Hive

Transactional and Analytical Processing

Transactional Processing	Analytical Processing
Analyzes individual entries	Analyzes large batches of data
Access to recent or immediate data, from the last few hours or days	Access to older data going back months, or even years
Updates data Involves many write operations	Only reads data
Updates data Fast real-time access	Long running jobs
Usually a single data source	Multiple data sources

Small data

- Both the objects could be achieved using the same database system.
- Single machine with backup
- Structured, well-defined data
- Can access individual records or the entire dataset
- No replication, updated data available instantaneously
- Different tables store data from different sources

Big data

- Very hard to meet all requirements with the same database system
- Data distributed on a cluster with multiple machines
- Semi-structured or unstructured data
- No random access to data
- Data replicated, propagation of updates take time
- Different sources may have different unknown formats

Data warehouse

- It typically has long running batch jobs that are used to crunch and much the data.
- They are optimized for read operations because there is no real time processing or real time updates on the data.
- The data is from multiple sources.
- Data stored tends to go back many years in time.
- Data may be lagged, not real-time.
- Examples of data warehouses – Vertica, Teradata, Oracle, IBM

Apache Hive

- It is an open-source data warehouse.
- Hive is part of the larger Hadoop ecosystem.
- Hive runs on top of the Hadoop distributed computing framework.
- It abstracts away the distributed nature of the storage and the processing to present a simple SQL-like interface.

Hive

- Hive stores its data in HDFS
- Hadoop Distributed File System
  - Data is stored as files - text files, binary files
  - Partitioned across machines in the cluster
  - Replicated for fault tolerance
  - Processing tasks parallelized across multiple machines
- Hive runs all processes in the form of MapReduce jobs under the hood
- MapReduce
  - A parallel programming model.
  - Defines the logic to process data on multiple machines
  - Batch processing operations on files in HDFS

- Usually written in Java using the Hadoop MapReduce library

HIVE QL

- Hive Query Language
  - A SQL-like interface to the underlying data.
  - Modeled on the Structured Query Language (SQL)
  - Familiar to analysts and engineers
  - Has simple query constructs
    - select
    - group by
    - join
- Hive exposes files in HDFS in the form of tables to the user
- Write SQL-like query in HiveQL and submit it to Hive
- Hive will translate the query to MapReduce tasks and run them on Hadoop
- MapReduce will process files on HDFS and return results to Hive

Hive metastore

- A Hive user sees data as if they were stored in tables
- Exposes the file-based storage of HDFS in the form of tables

- It is the bridge between data stored in files and the tables exposed to users
- Stores metadata for all the tables in Hive
- Maps the files and directories in Hive to tables
- Holds table definitions and the schema for each table
- Has information on converting files to table representations
- Any database with a JDBC driver can be used as a metastore
- Development environments use the built-in Derby database i.e. Embedded metastore

Hive vs. RDBMS

HIVE	RDBMS
Large datasets	Small datasets
Parallel computations	Serial computations
High latency	Low latency
Read operations	Read/write operations
Not ACID compliant by default	ACID compliant
HiveQL	SQL

HIVE	RDBMS
ϖ Large datasets δ Gigabytes or petabytes δ Calculating Trends	ϖ Small datasets δ Megabytes or gigabytes δ Accessing and updating individual records
ϖ Parallel Computations δ Distributed system with multiple machines δ Semi-structured data files partitioned across machines δ Disk space cheap, can add space by adding machines	ϖ Serial Computations δ Single computer with backup δ Structured data in tables on one machine δ Disk space expensive on a single machine
ϖ High Latency δ Records not indexed, cannot be accessed quickly δ Fetching a row will run a MapReduce that might take minutes	ϖ Low Latency δ Records indexed, can be accessed and updated fast δ Queries can be answered in milliseconds or microseconds
ϖ Read Operations	ϖ Read Operations

