파일 시스템을 조회하면 /data_dir 디렉토리가 생성된 것을 확인할 수 있다.

MEMO

❖

● 파일시스템 파일 복사

➢명령어 : dfs -ls /;

```
hive> dfs -ls /;
Found 5 items
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-19 23:25 /data_dir
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
```

dfs -ls 명령어를 이용하여 분산 파일 시스템의 디렉토리와 파일을 조회 할수 있다.
/README.txt 파일을 복사해서 /data_dir 디렉토리 안에 추가로 생성해 보자

➢명령어 : dfs -cp /README.txt /data_dir/;

```
hive> dfs -cp /README.txt /data_dir/
    > ;
hive> dfs -lsr /
    > ;
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-19 23:44 /data_dir
-rw-r--r--   1 root supergroup       1366 2014-11-19 23:44 /data_dir/README.txt
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
```

dfs -cp 명령어로 /README.txt 파일을 복사해서 /data_dir 디렉토리 안에 복사본을 추가 시킨다.
dfs -lsr 명령어를 이용하여 /data_dir 디렉토리 안에 README.txt 파일을 조회할 수 있다.

MEMO

❖

● 파일시스템 파일 이동

➢명령어 : dfs -ls /;

```
hive> dfs -ls /;
Found 5 items
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-19 23:25 /data_dir
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
```

dfs -ls 명령어를 이용하여 분산 파일 시스템의 디렉토리와 파일을 조회 할수 있다.
이동 명령어를 사용하여 /README.txt 파일명을 READMEDATA.txt로 변경 해 보자.

➢명령어 : dfs -mv /README.txt /data_dir/READMEDATA.txt;

```
hive> dfs  -mv /README.txt  /data_dir/READMEDATA.txt;
hive> dfs -lsr /;
drwxr-xr-x   - root supergroup          0 2014-11-19 23:54 /data_dir
-rw-r--r--   1 root supergroup       1366 2014-11-19 23:44 /data_dir/README.txt
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /data_dir/READMEDATA.
txt
```

dfs -cp 명령어로 /README.txt 파일을 /data_dir/로 이동 시킨다.
이때, 이동된 파일은 READMEDATA.txt로 파일명이 변경된다.
dfs -lsr 명령어를 이용하여 /data_dir 디렉토리 안에 README.txt 와 READMEDATA.txt 파일을 조회할 수 있다.

MEMO

❖

● 파일시스템 파일 수 확인

➢명령어 : dfs -count /;

```
hive> dfs -count /;
          17            4                4102 hdfs://localhost:9000/
```

dfs -count 명령어를 이용하여 분산 파일 시스템의 디렉토리와 파일 수를 확인할 수 있다..
내용은 / 아래의 17개의 디렉토리를 조사하였으며, 총 파일은 4개가 존재한다.

➢명령어 : dfs -count /data_dir;

```
hive> dfs -count /data_dir/;
          1            2             2732 hdfs://localhost:9000/data_dir
```

/ data_dir 디렉토리를 조사하였으며, 파일은 2개가 존재한다.

```
hive> dfs -lsr /data_dir;
-rw-r--r--   1 root supergroup       1366 2014-11-19 23:44 /data_dir/README.txt
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /data_dir/READMEDATA.txt
```

dfs -lsr 명령어를 이용하여 /data_dir 디렉토리의 파일 수를 확인하면 두개의 파일이 존재한다.

MEMO

❖

● 파일시스템 파일 삭제

➢명령어 : dfs -ls /;

```
hive> dfs -ls /;
Found 5 items
-rw-r--r--   1 root supergroup       1366 2014-11-19 12:56 /README.txt
drwxr-xr-x   - root supergroup          0 2014-11-19 23:25 /data_dir
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
```

dfs -ls 명령어를 이용하여 분산 파일 시스템의 디렉토리와 파일을 조회 할수 있다.
삭제 명령어를 사용하여 /README.txt 파일명을 삭제해 보자.

➢명령어 : dfs -rmr /README.txt;

```
hive> dfs -rmr /README.txt;
Deleted hdfs://localhost:9000/README.txt
hive> dfs -ls /;
Found 5 items
drwxr-xr-x   - root supergroup          0 2014-11-19 23:54 /data_dir
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 23:58 /skt
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
drwxr-xr-x   - root supergroup          0 2014-11-13 20:06 /user
```

dfs -rmr 명령어로 /README.txt 파일을 삭제 시킨다.
dfs -ls 명령어를 이용하여 조회할 수 있다.

