4일차. 데이터 적재 서비스 실습 - Hive

> *아파치 하이브를 통해 다양한 데이터 웨어하우스 예제를 실습합니다*

- 목차

- 참고 자료

  - Hive Language Manual DDL
  - Hive Language Manual DML
  - Top 7 Hive DDL Commands
  - Top 7 Hive DML Commands
  - IMDB data from 2006 to 2016
  - Hive update, delete ERROR

1 하이브 서비스 기동

> *스파크 실습을 위한 도커 컨테이너를 기동합니다*

- 최신 소스를 내려 받습니다

```
bash>
cd /home/ubuntu/work/data-engineer-intermediate-training
git pull
```

- 모든 컨테이너를 종료하고, 더 이상 사용하지 않는 도커 이미지 및 볼륨을 제거합니다

```
bash>
docker rm -f `docker ps -aq`
docker image prune
docker volume prune
```

- 스파크 워크스페이스로 이동하여 도커를 기동합니다

```
bash>
cd /home/ubuntu/work/data-engineer-intermediate-training/day4
```

docker-compose up -d docker-compose ps

```
* 실습에 필요한 IMDB 데이터를 컨테이너로 복사합니다
```bash
```
```

```
bash>
docker cp data/imdb.tsv day4_hive-server_1:/opt/hive/examples/imdb.tsv
docker-compose logs -f hive-server
```

2 하이브 기본 명령어 가이드

2-1 하이브 데이터베이스 DDL 가이드

1. CREATE

> *데이터베이스를 생성합니다*

```
bash>
docker-compose exec hive-server bash
beeline

beeline>
!connect jdbc:hive2://localhost:10000 scott tiger
```

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
[COMMENT database_comment]
[LOCATION hdfs_path]
[WITH DBPROPERTIES (property_name=property_value, ...)];

beeline>
create database if not exists testdb comment 'test database' location
'/user/hive/warehouse/testdb' with dbproperties ('createdBy' = 'psyoblade');
```

2. SHOW

> *데이터베이스 목록을 출력합니다*

```
SHOW (DATABASES|SCHEMAS);

beeline>
show databases;
```

3. DESCRIBE

> *데이터베이스 정보를 출력합니다*

```
DESCRIBE DATABASE/SCHEMA [EXTENDED] db_name;

beeline>
describe database testdb;
```

4. USE

> *해당 데이터베이스를 사용합니다*

```
USE database_name;

beeline>
use testdb;
```

5. DROP

> *데이터베이스를 삭제합니다 (default:RESTRICT) 테이블이 존재하는 경우 오류가 발생하며, CACADE 옵션을 주는 경우 모든 테이블까지 삭제됩니다*

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];

beeline>
drop database testdb;
show databases;
```

6. ALTER

> *데이터베이스의 정보를 변경합니다*

- DBPROPERTIES

  ```
  ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES
  (property_name=property_value, ...);
  ```

beeline> create database if not exists testdb comment 'test database' location '/user/hive/warehouse/testdb' with dbproperties ('createdBy' = 'psyoblade'); alter database testdb set dbproperties ('createdfor'='park.suhyuk'); describe database extended testdb;

```
* OWNER
```sql
ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;

beeline>
alter database testdb set owner role admin;
describe database extended testdb;
```

2-2 하이브 테이블 DDL 가이드

1. CREATE

> *테이블을 생성합니다*

```
CREATE TABLE [IF NOT EXISTS] [db_name.] table_name [(col_name data_type [COMMENT
col_comment], ... [COMMENT col_comment])]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
[LOCATION hdfs_path];

beeline>
create table if not exists employee (emp_id string comment 'employee id',
emp_name string comment 'employee name',
emp_salary bigint comment 'employee salary')
comment 'test employee table'
row format delimited
fields terminated by ','
stored as textfile;
```
```

2. SHOW

> *테이블 목록을 조회합니다*

```
SHOW TABLES [IN database_name];

beeline>
show tables;
```