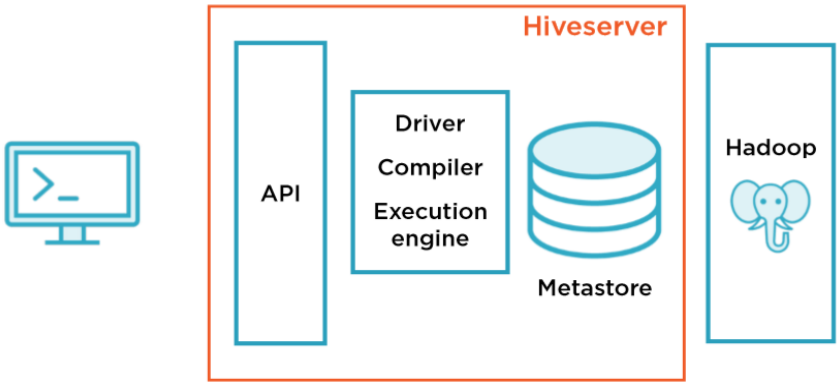
δ Not the owner of data. ₪ Hive stores files in HDFS ₪ Hive files can be read and written by many technologies ⇒ Hadoop, Pig, Spark ₪ Hive database schema cannot be enforced on these files	δ Schema-on-read ₪ Number of columns, column types, constraints specified at table creation ₪ Hive tries to impose this schema when data is read ₪ It may not succeed, may pad data with nulls ϖ Read/ Write Operations δ Sole gatekeeper for data Schema on write
ϖ Not ACID Compliant δ Data can be dumped into Hive tables from any source	ϖ ACID Compliant δ Only data which satisfies constraints are stored in the database

HIVEQL	SQL
Schema on read, no constraints enforced	Schema on write keys, not null, unique all enforced
Minimal index support	Indexes allowed
Row level updates, deletes as a special case	Row level operations allowed in general
Many more built-in functions	Basic built-in functions
Only equi-joins allowed	No restriction on joins
Restricted subqueries	Whole range of subqueries

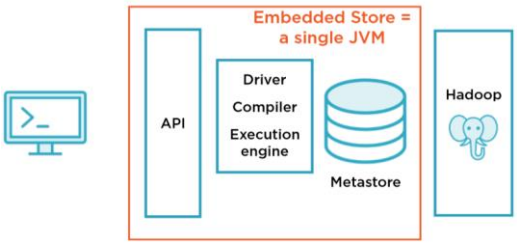
Basic HIVE BUILDING BLOCKS

- ϖ There is a terminal window on which the queries are written that look exactly like MySQL.
- ϖ This talks to the Hiveserver.

- The Hiveserver consults the metastore for the Hive table information.
- It passes these queries, converts them to MapReduce jobs and submits them to the underlying Hadoop ecosystem where they are run and the results are returned.



- Within Hive, the API is the gateway to the external world.
  - This allows various clients to connect to Hive and run queries.
- The combination of the driver, compiler and execution engine is the heart and soul of Hive.
  - It accepts queries through its API, parses it into the right form using metastore information, converts it to individual jobs or tasks that can be executed on Hadoop.
- The metastore is also part of Hive and it contains all table metadata information, the table schema, the column data types, how records from Hive can be serialized and deserialized onto HDFS.
- All together these components form the Hiveserver.
- The Hiveserver passes these MapReduce jobs onto Hadoop in order to be run on a cluster of machines.
- Connecting to the API component of the Hiveserver is done by the command line interface.
- Its called an embedded store when it is run on a single machine.



- The latest versions of Hive run a rewritten server called Hiveserver2
- Hiveserver2 supports concurrent clients and better authentication and authorization
- Beeline is a command line interface which works with this new server

Commands

Command	Explanation	Output

show databases;	Shows the databases existing within Hive.	`hive> show databases;`
		`OK`
		`default`
		`1 row selected (3.61 seconds)`

create database <database_name>;	Create a database and store it with the name given.	`hive> create database MCA;`
		`OK`
		`No rows affected (1.483 seconds)`
		
		`hive> show databases;`
		`OK`
		`default`
		`mca`
		`2 rows selected (0.129 seconds)`

use <database_name>;	Use the database	`hive> use MCA;`
		`OK`
		`No rows affected (0.065 seconds)`

create table <table_name> (<fieldname> <datatype>, ……..);	Create a table	`hive> create table customers (`
		`. . > id bigint,`
		`. . > name string,`
		`. . > address string`
		`. . > );`
		`OK`
		`No rows affected (4.583 seconds)`

create table if not exists <tablename> (<fieldname> <datatype>, …….);	To create a new table only if it does not exist.	`hive> create table if not exists orders (`
		`. . > id bigint,`
		`. . > product_id string,`
		`. . > cust_id bigint,`
		`. . > quantity int,`
		`. . > amt double`
		`. . > );`
		`OK`
		`No rows affected (3.022 seconds)`

show tables;	Show all tables in the current database.	`hive> show tables;`
		`OK`
		`customers`
		`1 row selected (0.757 seconds)`

describe <tablename>;	Shows the schema the table has.	`hive> describe customers;`
		`OK`
		`id bigint`
		`name string`
		`address string`
		`3 rows selected (0.703 seconds)`

insert into table <tablename> values (<value1>, <value2>, ……..);	Insert data into tables. Rarely used since you load data into tables from some file or sosme directory in HDFS.	
`hive> insert into table customers values (1, "aaa","bbb");`		
To insert many values at the same time		
`hive> insert into customers values (112, "AAA","ZZZ"), (113,"BBB","YYY"),(114,"CCC","XXX"),(178, "DDD","WWW");`		