MEMO

❖

● 파일시스템 파일 내보내기

➢명령어 : dfs -get /data_dir/README.txt /home/root/;

```
hive> dfs -get /data_dir/README.txt /home/root/;
hive>
```

dfs -get 명령어를 이용하여 분산 파일 시스템에 README.txt 파일을 로컬 디렉토리 /home/root/ 에 생성한다.

```
[root@localhost root]# ls /home/root/
derby.log                  hadoop-1.0.4-preset.tar.gz  local_export
hadoop-1.0.4               hive-0.8.1                  metastore_db
hadoop-1.0.4-bin.tar.gz    hive-0.8.1.tar.gz           README.txt
[root@localhost root]#
```

새 터미널 창을 열어서 ls 명령어로 /home/root/ 디렉토리를 조회하면 README.txt 파일이 생성되어 있다.

MEMO

❖

● 하이브 테이블

➢테이블
○ RDBMS 테이블과 유사한 스키마 가짐
○ 파일 시스템의 파일을 테이블 데이터로 사용

테이블의 종류

○ 관리(managed) 테이블-기본
■ 하이브가 데이터 파일을 관리
■ 테이블 삭제시 메타 정보와 파일 함께 삭제

○ 외부(external) 테이블
■ 기존 파일을 테이블의 데이터로 사용
■ 테이블 삭제시 메타 정보만 삭제

MEMO

❖

● 하이브 테이블 생성하기

➢ 명령어 : create external table

```
hive> create table airplan(
    > year TINYINT,
    > month TINYINT,
    > dayofweek TINYINT,
    > dayofmonth INT,
    > deptime INT
    > )COMMENT 'delay airplan time'
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > LINES TERMINATED BY '\n';
OK
Time taken: 6.184 seconds
```

➢ 명령어 : show tables

```
hive> show tables;
OK
airplan
readme2
readme3
readme4
Time taken: 0.159 seconds
```

MEMO

❖

● 하이브 테이블 속성 확인

➢ 명령어 : describe airplan

```
hive> describe airplan;
OK
year    tinyint
month   tinyint
dayofweek       tinyint
dayofmonth      int
deptime int
Time taken: 0.194 seconds
```

➢ 명령어 : describe formatted airplan

테이블 속성은 formatted 옵션으로 좀 더 상세하게 확인할 수 있다.

MEMO

❖

● 하이브 테이블 속성 확인

```
hive> describe airplan;
OK
year    tinyint
month   tinyint
dayofweek       tinyint
dayofmonth      int
deptime int
Time taken: 0.194 seconds
hive> describe formatted  airplan;
OK
# col_name              data_type               comment

year                    tinyint                 None
month                   tinyint                 None
dayofweek               tinyint                 None
dayofmonth              int                     None
deptime                 int                     None

# Detailed Table Information
Database:               default
Owner:                  root
CreateTime:             Thu Nov 20 01:04:20 KST 2014
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://localhost:9000/user/hive/warehouse/airplan
Table Type:             MANAGED_TABLE
Table Parameters:
        comment                 delay airplan time
        transient_lastDdlTime   1416413060

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        line.delim              \n
        serialization.format    ,
Time taken: 0.123 seconds
hive>
```

MEMO

❖

● 하이브 데이터 불러오기

➢ 명령어 : load data inpath '/data_dir/README.txt' into table readme;

/data_dir 경로에 있는 README.txt 파일을 하이브 readme 테이블로 불러온다.

○ 디렉토리 명으로 로드하면 디렉토리에 포함된 모든 파일을 불러올수 있다.
○ LOCAL: 로컬 파일을 테이블 데이터 경로에 로딩
    ■ local이 없으면 HDFS 파일을 데이터 위치로 이동
○ OVERWRITE: 기존 파일들 삭제

```
hive> create table readme( cont string );
OK
Time taken: 0.118 seconds
hive> load data inpath '/data_dir/README.txt' into table readme;
Loading data to table default.readme
OK
Time taken: 0.21 seconds
```

