HIVE Manual:

Apache **Hive** is a data warehouse software project built on top of Apache **Hadoop** for providing data query and analysis.**Hive** gives a SQL-like interface to query data stored in various databases and file systems that integrate with **Hadoop**

Key Differences between Hadoop vs Hive: ... 1) Hadoop is a framework to process/query the Big data while Hive is an SQL Based tool which builds overHadoop to process the data. 2) Hive process/query all the data using HQL (HiveQuery Language) it's SQL-Like Language while Hadoop can understand Map Reduce only.

Is hive a relational database?

No, we cannot call Apache **Hive a relational database**, as it is a data warehouse which is built on top of Apache Hadoop for providing data summarization, query and, analysis. It differs from a **relational database** in a way that it stores schema in a**database** and processed data into HDFS.

Difference between Hive and **HBase**

Hive vs. **HBase** - **Difference between Hive** and **HBase**. **Hive** is query engine that whereas **HBase** is a data storage particularly for unstructured data. Apache **Hive** is mainly used for batch processing i.e. ... Unlike **Hive**, operations in **HBase** are run in real-time on the database instead of transforming into mapreduce jobs

Apache **Hive** is a SQL layer on top of Hadoop. **Hive** uses a SQL-like HiveQL query language to execute queries over the large volume of data stored in HDFS. HiveQL queries are executed using Hadoop MapReduce, but **Hive** can also use other distributed computation engines like Apache **Spark** and Apache Tez.

DDL: create table, create index, create views.
DML: Select, Where, group by, Join, Order By

Pluggable Functions:
UDF: User Defined Function

Hive DDL Commands

create database
drop database
create table

drop table
alter table
create index
create views

Hive DML Commands

Select
Where
Group By
Order By
Load Data
Join:

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

create database and tables --- text file format
create database and tables --- orc file format

running hive queries
spark sql application--- Hive or SQLContext
spark sql application ---Dataframe Operations.

create table:

```
load data  inpath '/user/root/orders/' into table orders;

create database ram_retail_db_txt_2;
use ram_retail_db_txt_2;

create table orders (
  order_id int,
  order_date string,
  order_customer_id int,
  order_status string
)row format delimited fields terminated by '_'
stored as textfile;
```

```
create table orders (
  order_id int,
  order_date string,
  order_customer_id int,
  order_status string
) row format delimited fields terminated by ','
stored as textfile;

load data  inpath '/data/retail_db/orders' into table orders;

create table order_items (
  order_item_id int,
  order_item_order_id int,
  order_item_product_id int,
  order_item_quantity int,
  order_item_subtotal float,
  order_item_product_price float
) row format delimited fields terminated by ','
stored as textfile;

load data local inpath '/data/retail_db/order_items' into table order_items;
```

ORC:

```
create database dgadiraju_retail_db_orc;
use dgadiraju_retail_db_orc;

create table orders (
  order_id int,
  order_date string,
  order_customer_id int,
  order_status string
)
stored as orc;

load data  inpath '/user/root/p91_orders' into table orders;

insert into table orders select * from user_txt.orders;


create table order_items (
  order_item_id int,
  order_item_order_id int,
  order_item_product_id int,
  order_item_quantity int,
  order_item_subtotal float,
  order_item_product_price float
)
stored as orc;

load data  inpath '/user/root/p92_orders_items' into table order_items;

insert into table order_items  select * from user_txt.order_items;
```

==
=============================

Hive String functions:

```
create table customers(
customer_id int,
customer_fname varchar(45),
customer_lname varchar(45),
customer_email varchar(45),
customer_password varchar(45),
customer_street varchar(45),
customer_city varchar(45),
customer_state varchar(45),
customer_zipcode varchar(45)
)
row format delimited fields terminated by ','
stored as textfile;

load data inpath '/user/root/customers/' into table customers;


show functions;
substr(str,starting position,length of words)

select substr('Hello World ,How are you?',-5,5);
```

substr
instr

like
rlike
length
lcase or lower
ucase or upper
trim,ltrim,rtrim
lpad,rpad
split
initcap

desc function like;

select "Hello World,How are you?" like 'Hello'
select "Hello World,How are you?" like 'Hello%'
select "Hello World,How are you?" like '%World%'
select "Hello World,How are you?" rlike '[a-z]'

lpad and rpad===== using to appernd the string value.