3. DESCRIBE

> *테이블 정보를 조회합니다*

```
DESCRIBE [EXTENDED|FORMATTED] [db_name.] table_name[.col_name ( [.field_name])];

beeline>
describe employee;
```

4. DROP

> *테이블을 삭제합니다. 일반 DROP 의 경우 .Trash/current directory 경로로 이동하지만 PURGE 옵션을 주는 경우 즉시 삭제됩니다*

```
DROP TABLE [IF EXISTS] table_name [PURGE];

beeline>
drop table if exists employee purge;
show tables;
```

5. ALTER

> *테이블을 변경합니다*

- RENAME

  ```
  ALTER TABLE table_name RENAME TO new_table_name;
  ```

beeline> create table if not exists employee (emp_id string comment 'employee id', emp_name string comment 'employee name', emp_salary bigint comment 'employee salary') comment 'test employee table' row format delimited fields terminated by ',' stored as textfile;

alter table employee rename to renamed_emp; show tables;

```
* ADD COLUMNS
```sql
ALTER TABLE table_name ADD COLUMNS (column1, column2) ;

beeline>
create table if not exists employee (emp_id string comment 'employee id')
comment 'test employee table'
row format delimited
fields terminated by ','
stored as textfile;
```
```

```
alter table employee add columns (emp_name string comment 'employee name',
emp_salary bigint comment 'employee salary');

desc employee;
desc renamed_emp;
```

6. TRUNCATE

> *테이블의 데이터만 제거합니다*

```
TRUNCATE TABLE table_name;

beeline>
use testdb;
insert into renamed_emp values (1, 'suhyuk', 1000);
select count(1) from renamed_emp;
+--------+
|  _c0   |
+--------+
| 1      |
+--------+

truncate table renamed_emp;
select count(1) from renamed_emp;
+-------+
| _c0   |
+-------+
| 0     |
+-------+
```

2-3 하이브 DML 가이드

1. LOAD

> *로컬(LOCAL) 혹은 클러스터 저장된 데이터를 하둡 클러스터에 업로드(Managed) 혹은 링크(External) 합니다*

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION
(partcol1=val1, partcol2=val2 ...)];

beeline>
drop table if exists imdb_movies;
create table imdb_movies (rank int, title string, genre string, description string,
director string, actors string, year string, runtime int, rating string, votes int,
revenue string, metascore int) row format delimited fields terminated by '\t';

load data local inpath '/opt/hive/examples/imdb.tsv' into table imdb_movies;
```

2. SELECT

> *테이블에 저장된 레코드를 SQL 구문을 통해서 조회합니다*

```
SELECT col1,col2 FROM tablename;
```

```
beeline>
select rank, title, genre from imdb_movies limit 5;
```

3. INSERT

> *테이블에 읽어온 레코드 혹은 생성된 레코드를 저장합니다*

- INSERT INTO

  ```
  INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
  select_statement1 FROM from_statement;
  ```

beeline> create table if not exists imdb_title (title string); insert into table imdb_title select title from imdb_movies; select title from imdb_title limit 5;

```
* INSERT OVERWRTIE
```sql
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, ..) [IF NOT EXISTS]]
select_statement FROM from_statement;

beeline>
create table if not exists imdb_title (title string);
insert overwrite table imdb_title select description from imdb_movies;
select title from imdb_title limit 5;
```

- INSERT VALUES

  ```
  INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)]
  VALUES values_row [, values_row ...];
  ```

beeline> insert into imdb_title values ('1 my first hive table record'), ('2 my second records'), ('3 third records'); select title from imdb_title where title like '%record%';

```

#### 4. DELETE
> 테이블에 저장된 데이터를 삭제합니다
* 현재 ACID-based transaction 을 지원하는 것은 Bucketed ORC 파일만 지원합니다
  * [Hive Transactions]
(https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions)
```sql
DELETE FROM tablename [WHERE expression]

beeline>
create table imdb_orc (rank int, title string) clustered by (rank) into 4 buckets
stored as orc tblproperties ('transactional'='true');