MEMO

❖

● 하이브 데이터 조회

➢ 명령어 : select * from readme;

```
hive> select * from readme limit 10;
OK
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/core/

and our wiki, at:

   http://wiki.apache.org/hadoop/

This distribution includes cryptographic software.  The country in
which you currently reside may have restrictions on the import,
Time taken: 0.246 seconds
```

MEMO

❖

● 하이브 데이터 조회 결과 저장

➢ 명령어 : create table readmelimit
as select * from readme limit 15;

```
hive> create table readmelimit
    > as select * from readme limit 15;
Total MapReduce jobs = 1
Launching Job 1 out of 1
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 mapred.reduce.tasks=<number>
```

➢ 명령어 : insert overwrite local directory '/home/root/readmelimit'
select * from readme limit 15;

```
hive> insert overwrite local directory '/home/root/readmelimit'
    > select * from readme limit 15;
Total MapReduce jobs = 1
Launching Job 1 out of 1
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:
```

MEMO

❖

● 활용 예제 1

➢예제파일 : cite.TXT

➢파일위치 : /hive/cite/

➢예제 명 : WordCount

Variable Name	Variable type	Characters	Contents
citing	numeric	7	Citing Patent Number
cited	numeric	7	Cited Patent Number

➢root@localhost] hadoop dfs -mkdir /hive/cite
➢root@localhost] hadoop dfs -put ./cite.TXT /hive/cite/

mkdir 명령어를 실행하여 하둡에 /hive/cite라는 실습 폴더를 생성하고 put 명령어를 실행하여 서버에 위치한 cite.TXT 파일을 하둡에 생성한 /hive/cite/ 실습 폴더로 옮긴다.

MEMO

❖

● 활용 예제 1

➢테이블 생성

```
create external table cite (
citing int , cited int
)row format delimited fields terminated by ',' lines terminated by '₩n'
stored as textfile;
```

➢데이터 불러오기

```
load data inpath '/hive/cite/cite.TXT' overwrite into table cite;
```

➢데이터 처리 예제

```
select citing, count(citing) from cite group by citing limit 100;

select citing, cited, count(*) from cite group by citing, cited limit 100;
```

MEMO

❖

● 활용 예제 2

➢예제파일 : apat.TXT

➢파일위치 : /hive/apat/

➢예제 명 : MyJob

Variable Name	Variable type	Characters	Contents
patent	numeric	7	Patent Number
gyear	numeric	12	Grant Year
gdate	numeric	12	Grant Date
appyear	numeric	12	Application Year
country	Character	3	Country of First Inventor
postate	numeric	3	State of First Inventor (if US)
assignee	numeric	12	Assignee Identifier (missing 1963-1967)
asscode	numeric	12	Assignee Type
claims	numeric	12	number of Claims
nclass	numeric	12	Main Patent Class (3 digit)

➢root@localhost] hadoop dfs -mkdir /hive/apat
➢root@localhost] hadoop dfs -put ./apat.TXT /hive/apat/

mkdir 명령어를 실행하여 하둡에 /hive/apat라는 실습 폴더를 생성하고 put 명령어를 실행하여 서버에 위치한 apat.TXT 파일을 하둡에 생성한 /hive/apat/ 실습 폴더로 옮긴다.

● 활용 예제 2

➢테이블 생성

```
create external table apat (
patent int , gyear int, gdate int, appyear int, country string, postate
int, assignee int, asscode int, claims int, nclass int
)row format delimited fields terminated by ',' lines terminated by '₩n'
stored as textfile;
```

➢데이터 불러오기

```
load data inpath '/hive/apat/apat.TXT' overwrite into table apat;
```

➢데이터 처리 예제

```
select country, sum(cnt) from (
                select country, patent , count(*) as cnt from apat group by
country, patent
) a
group by country limit 100;
```

MEMO

❖

● 활용 예제 3

➢테이블 생성

```
create external table cite2 (
citing int , cited int
)row format delimited fields terminated by ',' lines terminated by '₩n'
stored as textfile;
```

➢데이터 인설트

```
insert overwrite table cite2
select * from cite where citing = 3858241;
```

➢데이터 조회

```
select * from cite2;
```

MEMO

❖

● 활용 예제 3

➢테이블 생성

```
create external table cite3(
citing int , cited int
)row format delimited fields terminated by ',' lines terminated by '₩n'
stored as textfile;


create external table cite4(
citing int , cited int
)row format delimited fields terminated by ',' lines terminated by '₩n'
stored as textfile;
```

