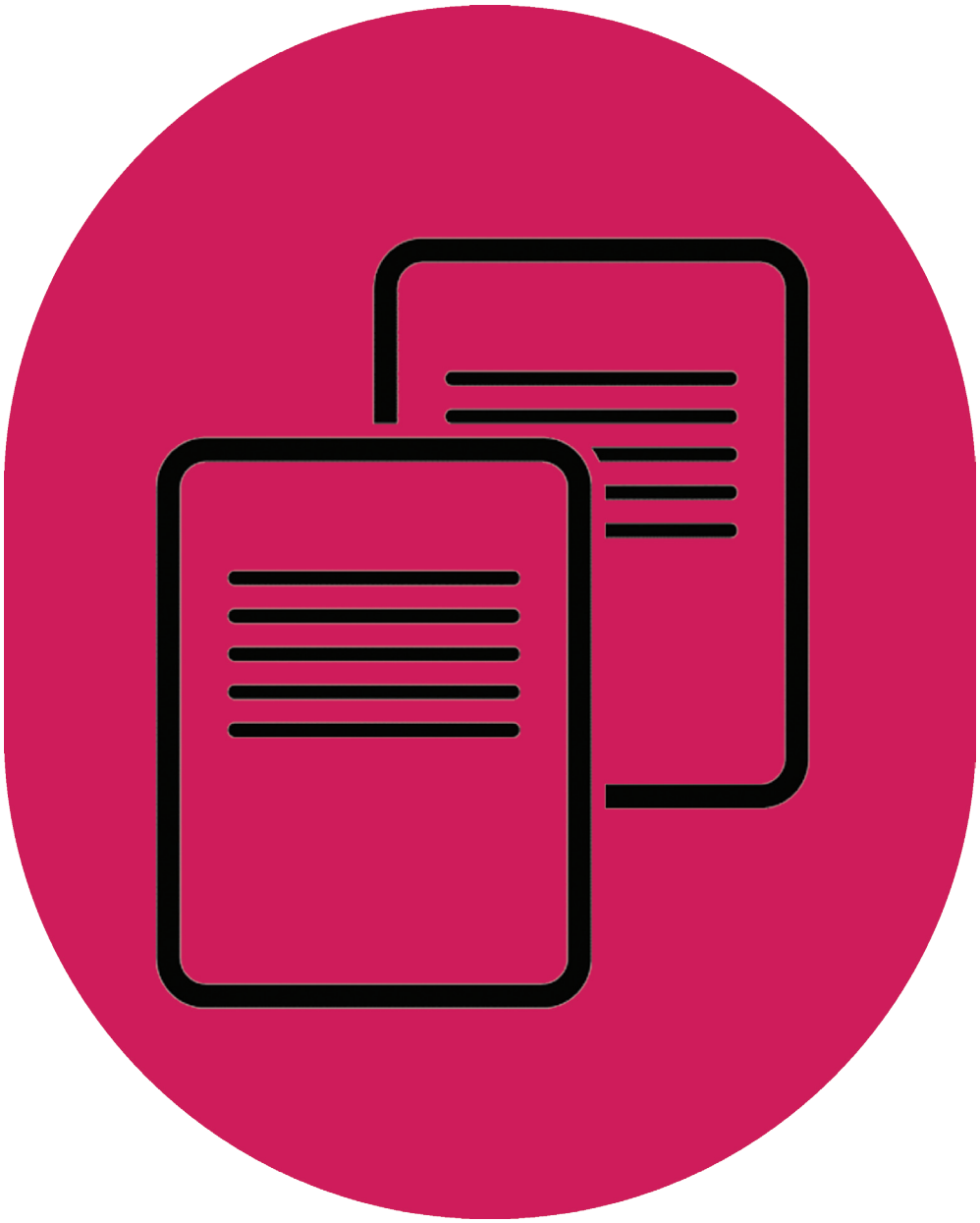

HiveQL example queries

UK Data Service

Author: UK Data Service
Created: April 2016
Version: 1

Contents

1. Introduction

This workbook contains some practical excises for researchers and/or data analysts who want to run simple queries using Apache Hive. The queries in this document are the ones which were used as part of the 'What is Hive?' webinar. A few of the simpler queries, which were repeated for different tables, have been omitted for brevity.

Hive is a data warehouse infrastructure and supports analysis of large datasets stored in Hadoop's HDFS and compatible file systems. It provides an SQL (Structured Query Language)-like language called Hive Query Language (HiveQL).

1.1. Sandbox

These HiveQL queries can be run on a Sandbox running Hadoop in which Hive is already available. To set up your own Sandbox please follow the instructions available in the Obtaining and downloading the HDP Sandbox guide.

1.2. Data used

The data to be used to run these queries in Hive are the Energy Demand Research Project: Early Smart Meter Trials, 2007-2010, a set of trials on smart meter data available for download from the UK Data Service. You can simply download the data after registering with the UK Data Service. Once you have downloaded the data, you have to load it into the Hadoop Distributed File System (HDFS) in the Sandbox. More information and instructions about how to load data into HDFS can be found in the Loading data into HDFS guide.

2. Running HiveQL queries

Once you have loaded the Smart Meter Trials data into HDFS, you can run HiveQL queries.

For more information on how to run the following queries watch the 'What is Hive?' webinar recording or see Section 7 of the Obtaining and downloading the HDP Sandbox guide.