// 아래와 같이 동시성 및 버킷팅 설정이 제대로 되어 있어서 ACID 트랜젝션 수행이 가능합니다
set hive.support.concurrency=true;
set hive.enforce.bucketing=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
set hive.compactor.initiator.on=true;
set hive.compactor.worker.threads=1;
```

```
insert into table imdb_orc values (1, 'psyoblade'), (2, 'psyoblade suhyuk');
delete from imdb_orc where rank = 1;

select * from imdb_orc;
```

- 제대로 설정되지 않은 경우 아래와 같은 오류를 발생시킵니다

```
delete from imdb_orc where rank = 1;
Error: Error while compiling statement: FAILED: SemanticException [Error
10294]: Attempt to do update or delete using transaction manager that does not
support these operations. (state=42000,code=10294)
```

5. UPDATE

> *대상 테이블의 컬럼을 업데이트 합니다. 단, 파티셔닝 혹은 버킷팅 컬럼은 업데이트 할 수 없습니다*

```
UPDATE tablename SET column = value [, column = value ...] [WHERE expression];

beeline>
update imdb_orc set title = 'psyoblade title';
select * from imdb_orc;
```

6. EXPORT

> *테이블 메타데이터(_metadata)와 데이터(data) 정보를 HDFS 경로에 백업 합니다*

```
EXPORT TABLE tablename [PARTITION (part_column="value"[, ...])] TO
'export_target_path' [ FOR replication('eventid') ];

beeline>
export table imdb_orc to '/user/ubuntu/archive/imdb_orc';
```

- 익스포트 된 결과를 확인합니다 ```bash bash> docker-compose exec hive-server bash hadoop fs -ls /user/ubuntu/archive/imdb_orc
- rwxr-xr-x 3 root supergroup 1244 2020-08-23 14:17 /user/ubuntu/archive/imdb_orc/_metadata drwxr-xr-x - root supergroup 0 2020-08-23 14:17 /user/ubuntu/archive/imdb_orc/data

7. IMPORT

> *백업된 데이터로 새로운 테이블을 생성합니다*

```
IMPORT [[EXTERNAL] TABLE new_or_original_tablename [PARTITION (part_column="value"[,
...])]] FROM 'source_path' [LOCATION 'import_target_path'];

beeline>
import table imdb_orc_imported from '/user/ubuntu/archive/imdb_orc';
select * from imdb_orc_imported;
+---------------------------+-----------------------------+
| imdb_title_imported.rank  | imdb_title_imported.title   |
+---------------------------+-----------------------------+
```

```
| 2                          | psyoblade title             |
+----------------------------+-----------------------------+
```

3 하이브 트러블슈팅 가이드

> *IMDB 영화 예제를 통해 테이블을 생성하고, 다양한 성능 개선 방법을 시도해보면서 왜 그리고 얼마나 성능에 영향을 미치는 지 파악합니다*

3-1 파티셔닝을 통한 성능 개선

1.1. 하이브 서버로 접속합니다

- 하이브 터미널을 통해 JDBC Client 로 하이브 서버에 접속합니다

```
bash>
docker-compose exec hive-server bash
```

beeline> !connect jdbc:hive2://localhost:10000 scott tiger use testdb;

```
#### 1.2. 데이터집합의 스키마를 확인하고 하이브 테이블을 생성합니다
* 데이터집합은 10년(2006 ~ 2016)의 가장 인기있는 1,000개의 영화에 대한 데이터셋입니다

| 필드명 | 설명 |
| --- | --- |
| Title | 제목 |
| Genre | 장르 |
| Description | 설명 |
| Director | 감독 |
| Actors | 배우 |
| Year | 년도 |
| Runtime | 상영시간 |
| Rating | 등급 |
| Votes | 투표 |
| Revenue | 매출 |
| Metascrore | 메타스코어 |

