## Hive

- Doc: [LanguageManual DDL - Apache Hive - Apache Software Foundation]

- DDL Commands:

```
CREATE  (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
      [COMMENT database_comment]
      [LOCATION hdfs_path]


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


USE database_name; / USE DEFAULT;


CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
[db_name.]table_name
      [(col_name data_type [column_constraint_specification]
      [COMMENT col_comment], ... [constraint_specification])]
      [COMMENT table_comment]
      [PARTITIONED BY (col_name data_type [COMMENT col_comment],
      ...)]
      [LOCATION hdfs_path]
      [TBLPROPERTIES (property_name=property_value, ...)]
      [AS select_statement];


DROP TABLE [IF EXISTS] table_name;


TRUNCATE [TABLE] table_name;


ALTER TABLE table_name RENAME TO new_table_name;


ALTER TABLE table_name SET TBLPROPERTIES table_properties;

table_properties:
      : (property_name = property_value, property_name =
      property_value, ... )
```

```
ALTER TABLE page_view ADD PARTITION (dt='2008-08-08', country='us')
location '/path/to/us/part080808'
                 PARTITION (dt='2008-08-09', country='us') location
                 '/path/to/us/part080809';

SHOW CREATE TABLE;
```

- Hive DML Commands based on Movies dataset

0) CSV ტიპის managed ცხრილის შექმნა და დატის ჩაწერა
(პირდაპირ ექსტერნალ ცხრილის შექმნა ჯობია 😊 )

```
CREATE TABLE db_name.name
(
 …
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;


LOAD DATA INPATH 'hdfs:///…' INTO TABLE name
```

1) User-ების რაოდენობა და უნიკალური გვარების რაოდენობა:

```
select
    count(*) as total_num_of_users,
    count(distinct last_name) as unique_last_names
from users;
```

2) თითოეული ფილმისთვის ყველა უნიკალური ტეგი:
ა) თითოეული თეგი row-ებად:

```
select distinct
    movie_id,
    tag
from tags;
```

ბ) თეგები სეტის სახით:

```
select
    movie_id,
    collect_set(tag) as tags_set
from tags
group by
    movie_id;
```

3) Top-10 ყველაზე ხშირი თეგი:

```
SELECT
    tag,
    count(*) as freq
FROM movies.tags
GROUP BY tag
ORDER BY freq desc
LIMIT 10;
```

4) User-ების ტრანსფორმაცია - კონკატენაცია, struct vs named_struct:

```
select
    concat_ws(' ', first_name, last_name) full_name,
    struct(first_name, last_name) as user_details,
    named_struct('first_name', first_name, 'last_name', last_name) as
user_details_named
from users;
```

5) Map ტიპის ველის შექმნის მაგალითი და სტრინგის დაკასტვა:

```
select
   map(1, 'A', 2, 'B') as `map`,
   str_to_map('1:A, 2:B') as `str_to_map`
```

6) User-ების ასაკის გამოთვლა და საშუალო ასაკი:

```
with user_data as
(
    select
        concat_ws(' ', first_name, last_name) full_name,
        from_unixtime(unix_timestamp(birth_date, 'm/dd/yyyy'), 'yyyy-
mm-dd') birth_date
    from users
)
select
    full_name,
    cast(months_between(current_date(), birth_date) / 12 as int) as age
from user_data;
```

იგივე Subquery-თი:

```
select
    full_name,
    cast(months_between(current_date(), birth_date) / 12 as int) as age
from
    (
        select
            concat_ws(' ', first_name, last_name) full_name,
            from_unixtime(unix_timestamp(birth_date, 'm/dd/yyyy'),
'yyyy-mm-dd') birth_date
        from users
    ) u;
```

საშუალო ასაკი:

```
select
    avg(cast(months_between(current_date(), birth_date) / 12 as int))
as avg_age
from
    (
        select
            concat_ws(' ', first_name, last_name) full_name,
            from_unixtime(unix_timestamp(birth_date, 'm/dd/yyyy'),
'yyyy-mm-dd') birth_date
            from users
    ) u
```

7) Row-ების list-ად ან Set-ად გარდაქმნა:

```
select
    collect_list(fruit) as all_fruits,
    collect_set(fruit) as unique_fruits
from
      (
            Select
                  'Apple' as fruit
            Union all
            Select
                  'Banana' as fruit
            Union all
            Select
                  'Pear' as fruit
            Union all
            Select
                  'Peach' as fruit
Union all
Select
 'Apple' as fruit
   ) fruits
```