select rpad(2,3,'R');
select lpad(2,3,'R');

select cast(substr(order_date,6,2) as int) from orders limit 10;

select split("Hello world, how are you?"," ")
select index(split("Hello world, how are you?"," "),3);

==

Date Manupulations:

current_date
current_timestamp
date_add
date_format
date_sub
datediff
day
dayofmonth
to_date

to_unix_timestamp
to_utc_timestamp----read
from_utc_timestamp---read
minute
month
months_between
next_day

select to_date(current_timestamp);

select cast(substr(order_date,6,2) as int) from orders limit 10;

http://hadooptutorial.info/hive-date-functions/ ---------------------> date time conversations.

case and nvl:

CASE a when b then c [when d then e] * [else f] end------ when a=b, return c; when a=d, retrun e; else return f;

```
select  case order_status
          when 'CLOSED' then 'NO Action'
          when 'COMPLETE' then 'No Action'
          end from orders limit 10;
```

```
select order_status,
      case
          when order_status IN ('CLOSED', 'COMPLETE') then 'No Action'
          when order_status IN ('ON_HOLD', 'PAYMENT_REVIEW', 'PENDING',
'PENDING_PAYMENT', 'PROCESSING') then 'Pending Action'
          else 'Risky'
      end from orders limit 10;
```

Nvl :It is special type case functions.

If customer primary contact medium is email, if email is null then phonenumber, and if phonenumber is also null then address.
It would be written using COALESCE as

select nvl(order_status,'STATUS_MISSING') from orders limit 100;

or

select order_status case when order_staus is null then 'STATUS_MISSING' else order_status end from orders limit 100;

coalesce(email,phonenumber,address)

===

Row level transform and standardations.

select cast(concat(substr(order_date,1,4),substr(order_date,6,2)) as int) from orders limit 10;

select cast(date_format(order_date,'yyyyMM') as int) from orders limit 10;

select cast(date_format(order_date,'yyyyMM') as int) from orders limit 10;

Joining:

select o.*,c.* from orders o,customers c where o.order_customer_id = c.customer_id limit 10;(filter is different using where)
or
select o.*,c.* from orders o inner join customers c on o.order_customer_id = c.customer_id limit 10;

select o.*,c.* from orders o left outer join customers c on o.order_customer_id = c.customer_id limit 10;

select count(1) from orders o inner join customers c on o.order_customer_id = c.customer_id limit 10;

select count(1) from orders o left outer join customers c on o.order_customer_id = c.customer_id limit 10 where o.order_customer_id is null;

select * from customers where customer_id not in(select distinct(order_customer_id) from orders)

==

GroupBy and Aggregations:

```
  select order_status,count(1) from orders group by order_status;

  select o.order_id,o.order_date,o.order_status,sum(oi.order_item_subtotal) as order_revenue
from orders o join order_items oi on o.order_id= oi.order_item_order_id group by
o.order_id,o.order_status,o.order_date;

  select o.order_id,o.order_date,o.order_status,sum(oi.order_item_subtotal) as order_revenue
from orders o join order_items oi on o.order_id= oi.order_item_order_id group by
o.order_id,o.order_status,o.order_date having sum(oi.order_item_subtotal)>=1000;

  daily revenue per day:
  select o.order_date, round(sum(oi.order_item_subtotal)) as daily_order_revenue from orders o
join order_items oi on o.order_id= oi.order_item_order_id
  where o.order_status in('COMPLETE','CLOSED')
  group by o.order_date;
```