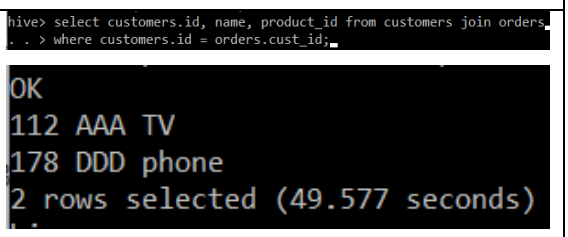
select * from <tablename>;	To see all the values	`hive> select * from customers;`
		`OK`
		`1 aaa bbb`
		`112 AAA ZZZ`
		`113 BBB YYY`
		`114 CCC XXX`
		`178 DDD WWW`
		`5 rows selected (0.821 seconds)`

select <columnname> from <tablename>	To see a particular column value	`hive> select name from customers;`
		`OK`
		`aaa`
		`AAA`
		`BBB`
		`CCC`
		`DDD`
		`5 rows selected (0.828 seconds)`

select <columname> from <tablename> where <condition>	To show a particular column's value based on a condition.	`hive> select name from customers where address = "ZZZ";`
		`OK`
		`AAA`
		`1 row selected (1.303 seconds)`
		
		`hive> select * from customers where address = "ZZZ";`
		`OK`
		`112 AAA ZZZ`
		`1 row selected (0.779 seconds)`
		
		`hive> select name from customers where address = "ZZZ" and id >100;`
		`OK`
		`AAA`
		`DDD`
		`EEE`
		`3 rows selected (3.007 seconds)`

select distinct <columname> from <tablename>;	To show unique values of a particular	`hive> select DISTINCT address from customers;`

	column.	WWW XXX YYY ZZZ bbb 5 rows selected (73.549 seconds)
select <columnname> from <tablename> order by <columname>;	To order the result based on a particular column name.	hive> select name, address from customers order by address; OK DDD WWW CCC XXX BBB YYY EEE ZZZ DDD ZZZ AAA ZZZ aaa bbb 7 rows selected (60.537 seconds)
select count(*) from <tablename>;	To find the total number of records in the table.	hive> select count(*) from customers; OK 7 1 row selected (57.382 seconds)
select sum(<columnname>) from <tablename>;	To sum a column that contains numbers.	hive> select sum(id) from customers; OK 750 1 row selected (52.21 seconds)
select <aggregate(<columname>)> from <tablename>;	To apply any aggregate function on the table.	hive> select max(id) from customers; OK 178 1 row selected (55.873 seconds)
select * from <tablename> limit <n>;	To display first 'n' rows of a table.	hive> select * from customers limit 2; OK 1 aaa bbb 112 AAA ZZZ 2 rows selected (0.764 seconds)
select <columname>, <aggregate(<columname>)> from <tablename> group by <columname>; To group the records of a table based on a particular column.	hive> select address, count(*) from customers group by address; OK WWW 1 XXX 1 YYY 1 ZZZ 3 bbb 1 5 rows selected (45.107 seconds)	

All select commands are MapReduce jobs except select * since it is a simple read operation on HDFS.		
select * from <table1name> join <table2name> where <table1name>.<columname> = <table2name>.<columname> ;	Join two tables based on equijoin.	
Hive only supports equi-joins where the 'where' clause on the joined column is an equality clause. It does not support natural joins.		