➢데이터 멀티 인설트

```
from cite

insert overwrite table cite3
select *
where citing = 3858246

insert overwrite table cite4
select *
where citing = 3858245;
```

➢데이터 조회

```
select * from cite3;
select * from cite4;
```

● 활용 예제 4

➢데이터 조회 결과 합쳐서 보기

```
select * from (
select * from cite2
Union all
select * from cite3
Union all
select * from cite4
) citeall
```

MEMO

❖

● 파티셔닝 실습

➢테이블 생성

```
create external table cite2 (
citing int , cited int
)
PARTITIONED BY ( dt string )
row format delimited fields terminated by ',' lines terminated by '₩n'
stored as textfile;
```

➢데이터 멀티 인설트

```
from cite

insert overwrite table cite2 partition (dt=3858246)
select *
where citing = 3858246

insert overwrite table cite2 partition (dt=3858245)
select *
where citing = 3858245;
```

MEMO

❖

Chapter

03 아파치 하이브 활용 실습2

● LOCATION DIRECTORY

➢테이블 생성

```
create table airplan(
year TINYINT,
month TINYINT,
dayofweek TINYINT,
dayofmonth INT,
deptime INT
)COMMENT 'delay airplan time'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '₩n';
```

```
create table airplan2(
year TINYINT,
month TINYINT,
dayofweek TINYINT,
dayofmonth INT,
deptime INT
)COMMENT 'delay airplan time'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '₩n'
LOCATION '/air/data/';
```

MEMO

❖

● LOCATION DIRECTORY

```
hive> create table airplan2(
    > year TINYINT,
    > month TINYINT,
    > dayofweek TINYINT,
    > dayofmonth INT,
    > deptime INT
    > )COMMENT 'delay airplan time'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > LINES TERMINATED BY '\n'
    > LOCATION '/air/data/';
OK
Time taken: 0.13 seconds
```

➢ dfs -ls /;

dfs -ls 명령어를 통하여 LOCATION에 지정된 디렉토리가 파일시스템에 생성된 것을 확인한다.

```
hive> dfs -ls /;
Found 6 items
drwxr-xr-x   - root supergroup          0 2014-11-20 04:36 /air
drwxr-xr-x   - root supergroup          0 2014-11-20 01:44 /data_dir
drwxr-xr-x   - root supergroup          0 2014-11-13 11:29 /readme3
drwxr-xr-x   - root supergroup          0 2014-11-19 23:58 /skt
drwxr-xr-x   - root supergroup          0 2014-11-19 09:40 /tmp
drwxr-xr-x   - root supergroup          0 2014-11-13 20:06 /user
```

● SELECT

➢select year,month,dayofmonth,deptime from airplan

➢select * from airplan

➢select * from airplan limit 1000

➢select * from airplan where year= '2007'

➢select year,month avg(deptime) as avg
from airplan where year = '2007'
group by year,month
having avg(res_time) > 20

● 중첩 SELECT

➢형식1:
○ select e.deptime from
(select year, month, avg(deptime) as deptime
from airplan where year = '2008' group by year,month) e
where e.month > 6;

➢형식2:
○ from (
select year,month, avg(deptime) as deptime
from airplan where year = 2008' group by year,month) e
select e. deptime where e.month > 6;

● 내부 JOIN

➢내부 조인
○ select a.year, a.month, a.dayofmonth, a.deptime, b.kor_name
from airplan a JOIN dairport b
ON a.dest = b.port_code
where a.month >5

➢n개 이상 조인 가능
○ select … from airplan a
join airport b on a.dest = b.port_code
join airport c on a.dest = c.port_code

on 절의 제약
■ 동등 조인만 가능 (a.dest = b.port_code)
● 비동등 조인은 사용 불가 (a.dest>= b.port_code)
■ 각 조건의 연결은 and 만 사용 가능

MEMO

❖

● 외부 JOIN

➢외부 조인
○ select a.year, a.month, a.deptime, b.kor_name
from airplan a LEFT OUTER JOIN airport b
ON a.dest = b.port_code
where a.month between 5 and 10