* 테이블 생성 및 조회를 합니다
  - Q1. 년도 별 개봉된 영화의 수를 년도 오름차순으로 출력하시오
  - Q2. 2015년도 개봉된 영화 중에서 최고 매출 Top 3 영화 제목과 매출금액을 출력하시오
```sql
beeline>
drop table if exists imdb_movies;

create table imdb_movies (rank int, title string, genre string, description string,
director string, actors string, year string, runtime int, rating string, votes int,
revenue string, metascore int) row format delimited fields terminated by '\t';

load data local inpath '/opt/hive/examples/imdb.tsv' into table imdb_movies;
select * from imdb_movies limit 10;
```

- 기존 테이블을 이용하여 파티션 구성된 테이블을 생성합니다
  - 다이나믹 파티션은 마지막 SELECT 절 컬럼을 사용합니다
```

```
beeline>
drop table if exists imdb_partitioned;
```

create table imdb_partitioned (rank int, title string, genre string, description string, director string, actors string, runtime int, rating string, votes int, revenue string, metascore int) partitioned by (year string) row format delimited fields terminated by '\t';

set hive.exec.dynamic.partition=true; set hive.exec.dynamic.partition.mode=nonstrict; insert overwrite table imdb_partitioned partition (year) select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore, year from imdb_movies;

select year, count(1) as cnt from imdb_partitioned group by year;

```
* 2가지 테이블 조회 시의 차이점을 비교합니다
```sql
beeline>
explain select year, count(1) as cnt from imdb_movies group by year;
explain select year, count(1) as cnt from imdb_partitioned group by year;
```

- **일반 테이블과, 파티셔닝 테이블의 성능을 비교합니다**

```
cd /home/ubuntu/work/data-engineer-intermediate-training/day4/ex1
vimdiff agg.imdb_movies.out agg.imdb_partitioned.out
```

- 관련 링크
  - Hive Language Manul DML

3-2 파일포맷 변경을 통한 성능 개선

> *파일포맷을 텍스트 대신 파케이 포맷으로 변경하는 방법을 익히고, 예상한 대로 결과가 나오는지 확인합니다*

- 파케이 포맷 기반의 테이블을 CTAS (Create Table As Select) 통해 생성합니다 (단, CTAS 는 SELECT 절을 명시하지 않습니다)

```
beeline>
drop table if exists imdb_parquet;
create table imdb_parquet row format delimited stored as parquet as select *
from imdb_movies;
select year, count(1) as cnt from imdb_parquet group by year;
```

- **파티셔닝 테이블과 파케이 포맷 테이블의 성능을 비교합니다**

```
cd /home/ubuntu/work/data-engineer-intermediate-training/day4/ex1
vimdiff agg.imdb_partitioned.out agg.imdb_parquet.out
```

- 텍스트, 파티션 및 파케이 테이블의 조회시에 어떤 차이점이 있는지 확인해 봅니다.

```
beeline>
explain select year, count(1) as cnt from imdb_movies group by year;
# Statistics: Num rows: 3096 Data size: 309656 Basic stats: COMPLETE Column
stats: NONE
```

explain select year, count(1) as cnt from imdb_partitioned group by year;
```

Statistics: Num rows: 1000 Data size: 302786 Basic stats: COMPLETE Column stats: NONE

explain select year, count(1) as cnt from imdb_parquet group by year;

Statistics: Num rows: 1000 Data size: 12000 Basic stats: COMPLETE Column stats: NONE

create table imdb_parquet_sorted stored as parquet as select title, rank, metascore, year from imdb_movies sort by metascore;

```
* 파케이 파일 테이블의 경우에도 필요한 컬럼만 유지하는 것이 효과가 있는지 확인해봅니다
```sql
beeline>
create table imdb_parquet_small stored as parquet as select title, rank, metascore,
year from imdb_movies sort by metascore;
explain select rank, title, metascore from imdb_parquet order by metascore desc limit
10;
# Statistics: Num rows: 1000 Data size: 12000 Basic stats: COMPLETE Column stats: NONE

explain select rank, title, metascore from imdb_parquet_small order by metascore desc
limit 10;
# Statistics: Num rows: 1000 Data size: 4000 Basic stats: COMPLETE Column stats: NONE
```