HIVE DATA TYPES

- Primitive Data types
  - Boolean
  - Numeric
  - String
  - Timestamp

Boolean	Numeric	
ϖ True or false ϖ Yes /no questions	ϖ Integers δ Tinyint: 1 byte, range -128 to 128 δ Smallint: 2 bytes, range -2^{15} to $2^{15} - 1$ δ Int: 4 bytes, range -2^{31} to $2^{31} - 1$ δ Bigint: 8 bytes, range -2^{63} to $2^{63} - 1$	ϖ Decimals δ Float: 4 bytes δ Double: 8 bytes δ Decimal: Arbitrary precision - dec(10, 2) – will occupy 10 digits of space and after the decimal point two spaces

String	Timestamp
ϖ String: Unbounded, variable length character string δ Char: Fixed length character string δ Varchar: Bounded,	ϖ Integers: Unix timestamp in nanoseconds ϖ Floats: Unix timestamp in seconds with decimal precision ϖ String: JDBC compliant "YYYYMM-DD HH:MM:SS.fffffffff"

variable length character string	ϖ Dates Values of the form "YYYY-MM-DD"

Storing Data in HIVE

- ϖ Data
  - δ The records in the table which holds the actual data
  - δ Stored in HDFS, the reliable storage for data in Hadoop
  - δ Files partitioned across multiple machines in the cluster
  - δ Stored in directories under Hive's warehouse directory
  - δ hive.metastore.warehouse.dir property in hive-site.xml
  - δ Defaults to /user/hive/warehouse

```
C:\hive\apache-hive-2.1.0-bin>hadoop fs -ls /user/hive/warehouse/mca.db
Found 2 items
drwxr-xr-x   - raoal supergroup          0 2019-11-13 18:08 /user/hive/warehouse/mca.db/customers
drwxr-xr-x   - raoal supergroup          0 2019-11-13 23:04 /user/hive/warehouse/mca.db/orders
```

- ϖ Metadata
  - δ Information about the underlying data in the table
  - δ Metastore, acts as a bridge between Hive and files in HDFS
  - δ A relational database with information on
    - ₪ databases, tables
    - ₪ columns, owners, storage, serialization/ deserialization information
    - ₪ user supplied metadata

HIVE TABLES

- ϖ Managed
  - δ Data managed by Hive and stored in the warehouse directory
- ϖ External
  - δ Data not fully managed by Hive and exists outside the warehouse directory
- ϖ The metadata for both is in the metastore

Managed Tables

- ϖ All tables so far have been managed tables
- ϖ Hive owns the files and directories
- ϖ These can be modified by other technologies
- ϖ Deleting a managed table deletes both data and metadata

External tables

- Share the underlying data across other technologies
- Hadoop, Pig, HBase all of these may access and edit those files
- Deleting an external table deletes only the metadata

Demo

Use products.csv	It contains id, name and cost of a product.	iphone7, iPhone 7, 950 camera_canon, Canon 570x, 1000 washingmachine_samsung, Samsung Swift, 400 tv_vu, Vu 56 Inch, 600
hadoop fs -mkdir /<directoryname>/	Create a directory in Hadoop to store this products.csv	`C:\WINDOWS\system32>hadoop fs -mkdir /data`

hadoop fs -copyFromLocal <filename> /<directoryname>/	Copy the file products.csv to the hdfs directory created	`C:\hive\apache-hive-2.1.0-bin>hadoop fs -copyFromLocal products.csv /data/`
		`C:\hive\apache-hive-2.1.0-bin>hadoop fs -ls /data`
		`Found 1 items`
		`-rw-r--r--  1 raoal supergroup  123 2019-11-15 12:49 /data/products.csv`

create external table if not exists <tablename> (<fieldname> <datatype>,……) location '/<directoryname>/'	Create the external table .	`hive> create external table if not exists products (`
		`. . > id string,`
		`. . > title string,`
		`. . > cost float`
		`. . > )`
		`. . > comment "Table to store information sold in stores"`
		`. . > location '/data/';`
		`OK`
		`No rows affected (71.257 seconds)`
		
		`hive> describe products;`
		`OK`
		`id string`
		`title string`
		`cost float`
		`3 rows selected (7.26 seconds)`

```
hive> show tables;
OK
customers
orders
products
values__tmp__table__2
4 rows selected (2.267 seconds)
```

- The external keyword indicates that the data for this table is not managed by Hive.
  - The data for this table exists in an HDFS directory specified by the location keyword.
- Make sure that the order in which the columns are specified in the table matches the order in which the data is available in the csv file.
- Even though the data has not been inserted when a select command is used all the data that was available in the csv file is available.

Hive is schema-on-read.

For the schema that has been defined for the table, hive will try to impose the schema on the underlying data which is referenced in HDFS.