  ==

Sorting the Data:

```
  select o.order_id, o.order_date, o.order_status, round(sum(oi.order_item_subtotal), 2)
order_revenue
from orders o join order_items oi
on o.order_id = oi.order_item_order_id
where o.order_status in ('COMPLETE', 'CLOSED')
group by o.order_id, o.order_date, o.order_status
having sum(oi.order_item_subtotal) >= 1000
order by o.order_date, order_revenue desc;

select o.order_id, o.order_date, o.order_status, round(sum(oi.order_item_subtotal), 2)
order_revenue
from orders o join order_items oi
on o.order_id = oi.order_item_order_id
where o.order_status in ('COMPLETE', 'CLOSED')
group by o.order_id, o.order_date, o.order_status
having sum(oi.order_item_subtotal) >= 1000
distribute by o.order_date sort by o.order_date, order_revenue desc;
```

===
=====

Set Operations:

Union and UnionAll

```
select 1, "Hello"
union all
select 2, "World"
union all
select 1, "Hello"
union all
select 1, "World";

select 1, "Hello"
union
select 2, "World"
union
select 1, "Hello"
union
select 1, "World";
```

==
=========

Analytic Functions:

1)Windows functions
  lead,lag,first value,last value.
2)Over cluause
  All aggregate functions

Ranking:

```
rank() over (partition by o.order_id order by oi.order_item_subtotal desc) rnk_revenue,
```

```
val dataset = Seq(
  ("Thin",       "cell phone", 6000),
  ("Normal",     "tablet",       1500),
  ("Mini",       "tablet",       5500),
  ("Ultra thin", "cell phone", 5000),
  ("Very thin",  "cell phone", 6000),
  ("Big",        "tablet",       2500),
  ("Bendable",   "cell phone", 3000),
  ("Foldable",   "cell phone", 3000),
  ("Pro",        "tablet",     4500),
  ("Pro2",       "tablet",     6500))
  .toDF("product", "category", "revenue")

dataset.registerTempTable("items")
```

Rank:

```
sqlContext.sql("select *,rank() over (partition by category order by revenue desc) rnk from
items").show


sqlContext.sql("select *,dense_rank() over (partition by category order by revenue desc) rnk
from items").show
```

```
scala>  sqlContext.sql("select *,rank() over (partition by category order by revenue desc) rnk
from items").show
+----------+----------+-------+---+
|   product|  category|revenue|rnk|
+----------+----------+-------+---+
|      Pro2|    tablet|   6500|  1|
|      Mini|    tablet|   5500|  2|
|       Pro|    tablet|   4500|  3|
|       Big|    tablet|   2500|  4|
|    Normal|    tablet|   1500|  5|
|      Thin|cell phone|   6000|  1|
| Very thin|cell phone|   6000|  1|
|Ultra thin|cell phone|   5000|  3|
|  Bendable|cell phone|   3000|  4|
|  Foldable|cell phone|   3000|  4|
+----------+----------+-------+---+
```

```
scala> sqlContext.sql("select *,dense_rank() over (partition by category order by revenue desc)
rnk from items").show
+----------+----------+-------+---+
|   product|  category|revenue|rnk|
+----------+----------+-------+---+
|          Pro2|   tablet|   6500|  1|
|          Mini|    tablet|   5500|  2|
|          Pro|     tablet|   4500|  3|
|          Big|      tablet|   2500|  4|
|          Normal|          tablet|   1500|  5|
|          Thin|cell phone|   6000|  1|
| Very thin|cell phone|   6000|  1|
|Ultra thin|cell phone|   5000|  2|
|  Bendable|cell phone|   3000|  3|
|  Foldable|cell phone|   3000|  3|
+----------+----------+-------+---+


scala>

scala>

scala>

scala>

scala> sqlContext.sql("select *,row_number() over (partition by category order by revenue desc)
rnk from items").show
+----------+----------+-------+---+
|   product|  category|revenue|rnk|
+----------+----------+-------+---+
|          Pro2|   tablet|   6500|  1|
|          Mini|    tablet|   5500|  2|
|          Pro|     tablet|   4500|  3|
|          Big|      tablet|   2500|  4|
|          Normal|          tablet|   1500|  5|
|          Thin|cell phone|   6000|  1|
| Very thin|cell phone|   6000|  2|
|Ultra thin|cell phone|   5000|  3|
|  Bendable|cell phone|   3000|  4|
|  Foldable|cell phone|   3000|  5|
+----------+----------+-------+---+


scala>
```

```
select o.order_id, o.order_date, o.order_status,oi.order_item_subtotal,
round(sum(oi.order_item_subtotal) over(partition by o.order_id), 2) order_revenue,
oi.order_item_subtotal/round(sum(oi.order_item_subtotal) over(partition by o.order_id), 2) from
orders o join order_items oi
on o.order_id = oi.order_item_order_id
where o.order_status in ('COMPLETE', 'CLOSED')
distribute by o.order_date sort by o.order_date, order_revenue desc;
```

```
select * from (
select o.order_id, o.order_date, o.order_status, oi.order_item_subtotal,
round(sum(oi.order_item_subtotal) over (partition by o.order_id), 2) order_revenue,
oi.order_item_subtotal/round(sum(oi.order_item_subtotal) over (partition by o.order_id), 2)
pct_revenue,
round(avg(oi.order_item_subtotal) over (partition by o.order_id), 2) avg_revenue
from orders o join order_items oi
on o.order_id = oi.order_item_order_id
where o.order_status in ('COMPLETE', 'CLOSED')) q
where order_revenue >= 1000
order by order_date, order_revenue desc, rank_revenue;
```