- **필요한 컬럼만 유지하는 경우에도 성능개선의 효과가 있는지 비교합니다**

```
cd /home/ubuntu/work/data-engineer-intermediate-training/day4/ex2
vimdiff sort.imdb_parquet.out sort.imdb_parquet_small.out
```

3-3 비정규화를 통한 성능 개선

> *일반적으로 관계형 데이터베이스의 경우 Redundent 한 데이터 적재를 피해야만 Consistency 문제를 회피할 수 있고 변경 시에 일관된 데이터를 저장할 수 있습니다. 그래서 PK, FK 등으로 Normalization & Denormalization 과정을 거치면서 모델링을 하게 됩니다. 하지만 분산 환경에서의 정규화 했을 때의 관리 비용 보다 Join 에 의한 리소스 비용이 더 큰 경우가 많고 Join 의 문제는 Columnar Storage 나 Spark 의 도움으로 많은 부분 해소될 수 있기 때문에 Denormalization 을 통해 Superset 데이터를 가지는 경우가 더 많습니다. Daily Batch 작업에서 아주 큰 Dimension 데이터를 생성하고 Daily Logs 와 Join 한 모든 데이터를 가진 Fact 데이터를 생성 (User + Daily logs) 하고 이 데이터를 바탕으로 일 별 Summary 혹은 다양한 분석 지표를 생성하는 것이 효과적인 경우가 많습니다*

- 분산 환경의 대부분의 프레임워크나 엔진들은 트랜잭션 및 Consistency 성능을 희생하여 처리량과 조회 레이턴시를 향상시키는 경우가 많습니다
- OLTP 가 아니라 OLAP 성 데이터 분석 및 조회의 경우에는 Join 을 통해 실시간 데이터 보다 Redundent 한 데이터를 가지고 빠른 분석이 더 유용한 경우가 많습니다
- 특히 Spark 의 경우에도 RDD 는 Immutable 이며 모든 RDD 는 새로 생성되는 구조인 점을 보면 더 이해가 빠르실 것입니다

3-4 글로벌 정렬 회피를 통한 성능 개선

> *Order By, Group By, Distribute By, Sort By, Cluster By 실습을 통해 차이점을 이해하고 활용합니다*
```

- Q1) 예제 emp.txt 파일은 중복된 레코드가 많아서 어떻게 하면 중복을 제거하고 emp.uniq.txt 파일을 생성할 수 있을까요?
  - Hint) cat, sort and redirection

```
bash>
docker-compose exec hive-server bash
cd /opt/hive/examples/files
cat emp.txt
```

cat emp.uniq.txt # 정답 John|31|6 Jones|33|2 Rafferty|31|1 Robinson|34|4 Smith|34|5 Steinberg|33|3

```
* 비라인을 통해 직원 및 부서 테이블을 생성합니다
```bash
bash>
beeline jdbc:hive2://localhost:10000 scott tiger
use testdb;

beeline>
drop table if exists employee;
create table employee (name string, dept_id int, seq int) row format delimited fields
terminated by '|';
load data local inpath '/opt/hive/examples/files/emp.uniq.txt' into table employee;

drop table if exists department;
create table department (id int, name string) row format delimited fields terminated
by '|';
load data local inpath '/opt/hive/examples/files/dept.txt' into table department;
```