2.1. Creating tables

When an external table is created in Hive, the table definition is used to describe the layout of the data within the dataset and the location of the data within HDFS to Hive. Hive doesn't actually move the data into the table.

2.1.1. Creating the Readings (allgas) table

```
CREATE EXTERNAL TABLE allgas
(
   anon_id              INT,
   advancedatetime      STRING,
   hh                   INT,
   gaskwh               DOUBLE
```

```
)
ROW FORMAT DELIMITED
           FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/energy'
TBLPROPERTIES ("skip.header.line.count" = "1");
```

2.1.2. Creating the Geography table

```
CREATE EXTERNAL TABLE geography
(
   anonid               INT,
   eprofileclass        INT,
   fueltypes            STRING,
   acorn_category       INT,
   acorn_group          STRING,
   acorn_type           INT,
   nuts4                STRING,
   lacode               STRING,
   nuts1                STRING,
   gspgroup             STRING,
   ldz                  STRING,
   gas_elec             STRING,
   gas_tout             STRING
)
ROW FORMAT DELIMITED
           FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/geography'
TBLPROPERTIES ("skip.header.line.count" = "1");
```

2.2. Simple Queries

These simple queries are used to show how many records there are in a table and to show the first few (5) records of the table. The queries can easily be adapted for other tables by simply changing the table name (allgas in this case)

2.2.3. Count the number of records in the allgas table

```
select count(*) from allgas;
```

2.2.4. Show the first five records from the allgas table

```
select * from gas_days limit 5;
```

2.3. Creating frequency tables

Despite the title, these tables don't actually create "tables" in Hive, they simply show the numbers in each category of a categorical variable in the results. If you want to store the results in a table for future use, see Section 2.3.6.

2.3.5. Count occurrences of values within fields in the geography table

```
select acorn_category,
       count(*) as acorn_categorycount
from geography
group by  acorn_category;

select acorn_group,
       count(*) as acorn_groupcount
from geography
group by  acorn_group;

select acorn_type,
       count(*) as acorn_typecount
from geography
group by  acorn_type;

select nuts1,
       count(*) as nuts1count
from geography
group by nuts1;

select nuts4,
       count(*) as nuts4count
from geography
group by  nuts4;
```

2.3.6. Storing the results in a table for future use

```
Create table acorn_category_frequency
  as
select acorn_category,
       count(*) as acorn_categorycount
from geography
group by  acorn_category;
```

2.4. Aggregating and filtering data

2.4.7. Aggregation of the allgas table into gas used on a daily basis

```
create table gas_days as
select anon_id,
       substr(advancedatetime,1,7) as reading_date,
       sum(gaskwh) as totkwh,
       count(*) as count_readings
from allgas
group by anon_id, substr(advancedatetime,1,7);
```

2.4.8. Count the occurrences of each different count_readings values

```
select count_readings, count(*) as num_readings
from gas_days
group by count_readings
order by count_readings;
```

2.4.9. Re-create the gas_days table using only complete sets (48) of readings

```
create table gas_days as
select anon_id,
       substr(advancedatetime,1,7) as reading_date,
       sum(gaskwh) as totkwh,
       count(*) as count_readings
from allgas
group by anon_id, substr(advancedatetime,1,7)
having count_readings = 48;
```

2.4.10. Create the gas_months table

```
create table gas_months as
select anon_id,
       substr(reading_date,3,5) as reading_month,
       sum(totkwh) as monthlykwh,
       count(*) as count_readings
from gas_days
group by anon_id, substr(reading_date,3,5)
order by anon_id;
```

2.4.11. Create the gas_months_09 table

```
create table gas_months_09 as
select *
from gas_months
where substr(reading_month, 4,2) = '09';
```

2.5. Creating Summary Statistics

As well as using frequency tables, it is often useful to have some summary statistics relating to a column in the table. This can be done on a whole table basis, or like the earlier frequency tables the results are grouped by a particular column, in this case the anon_id column.

2.5.12. Create the gasdays_summary table

```
create table gasdays_summary as
select anon_id,
       min(totkwh) as Minimum,
       avg(totkwh) as Average,
       max(totkwh) as Maximum
from gas_days
   group by anon_id
   order by anon_id;
```

2.6. Joining tables

Joining tables is a very common requirement. In order to join two tables, there must be a column which is common to both. The name doesn't matter; it is the usage that counts. Although it is not an actual requirement, in most joins the tables involved are given simple alias' (a and b in this case), this makes it obvious from which table each column comes. In cases where the two tables have columns with the same name, which is quite often for the columns which they have in common and you are joining on, then the use of alias' is required.

2.6.13. Create the gas_09_geog table

```
create table gas_09_geog as
select a.anon_id,
       a.reading_month,
       a.monthlykwh,
       a.count_readings,
       b.fueltypes,
       b.acorn_category,
       b.acorn_group,
       b.acorn_type,
       b.nuts4,
       b.lacode,
       b.nuts1
from geography as b
join gas_months_09 as a
on b.anonid  = a.anon_id;
```

April 2016

T +44 (0) 1206 872143
E help@ukdataservice.ac.uk
W ukdataservice.ac.uk

The UK Data Service provides the UK's largest collection of social, economic and population data resources

UK Data Service

E·S·R·C
ECONOMIC & SOCIAL RESEARCH COUNCIL