8) შეიცავს თუ არა სია დუბლირებებს?

```
select
    fruit,
    count(*) as cnt
from
      (
            Select
                  'Apple' as fruit
            Union all
            Select
                  'Banana' as fruit
            Union all
            Select
                  'Pear' as fruit
            Union all
            Select
                  'Peach' as fruit
        Union all
        Select
            'Apple' as fruit
    ) fruits
group by
    fruit
having
    cnt > 1;
```

ან სხვანაირად:

```
with fr as
(
    select
        collect_list(fruit) as all_fruits,
        collect_set(fruit) as unique_fruits
    from
      (
            Select
                  'Apple' as fruit
            Union all
            Select
                  'Banana' as fruit
            Union all
            Select
                  'Pear' as fruit
            Union all
            Select
                  'Peach' as fruit
            Union all
            Select
                  'Apple' as fruit
        ) fruits
)
select
    case
        when size(all_fruits) != size(unique_fruits)
            then 'Contains duplicates'
        else 'Does not contain duplicates'
    end as result
from fr
where size(all_fruits) != size(unique_fruits);
```

9) Partitioned Tables: შევქმნათ ლოკაციაზე hdfs:///data_lake/users default სქემაში country-ებით დაფართიშენებული user-ების external ცხრილი და ჩავწეროთ შიგნით data:

```
create external table default.users_part_by_country
(
    user_id string,
    first_name string,
    last_name string,
    birth_date string
)
partitioned by
    (country string)
stored as parquet
location 'hdfs:///data_lake/users';

SET hive.exec.dynamic.partition.mode=nonstrict;

insert into default.users_part_by_country
partition(country)
select
    *
from staging.users;
```

## BigQuery

1) SDK:
   - **bq** = big query command
   - **bq show**
   - **bq show [dataset name] movies**
   - **bq show movies.movies [dataset.tablename]**
   - **bq query 'select * from movies.movies'**
   - **bq query 'select country from movies.users group by country order by 1'**
   - **bq shell**
     - **ls movies vs show movies**
     - **extract movies.movies gs://yet_another_bucket/extracted/ext_movies**
     - **mk -t movies.users user_id:integer,first_name:string,last_name:string,birth_date:string,country:string**
     - **load movies.users gs://yet_another_bucket/ml-latest-small/users.csv**
     - **select * from movies.users**
     - **rm movies.users**
     - **head -n 3 movies.movies**
     - **head -s 1 -n 3 movies.movies**
2) DDL-ების არაერთი მაგალითი არის Git-ის პროექტში Python client-ის სახით (bq.py);
3) SQL IDE:
   - Complex types examples:

```
SELECT
    ['A', 'B', 'C'] as str_array,
    [True, False] as bool_array;


SELECT struct('Student1' as name,
              'st1@fr.edu.ge' as email,
               struct(true as graduate, 2017 as grad_year) as grad_status) as
student
union   all
```

```
SELECT struct('Student2' as name,
              'st2@fr.edu.ge' as email,
               struct(false as graduate, null as grad_year) as grad_status) as
student;
```

- CTE & struct filter:

```
with students as
(
    SELECT struct('Student1' as name,
                  'st1@fr.edu.ge' as email,
                  struct(true as graduate, 2017 as grad_year) as grad_status) as
student
    union   all
    SELECT struct('Student2' as name,
                  'st2@fr.edu.ge' as email,
                  struct(false as graduate, null as grad_year) as grad_status)
as student
)
select *
from students s
where s.student.grad_status.grad_year is null;
```

- External table examples:

```
create external table uni.students
(
    uni string,
    student struct<name string, email string, grad_status struct<graduate
boolean, grad_year int>>
) options (
    format = 'json',
    uris = ['gs://yet_another_bucket/uni/students']
 );

create external table `complete-verve-325920.uni.movies`
(
    movie_id string,
    title string,
    genres string
) options (
```

```
    format = 'csv',
    uris = ['gs://yet_another_bucket/ml-latest-small/movies.csv']
);
```

- Partitioned table example:

```
create table uni.students
(
    enroll_year datetime,
    student struct<name string, email string, grad_status struct<graduate boolean, grad_year
int>>
)
partition by datetime_trunc(enroll_year, year);

insert into uni.students
SELECT
        current_date() as enroll_year,
        struct('Student1' as name,
               'st1@fr.edu.ge' as email,
                struct(true as graduate, 2017 as grad_year) as grad_status) as student
union   all
SELECT
        date_sub(current_date(), INTERVAL 2 year) enroll_year,
        struct('Student2' as name,
               'st2@agr.edu.ge' as email,
                struct(false as graduate, null as grad_year) as grad_status) as student;
```