○ 주의: 조인 후 where 평가
■ 외부 조인 결과가 null로 표시되어야 하는 컬럼에 대해 where 절에서 검사하면 내부 조인처럼 됨

○ LEFT, RIGHT, FULL 모두 지원

MEMO

❖

● Order, Sort

➢select a.year, a.month, a.deptime
from airplan a
sort by a.year, a.month

➢select a.year, a.month, a.deptime
from airplan a
order by a.year, a.month

Order by : 전체 데이터 정렬(1개 리듀스)
Sort by : 리듀스 별로 정렬
○ 부분 데이터에 대한 정렬
○ distribute by : 리듀스로 보낼 데이터 분류 기준

● Distribute, Cluster

➢select a.year, a.month, a.deptime
from airplan a
distribute by a.month
sort by a.year, a.month

distribute by : 리듀스로 보낼 데이터 분류 기준
cluster by
○ 지정 컬럼의 동일 값을 가진 행을 동일 리듀서로 보냄
■ 지정 컬럼에 대한 정렬 (리듀서의 키로 사용되니까)

● Union All

```
select data.year, data.month, data.deptime
      From (
      select a1.year, a1.month, a1.deptime from airplan a1
      union all
      select a1.year, a1.month, a1.deptime from airplan a2
      ) data
      order by data.year, data.month asc
```

● Insert

➢from airplan a

```
      insert overwrite table air2
      select * where a.month = 1
      insert overwrite table air3
      select * where a.month = 2
      insert into table a4
      select * where a.month between 5 and 10;
```

➢create table air_test

```
     as select * from airplan
     where year=2007 and month=10;
```

● Insert

```
➢insert overwrite local directory '/home/root/dnfile‘
      ROW FORMAT DELIMITED FIELDS TERMINATED BY '₩t'
      select * from airplan where year = 2007 and month = 2;
```

- local이 없으면 HDFS 사용
- row format이 없으면 테이블의 포맷 사용
- 지정 디렉토리에 ‘000000_0’ 등의 이름으로 파일 생성

MEMO

❖

● 활용 예제

➢Review.txt 파일을 활용하여 다음 예제의 결과를 확인 하시오.

➢Review.txt 파일을 활용 가능하도록 테이블을 생성 하고 데이터를 업데이트 시키시오.

Review.txt 파일은 맴버아이디, 상품아이디, 날짜, 유용한 답변, 답변 수, 조회, 제목, 내용로 구성되어 있다.
필드는 ₩t로 구분되어 있다.

➢예제파일 : review.txt

➢파일위치 : /hive/review/

MEMO

❖

● 활용 예제

➢Review.txt 파일을 활용 가능하도록 테이블을 생성 하고 데이터를 업데이트 시키시오.

Review.txt 파일은 맴버아이디, 상품아이디, 날짜, 유용한 답변, 답변 수, 조회, 제목, 내용로 구성되어 있다.
필드는 ₩t로 구분되어 있다.

➢Create external table reviewnew (member_id string, product_id string, date_info string, num_of_helpful_fb int, num_of_fb int, rating int, title string, body string) row format delimited fields terminated by '₩t' lines terminated by '₩n'
stored as textfile;

➢load data local inpath '/home/root/data/review.txt' overwrite into table reviewnew;

MEMO

❖

● 활용 예제

➢1. 리뷰작성자의 영향력(작성한 전체 리뷰가 받은 유용한 피드백/전체 피드백) 100건 출력하기

➢select member_id, sum(num_of_helpful_fb) /sum(num_of_fb) from reviewnew group by member_id limit 100;

➢2. 리뷰어의 리뷰 경력

➢select member_id, datediff(max(to_date(substr(FROM_UNIXTIME(UNIX_TIMESTAMP(date_info, "MMM dd, yyyy")),1,10))),min(to_date(substr(FROM_UNIXTIME(UNIX_TIMESTAMP(date_info, "MMM dd, yyyy")),1,10)))) from reviewnew group by member_id limit 100;

➢3. 리뷰어가 리뷰한 제품의 개수

➢select member_id, count(distinct product_id) from reviewnew group by member_id limit 100;

● 활용 예제

➢4. 리뷰의 길이

➢select member_id, product_id, length(body) from reviewnew limit 100;

➢5. 리뷰어가 동일책에 리뷰했는지 여부

➢select member_id, (case when count(distinct product_id)>1 then 1 else 0 end) as dup from reviewnew group by member_id limit 100;