All the individual records have been squished into one column.

```
hive> select * from products;
OK
 iphone7, iPhone 7, 950
 camera_canon, Canon 570x, 1000
 washingmachine_samsung, Samsung Swift, 400
 tv_vu, Vu 56 Inch, 600


6 rows selected (8.542 seconds)
```

hadoop fs –ls /user/hive/warehouse/mca.db

Check the contents of the warehouse by using the command

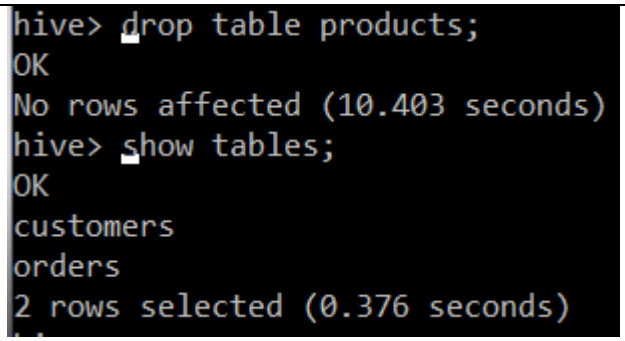
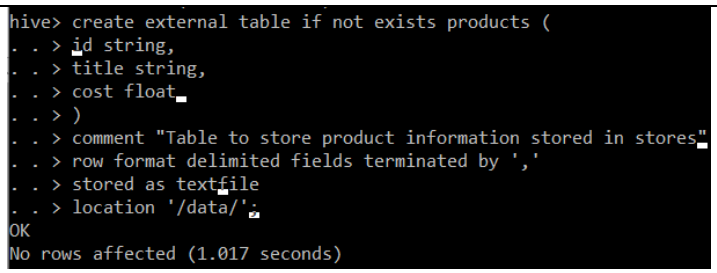
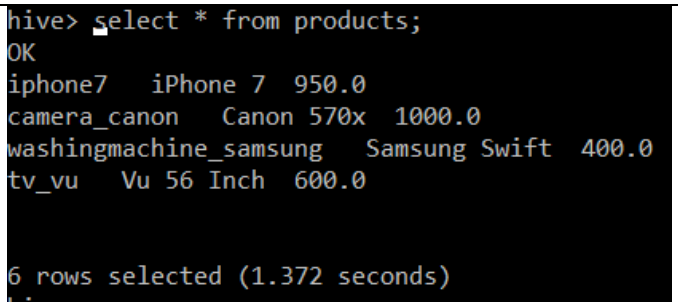
```
C:\hive\apache-hive-2.1.0-bin>hadoop fs -ls /user/hive/warehouse/mca.db
Found 2 items
drwxr-xr-x   - raoal supergroup          0 2019-11-13 18:08 /user/hive/warehouse/mca.db/customers
drwxr-xr-x   - raoal supergroup          0 2019-11-13 23:04 /user/hive/warehouse/mca.db/orders
```

It shows that a directory has been created for the managed tables of customers and orders but not for products.

The product table does not live within the Hive warehouse since it is an external table.

The show tables show the table.

The metadata for the table is present in the metastore and the actual data is not in the Hive but in the /data HDFS location

drop table products;	Drop the table	
The products.csv is still present in the Hadoop folder. Dropping the table has not deleted the underlying data. This is one of the main differences between managed and external tables in hive.		
row format delimited fields terminated by '<symbol>'	To see that the data is correctly stored in different columns use the command	
The format of every row is told to Hive. The row format is delimited fields where each field is terminated by a comma, since it is a comma separated file. Hive considers the default delimiter to be the ASCII value \001.		
stored as <option>	Specify the kind of file being read.	
The default is textfile so the stored as textfile is optional.		
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable		
select * from <tablename>;	Check the contents	

describe formatted <tablename>;	To check whether a table is managed or external use	hive> describe formatted customers; OK # col_name data_type comment id bigint name string address string # Detailed Table Information Database: mca Owner: raoal CreateTime: Tue Nov 12 09:50:45 IST 2019 LastAccessTime: UNKNOWN Retention: 0 Location: hdfs://0.0.0.0:19000/user/hive/warehouse/mca.db/customers Table Type: MANAGED_TABLE Table Parameters: COLUMN_STATS_ACCURATE {\"BASIC_STATS\":\"true\"} numFiles 3 numRows 7 rawDataSize 75 totalSize 82 transient_lastDdlTime 1573648726 hive> describe formatted products; OK # col_name data_type comment id string title string cost float # Detailed Table Information Database: mca Owner: raoal CreateTime: Fri Nov 15 20:50:50 IST 2019 LastAccessTime: UNKNOWN Retention: 0 Location: hdfs://0.0.0.0:19000/data Table Type: EXTERNAL_TABLE Table Parameters: EXTERNAL TRUE comment Table to store product information stored in stores numFiles 1 totalSize 123 transient_lastDdlTime 1573831250 # 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
like <tablename>	can be used with create command to create a table that is exactly like another table that exists	hive> create table if not exists fresh_products like products; OK No rows affected (1.591 seconds) hive> describe fresh_products; OK id string title string cost float 3 rows selected (1.326 seconds) hive> select * from fresh_products; OK No rows selected (2.001 seconds)