- Q2) 테이블의 정보를 조회하고 어떻게 조인해야 employee + department 정보를 가진 테이블을 조회할 수 있을까요?
  - Hint) SELECT a.key, b.key FROM tableA a JOIN tableB b ON a.key = b.key

```
beeline>
desc employee;
desc department;
```

select * from users; // 정답 +------------+--------+-------+--------------+ | e.name | e.seq | d.id | d.name | +------------+--------+-------+--------------+ | John | 6 | 31 | sales | | Jones | 2 | 33 | engineering | | Rafferty | 1 | 31 | sales | | Robinson | 4 | 34 | clerical | | Smith | 5 | 34 | clerical | | Steinberg | 3 | 33 | engineering | +------------+--------+-------+--------------+

```
* Q3) 직원 이름, 지원 부서 아이디, 직원 부서 이름을 가진 users 테이블을 생성할 수 있을까요?
  * Hint) CREATE TABLE users AS SELECT ...
```bash
beeline>

desc users; // 정답
+-----------+------------+----------+
| col_name  | data_type  | comment  |
+-----------+------------+----------+
| username  | string     |          |
| seq       | int        |          |
| id        | int        |          |
```

```
| name       | string     |           |
+------------+------------+-----------+
```

- Order By - 모든 데이터가 해당 키에 대해 정렬됨을 보장합니다 ```bash beeline> select * from employee order by dept_id;
- ----------------+------------------+---------------+ | employee.name | employee.dept_id | employee.seq |
- ----------------+------------------+---------------+ | Rafferty | 31 | 1 | | John | 31 | 6 | | Steinberg | 33 | 3 | | Jones | 33 | 2 | | Smith | 34 | 5 | | Robinson | 34 | 4 |
- ----------------+------------------+---------------+ ```
- Group By - 군집 후 집계함수를 사용할 수 있습니다 ```bash beeline> select dept_id, count(*) from employee group by dept_id;
- ----------+------+ | dept_id | _c1 |
- ----------+------+ | 31 | 2 | | 33 | 2 | | 34 | 2 |
- ----------+------+ ```
- Sort By - 해당 파티션 내에서만 정렬을 보장합니다 - mapred.reduce.task = 2 라면 2개의 개별 파티션 내에서만 정렬됩니다 ```bash beeline> set mapred.reduce.task = 2; select * from employee sort by dept_id desc;
- ----------------+------------------+---------------+ | employee.name | employee.dept_id | employee.seq |
- ----------------+------------------+---------------+ | Smith | 34 | 5 | | Robinson | 34 | 4 | | Steinberg | 33 | 3 | | Jones | 33 | 2 | | Rafferty | 31 | 1 | | John | 31 | 6 |
- ----------------+------------------+---------------+ ```
- Distribute By - 단순히 해당 파티션 별로 구분되어 실행됨을 보장합니다 - 정렬을 보장하지 않습니다. ```bash beeline> select * from employee distribute by dept_id;
- ----------------+------------------+---------------+ | employee.name | employee.dept_id | employee.seq |
- ----------------+------------------+---------------+ | Steinberg | 33 | 3 | | Smith | 34 | 5 | | Robinson | 34 | 4 | | Rafferty | 31 | 1 | | Jones | 33 | 2 | | John | 31 | 6 |
- ----------------+------------------+---------------+ ```
- Distribute By Sort By - 파티션과 해당 필드에 대해 모두 정렬을 보장합니다 ```bash beeline> select * from employee distribute by dept_id sort by dept_id asc, seq desc;
- ----------------+------------------+---------------+ | employee.name | employee.dept_id | employee.seq |
- ----------------+------------------+---------------+ | John | 31 | 6 | | Rafferty | 31 | 1 | | Steinberg | 33 | 3 | | Jones | 33 | 2 | | Smith | 34 | 5 | | Robinson | 34 | 4 |
- ----------------+------------------+---------------+ ```
- Cluster By - 파티션 정렬만 보장합니다 - 특정필드의 정렬이 필요하면 Distribute By Sort By 를 사용해야 합니다 ```bash beeline> select * from employee cluster by dept_id;
- ----------------+------------------+---------------+ | employee.name | employee.dept_id | employee.seq |
- ----------------+------------------+---------------+ | Rafferty | 31 | 1 | | John | 31 | 6 | | Steinberg | 33 | 3 | | Jones | 33 | 2 | | Smith | 34 | 5 | | Robinson | 34 | 4 |
- ----------------+------------------+---------------+ ```
- 전체 Global Order 대신 어떤 방법을 쓸 수 있을까?
  - Differences between rank and row_number : rank 는 tiebreak 시에 같은 등수를 매기고 다음 등수가 없으나 row_number 는 아님 ```bash beeline> select * from (select name, dept_id, seq, rank() over (partition by dept_id order by seq desc) as rank from employee) t where rank < 2;
- ------------+------------+--------+---------+ | t.name | t.dept_id | t.seq | t.rank |
- ------------+------------+--------+---------+ | John | 31 | 6 | 1 | | Steinberg | 33 | 3 | 1 | | Smith | 34 | 5 | 1 |
- ------------+------------+--------+---------+

5 버킷팅을 통한 성능 개선

> *버킷팅을 통해 생성된 테이블의 조회 성능이 일반 파케이 테이블과 얼마나 성능에 차이가 나는지 비교해봅니다*

- 파케이 테이블의 생성시에 버킷을 통한 인덱스를 추가해서 성능을 비교해 봅니다 (단, CTAS 에서는 partition 혹은 clustered 를 지원하지 않습니다)