MEMO

❖

● 활용 예제 2

➢Create external table booksinfo(row_id string, product_id string, publisher string, release_date string, product_demension string, shipping_weight string, language string, num_pages int, type string, edition string, fulldesc string) row format delimited fields terminated by '₩t' lines terminated by '₩n'
stored as textfile;

➢load data local inpath '/home/root/data/BooksInfo.txt' overwrite into table booksinfo;

➢Create external table member_location(user_name string, member_rank string, birth_day string, loc string, name string, member_id string, state string) row format delimited fields terminated by '₩t' lines terminated by '₩n'
stored as textfile;

➢load data local inpath '/home/root/data/locations.txt' overwrite into table member_location;

MEMO

❖

● 활용 예제 2

➢리뷰어가 받은 전체 피드백의 수, 전체 유용한 피드백의 수를 구해서 파일로 받기

➢INSERT OVERWRITE LOCAL DIRECTORY '/home/root/data/fd1'
select member_id,sum(num_of_helpful_fb),
sum(num_of_helpful_fb)/sum(num_of_fb),
datediff(max(to_date(substr(FROM_UNIXTIME(UNIX_TIMESTAMP(date_info, "MMM dd, yyyy")),1,10))),
min(to_date(substr(FROM_UNIXTIME(UNIX_TIMESTAMP(date_info, "MMM dd, yyyy")),1,10)))),
count(distinct reviewnew.product_id), avg(rating), avg(length(body)),
(case when count(distinct reviewnew.product_id)>1 then 1 else 0 end) as dup,
min(datediff(to_date(substr(FROM_UNIXTIME(UNIX_TIMESTAMP(date_info, "MMM dd, yyyy")),1,10)),
to_date(release_date))) from reviewnew
INNER JOIN booksinfo on(reviewnew.product_id=booksinfo.product_id)
group by member_id;

MEMO

❖

● 활용 예제 2

➢reviewer id-book id-rating-feedback-useful feedback 추출

➢Create external table fb2(member_id string, product_id string, rating string, num_of_fb string, num_of_helpful_fb string)
row format delimited fields terminated by '-' lines terminated by '₩n'
stored as textfile location '/fb2';

```
INSERT INTO TABLE fb2
Select reviewnew .member_id, reviewnew . product_id, rating,
num_of_fb, num_of_helpful_fb
from reviewnew
INNER JOIN booksinfo
on(reviewnew.product_id=booksinfo.product_id) ;

dfs -copyToLocal /fb2/000000_0 /home/root/data/fb2/

Ex> dfs -copyToLocal /home/root/data/fb2/*
/home/root/data/fb2_data;
```

MEMO

❖

● 활용 예제 2

➢reviewer id-지역(State)-book id-rating-feedback-useful feedback 추출

➢Create external table fb3(member_id string, state string, product_id string, rating string, num_of_fb string, num_of_helpful_fb string)
row format delimited fields terminated by '-' lines terminated by '₩n'
stored as textfile location '/fb3';

```
INSERT INTO TABLE fb3
Select fb2.member_id, member_location.state , fb2.product_id , rating,
num_of_fb, num_of_helpful_fb
from
fb2
INNER JOIN member_location on(fb2.member_id=member_location.
member_id);
```

➢dfs -copyToLocal /fb2/000000_0 /home/root/data/fb3/

➢Ex> dfs -copyToLocal /home/root/data/fb3/*
/home/root/data/fb3_data;

MEMO

❖

Chapter

04 아파치 하이브 문자열 함수

● ASCII(string str)

➢Example1: ASCII('hadoop') returns 104

➢Example2: ASCII('A') returns 65

➢select ascii(str) from readme;

■ 문자열의 첫 번째 문자를 ASCII 값으로 변환

```
hive> select ascii(str) from readme;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411250732_0001, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411250732_0001
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411250732_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-11-25 07:33:28,128 Stage-1 map = 0%,  reduce = 0%
```

●CONCAT(string str1, string str2...)

➢Example: CONCAT('hadoop','-','hive') returns 'hadoop-hive'

➢select concat(str,'-',str) from readme limit 1;

■ 모든 문자열의 연결 함수

```
hive> select concat(str,'-',str) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411250732_0002, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411250732_0002
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411250732_0002
```

● CONCAT_WS(string delimiter, string str1, string str2...)