Advanced Sql Function in hive:

Analytic function in Aggregations:

```
 select o.*,round(sum(oi.order_item_subtotal),2) order_revenue from orders o join order_items
 oi on o.order_id = oi.order_item_order_id where o.order_status in('COMPLETE','CLOSED')
 group by o.order_id,o.order_date,o.order_status,o.order_customer_id having
sum(oi.order_item_subtotal)>1000
 order by o.order_date desc,order_revenue;
```

```
 select o.order_id,o.order_date,o.order_status,oi.order_item_subtotal,
 round(sum(order_item_subtotal) over(partition by o.order_id),2) order_revenue ,
 order_item_subtotal/round(sum(order_item_subtotal) over(partition by order_id ),2)
 from orders o join order_items oi on o.order_id = oi.order_item_order_id
 where o.order_status in('COMPLETE','CLOSED') order by order_date,order_revenue;
```

Sub Query:

```
 select * from(select o.order_id,o.order_date,o.order_status,oi.order_item_subtotal,
 round(sum(order_item_subtotal) over(partition by o.order_id),2) order_revenue ,
 order_item_subtotal/round(sum(order_item_subtotal) over(partition by order_id ),2)
```

from orders o join order_items oi on o.order_id = oi.order_item_order_id
where o.order_status in('COMPLETE','CLOSED')) sample where order_revenue >1000 order by order_date,order_revenue;

select * from(select o.order_id,o.order_date,o.order_status,oi.order_item_subtotal,
round(sum(order_item_subtotal) over(partition by o.order_id),2) order_revenue ,
order_item_subtotal/round(sum(order_item_subtotal) over(partition by order_id),2) pct_revenue,
round(avg(order_item_subtotal) over(partition by o.order_id),2) avg_revenue
from orders o join order_items oi on o.order_id = oi.order_item_order_id
where o.order_status in('COMPLETE','CLOSED')) sample1 where order_revenue >1000 order by order_date,order_revenue;

Ranking Method:

select * from(select o.order_id,o.order_date,o.order_status,oi.order_item_subtotal,
round(sum(order_item_subtotal) over(partition by o.order_id),2) order_revenue ,
order_item_subtotal/round(sum(order_item_subtotal) over(partition by order_id),2) pct_revenue,
round(avg(order_item_subtotal) over(partition by o.order_id),2) avg_revenue,
rank() over (partition by o.order_id order by oi.order_item_subtotal desc) rnk_revenue,
dense_rank() over (partition by o.order_id order by oi.order_item_subtotal desc) dense_rnk_revenue,
percent_rank() over (partition by o.order_id order by oi.order_item_subtotal desc) percent_rnk_revenue,
row_number() over (partition by o.order_id order by oi.order_item_subtotal desc) rn_revenue,
row_number() over (partition by o.order_id) order_rn_revenue
from orders o join order_items oi on o.order_id = oi.order_item_order_id
where o.order_status in('COMPLETE','CLOSED')) sample1 where order_revenue >1000 order by order_date,order_revenue desc,rnk_revenue;