```
beeline>
create table imdb_parquet_bucketed (rank int, title string, genre string,
description string, director string, actors string, runtime int, rating string,
votes int, revenue string, metascore int) partitioned by (year string)
clustered by (rank) sorted by (metascore) into 10 buckets row format delimited
fields terminated by ',' stored as parquet;
```

set hive.exec.dynamic.partition=true; set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table imdb_parquet_bucketed partition(year='2006') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2006'; insert overwrite table imdb_parquet_bucketed partition(year='2007') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2007'; insert overwrite table imdb_parquet_bucketed partition(year='2008') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2008'; insert overwrite table imdb_parquet_bucketed partition(year='2009') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2009'; insert overwrite table imdb_parquet_bucketed partition(year='2010') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2010'; insert overwrite table imdb_parquet_bucketed partition(year='2011') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2011'; insert overwrite table imdb_parquet_bucketed partition(year='2012') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2012'; insert overwrite table imdb_parquet_bucketed partition(year='2013') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2013'; insert overwrite table imdb_parquet_bucketed partition(year='2014') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2014'; insert overwrite table imdb_parquet_bucketed partition(year='2015') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2015'; insert overwrite table imdb_parquet_bucketed partition(year='2016') select rank, title, genre, description, director, actors, runtime, rating, votes, revenue, metascore from imdb_movies where year = '2016';

```
* 생성된 파케이 테이블이 정상적으로 버킷이 생성되었는지 확인합니다
```sql
beeline>
desc formatted imdb_parquet_bucketed;
...
| Num Buckets:                  | 10                                                 |
NULL                        |
| Bucket Columns:               | [rank]                                             |
NULL                        |
| Sort Columns:                 | [Order(col:metascore, order:1)]                    |
```
```

```
NULL                        |
...
```

- 일반 파케이 테이블과 버킷이 생성된 테이블의 스캔 성능을 비교해봅니다

  > *TableScan 텍스트 대비 레코드 수에서는 2% (44/1488 Rows)만 읽어오며, 데이터 크기 수준에서는 약 0.1% (484/309,656 Bytes)만 읽어오는 것으로 성능 향상이 있습니다*

  ```
  beeline>
  explain select rank, metascore, title from imdb_parquet where year = '2006' and
  rank < 101 order by metascore desc;
  # Statistics: Num rows: 1000 Data size: 12000 Basic stats: COMPLETE Column
  stats: NONE
  ```

explain select rank, metascore, title from imdb_parquet_bucketed where year = '2006' and rank < 101 order by metascore desc;

Statistics: Num rows: 44 Data size: 484 Basic stats: COMPLETE Column stats: NONE

explain select rank, metascore, title from imdb_movies where year = '2006' and rank < 101 order by metascore desc;

Statistics: Num rows: 1488 Data size: 309656 Basic stats: COMPLETE Column stats: NONE