➢Example: CONCAT_WS('-','hadoop','hive') returns 'hadoop-hive'

➢select concat_ws('*','1','2') from readme limit 1;

■ CONCAT 함수와 유사하다 단, CONCAT_WS는 공통된 구분자 표현에 유리하다.

```
hive> select concat_ws('*','1','2') from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411250732_0004, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411250732_0004
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411250732_0004
```

● FIND_IN_SET(string search_string, string source_string_list)

➢Example: FIND_IN_SET('ha','hao,mn,hc,ha,hef') returns 4

➢select find_in_set('a','a,b,c,d,e') from readme limit 1;

■ 문자열 배열중에서 해당 문자가 포함된 순서를 나타냅니다.
■ 문자열 배열은 반드시 , 쉼표로 구분되어야 합니다.

```
hive> select   find_in_set('a','a,b,c,d,e') from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411250732_0009, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411250732_0009
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411250732_0009
```

● LENGTH(string str)

➢Example: LENGTH('hive') returns 4

➢select length(str) from readme limit 1;

■ 문자열의 문자수를 반환합니다.

```
hive> select   length(str) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411250732_0010, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411250732_0010
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411250732_0010
```

● LOWER(string str), LCASE(string str)

➢Example: LOWER('HiVe') returns 'hive'

➢select lower(str) from readme limit 1;

➢select lcase(str) from readme limit 1;

■ 문자열의 모든 문자를 소문자로 변환합니다.

```
hive> select   lcase(str) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411250732_0011, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411250732_0011
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job 201411250732 0011
```

● LPAD(string str, int len, string pad)

➢Example: LPAD('hive',6,'v') returns 'vvhive'

➢select lpad('hive',10,'*') from readme limit 1;

■ 지정한 문자열 길이만큼 왼쪽에 문자를 배치함

```
hive> select  lpad('hive',10,'*') from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0001, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0001
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0001
```

● LTRIM(string str)

➢Example: LTRIM(' hive') returns 'hive'

➢select ltrim(' hive') from readme limit 1;

■ 문자열의 왼쪽 공백을 제거합니다.

```
hive> select   ltrim('    hive')  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0002, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0002
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0002
```

● REPEAT(string str, int n)

➢Example: REPEAT('hive',2) returns 'hivehive'

➢select repeat('hive',2) from readme limit 1;

■ 지정한 문자열을 지정한 횟수만큼 반복 합니다.

```
hive> select  repeat('hive',2)  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0003, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0003
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0003
```

● RPAD(string str, int len, string pad)

➢Example: RPAD('hive',6,'v') returns 'hivevv'

➢select rpad('hive',6,'*') from readme limit 1;

■ 지정한 문자열 길이만큼 오른쪽에 문자를 배치함

```
hive> select   rpad('hive',6,'*')  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0004, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0004
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job 201411251950 0004
```

● REVERSE(string str)

➢Example: REVERSE('hive') returns 'evih'

➢select reverse('hive') from readme limit 1;

■ 반전된 문자열을 반환합니다.

```
hive> select  reverse('hive')  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0005, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0005
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job 201411251950 0005
```

● RTRIM(string str)

➢Example: RTRIM('hive ') returns 'hive'

➢select rtrim('hive ') from readme limit 1;

■ 문자열의 오른쪽 공백을 제거합니다.

```
hive> select   rtrim('hive     ')  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0006, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0006
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0006
```

● SPACE(int number_of_spaces)

➢Example: SPACE(4) returns ' '

➢select 'hive',space(4),'hive' from readme limit 1;

■ 문자열 사이에 공백을 입력합니다.

```
hive> select  space(4)  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0007, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0007
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0007
```

● SPLIT(string str, string pat)

➢Example: SPLIT('hive:hadoop',':') returns ["hive","hadoop"]

➢select split('hive:hadoop',':') from readme limit 1;

■ 문자열을 지정된 구분자로 분할하여 배열로 변환한다.

```
hive> select   split('hive:hadoop',':')  from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0009, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0009
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0009
```

● SUBSTR(string source_str, int start_position [,int length])

➢Example1: SUBSTR('hadoop',4) returns 'oop'

➢Example2: SUBSTR('hadoop',4,2) returns 'oo'

➢Select substr(str, 4, 5) from readme limit 1;

■ 지정한 문자열 위치에서 문자열 길이만큼 반환합니다.

```
hive> Select  substr(a, 4, 5) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0010, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0010
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0010
```