Windowing Functions:

Lag
Lead
First_Value
Last_Value.

select * from(select o.order_id,o.order_date,o.order_status,oi.order_item_subtotal,
round(sum(order_item_subtotal) over(partition by o.order_id),2) order_revenue ,

order_item_subtotal/round(sum(order_item_subtotal) over(partition by order_id),2)
pct_revenue,
round(avg(order_item_subtotal) over(partition by o.order_id),2) avg_revenue,
rank() over (partition by o.order_id order by oi.order_item_subtotal desc) rnk_revenue,
dense_rank() over (partition by o.order_id order by oi.order_item_subtotal desc)
dense_rnk_revenue,
percent_rank() over (partition by o.order_id order by oi.order_item_subtotal desc)
percent_rnk_revenue,
row_number() over (partition by o.order_id order by oi.order_item_subtotal desc) rn_revenue,
row_number() over (partition by o.order_id) order_rn_revenue,
lead(oi.order_item_subtotal) over (partition by o.order_id) lead_order_item_subtotal,
lag(oi.order_item_subtotal) over (partition by o.order_id) lag_order_item_subtotal,
first_value(oi.order_item_subtotal) over (partition by o.order_id)
first_value_order_item_subtotal,
last_value(oi.order_item_subtotal) over (partition by o.order_id) last_value_order_item_subtotal
from orders o join order_items oi on o.order_id = oi.order_item_order_id
where o.order_status in('COMPLETE','CLOSED')) sample1 where order_revenue >1000 order
by order_date,order_revenue desc,rnk_revenue;

select o.order_id,o.order_status,oi.order_subtotal,round(sum(oi.order_item_subtotal),2) as
total_Revenue,
rank() over(partition by o.order_id order by oi.order_subtotal desc) rnk_revenue,
lead(oi.order_item_subtotal) over (partition by o.order_id) lead_subtotal
from orders o join order_items oi on o.order_id = oi.order_item_order_id
where o.order_status in('COMPLETE','CLOSED') order by rnk_revenue;

Hive:

DML-----Load,insert,update and delete

Load:

CREATE TABLE tab1 (col1 int, col2 int) PARTITIONED BY (col3 int) STORED AS ORC;
LOAD DATA LOCAL INPATH 'filepath' INTO TABLE tab1;

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

OVERWRITE-----> delete the table name and

Insert:

insert into table tablename [partition(partcol1=val1,partcol = val2)] select * from statement;

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select_statement1 FROM from_statement;
INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1 FROM from_statement;

calculating total revenue per product and showing the top 10 revenue generating products::::

select c.category_name,count(order_item_quantity) as count from order_items oi join products p on oi.order_item_product_id = p.product_id join categories c on p.product_category_id = category_id
group by c.category_name order by count desc limit 10;

DDL Commands:

copy the existing table to new table:

create table dbname.new table name like dbname.old table name(existing table name);

drop table tablename;

truncate table tablename;

alter table dbname.old tablename rename to dbname.new table name;

1) create table table2 as select * from table1 where 1=1; or create table table2 as select * from table1;

2) insert overwrite table table2 select * from table1; --it will insert data from one to another. Note: It will refresh the target.

3) insert into table table2 select * from table1; --it will insert data from one to another. Note: It will append into the target.

4) load data local inpath 'local_path' overwrite into table table1; --it will load data from local into the target table and also refresh the target table.

5) load data inpath 'hdfs_path' overwrite into table table1; --it will load data from hdfs location iand also refresh the target table. or