● TRIM(string str)

➢Example: TRIM(' hive ') returns 'hive'

➢select trim(' hive ') from readme limit 1;

■ 문자열의 공백을 제거하여 반환합니다.

```
hive> select   trim('   hive   ') from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0011, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0011
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0011
```

● UPPER(string str), UCASE(string str)

➢Example: UPPER('HiVe') returns 'HIVE‘

➢Select upper(str) from readme limit 1;

■ 문자열의 모든 문자를 대문자로 변환합니다.

```
hive> Select  upper(a) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0012, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0012
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0012
```

MEMO

❖

Chapter

05 아파치 하이브 조건 함수

● IF(Test Condition, True Value, False Value)

➢Example: IF(1=1, 'working', 'not working') returns 'working'

➢Select if(length(str)>10, ‘long’, ‘not long’) from readme limit 1;

■ 지정한 문자열 길이가 10보다 큰 경우 long을 반환합니다.
■ 아닌 경우, not long을 반환합니다.

```
hive> Select  if(length(a)>10, 'long', 'not long') from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0013, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0013
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0013
```

● COALESCE(value1,value2,...)

➢Example: COALESCE(NULL,NULL,5,NULL,4) returns 5

➢select coalesce(null,5) from readme limit 1;

■ null이 아닌 첫번째 컬럼의 데이터를 반환합니다.

```
hive> select   coalesce(null,5) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0014, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0014
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job 201411251950 0014
```

● CASE Statement

➢CASE [expression]
WHEN condition1 THEN result1
WHEN condition2 THEN result2
ELSE result
END

Example:

```
CASE    Fruit
        WHEN 'APPLE' THEN 'The owner is APPLE'
        WHEN 'ORANGE' THEN 'The owner is ORANGE'
        ELSE 'It is another Fruit'
END
```

The other form of CASE is

```
CASE
        WHEN Fruit = 'APPLE' THEN 'The owner is APPLE'
        WHEN Fruit = 'ORANGE' THEN 'The owner is ORANGE'
        ELSE 'It is another Fruit'

END
```

```
hive> select case length(a)
    > when 1 then 'a'
    > when 2 then 'b'
    > else 'c'
    > end
    > from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0017, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0017
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job 201411251950 0017
```

Chapter

06 아파치 하이브 날짜 함수

● DATEDIFF(string date1, string date2)

➢Example: DATEDIFF('2000-03-01', '2000-01-10') returns 51

➢Select datediff('2000-03-01', '2000-01-10') from readme limit 1;

■ 날짜간의 간격 일수를 계산합니다.

```
hive> Select  datediff('2000-03-01', '2000-01-10') from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0018, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0018
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job 201411251950 0018
```

● DATE_ADD(string date, int days)

➢Example: DATE_ADD('2000-03-01', 5) returns '2000-03-06'

➢select date_add('2000-03-01', 5) from readme limit 1;

■ 변경된 이후 날짜를 계산합니다.

```
hive> select  date_add('2000-03-01', 5) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0019, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0019
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0019
```

● DATE_SUB(string date, int days)

➢Example: DATE_SUB('2000-03-01', 5) returns '2000-02-25'

➢Select date_sub('2000-03-01', 5) from readme limit 1;

■ 변경된 이전 날짜를 계산합니다.

```
hive> Select  date_sub('2000-03-01', 5) from readme limit 1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201411251950_0020, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201411251950_0020
Kill Command = /home/root/hadoop-1.0.4/libexec/../bin/hadoop job  -Dmapred.job.t
racker=localhost:9001 -kill job_201411251950_0020
```

MEMO

❖

● YEAR(string date)

➢Example: YEAR('2000-01-01 10:20:30') returns 2000

● MONTH(string date)

➢Example: YEAR('2000-01-01 10:20:30') returns 1

● DAY(string date), DAYOFMONTH(date)

➢Example: DAY('2000-03-01 10:20:30') returns 1

MEMO

❖

● HOUR(string date)

➢Example: HOUR('2000-03-01 10:20:30') returns 10

● MINUTE(string date)

➢Example: MINUTE('2000-03-01 10:20:30') returns 20

● SECOND(string date)

➢Example: SECOND('2000-03-01 10:20:30') returns 30

MEMO

❖