alter table <tablename> rename to <newname>;	To rename a table use the rename command with alter command.	hive> alter table fresh_products . . > rename to freshprods; OK No rows affected (3.6 seconds) hive> show tables; OK customers freshprods orders products 4 rows selected (0.358 seconds)
alter table <tablename> add columns (<fieldname> <datatype>);	To add columns use the add columns in the alter command.	hive> alter table freshprods add columns (. . > expiry_date date . . > comment "Expiry date of fresh produce" . . >); OK No rows affected (0.565 seconds) hive> describe freshprods; OK id string title string cost float expiry_date date Expiry date of fresh produce 4 rows selected (0.214 seconds)
alter table <tablename> change column <oldcolumname> <old/newcolumname> <datatype> after <columname>	To move one column's position use change with after. change column. columns.	hive> alter table freshprods . . > change column id id string . . > after title; OK No rows affected (0.496 seconds) hive> describe freshprods; OK title string id string cost float expiry_date date Expiry date of fresh produce 4 rows selected (0.152 seconds)

Moving columns does not move the underlying data in the	hive> alter table products . . > change column id id string . . > after title; OK No rows affected (0.184 seconds) hive> describe products; OK title string id string cost float 3 rows selected (0.278 seconds) hive> select * from products; OK iphone7 iPhone 7 950.0 camera_canon Canon 570x 1000.0 washingmachine_samsung Samsung Swift 400.0 tv_vu Vu 56 Inch 600.0 6 rows selected (1.134 seconds)

Temporary tables

- Tables created within a Hive session, deleted when the session ends
- They are active within the Hive session and as soon as that session ends, the table and all its contents are deleted.
- Store temporary data
- Tables of the same name can be created by different users
- Do not support partitions and indexes
- Can have the same name as a permanent table.
  - As long as temporary table exists the permanent table cannot be accessed.
  - The temporary table hides the permanent table if it has the same name.

Demo

Check the tables that exist in the database		hive> show tables; OK customers freshprods orders products 4 rows selected (14.34 seconds)
create temporary table <tablename>	Create temporary tables using the key word temporary	hive> create temporary table test_customers like customers; OK No rows affected (7.389 seconds) hive> describe test_customers; OK id bigint name string address string 3 rows selected (1.016 seconds) hive> show tables; OK customers freshprods orders products test_customers 5 rows selected (1.375 seconds)
insert into <tablename> values <v1, v2, v3>	Insert into the temporary table	hive> insert into test_customers values (189, "ACAC", "VVV");
		hive> select * from test_customers; OK 189 ACAC VVV 1 row selected (2.984 seconds)
quit; hive	Quit the session and restart hive again.	hive> show tables; OK customers freshprods orders products 4 rows selected (0.053 seconds)
The temporary table no longer exists.		
	Create another table name with	hive> create temporary table customers like orders; OK No rows affected (1.238 seconds)

	same name as permanent table.	
	The original customers table has been hidden.	hive> show tables; OK customers freshprods orders products 4 rows selected (0.168 seconds) hive> describe customers; OK id bigint product_id string cust_id bigint quantity int amt double 5 rows selected (0.045 seconds)
	Quit hive and restart it. Use the mca database and see the tables.	hive> show tables; OK customers freshprods orders products 4 rows selected (1.823 seconds) hive> describe customers; OK id bigint name string address string 3 rows selected (1.058 seconds)

Inserting data

- Standalone
  - Insert individual records
  - Files
  - Other tables

Demo

- Insert using files

<table>
<tr><td></td><td>Create a freshproducts.csv and store it in bin directory of hive.</td><td>broccolli Broccoli 5
spinach Spinach 7
carrot Local Carr 4
potato Idaho Pot 4
milk Skimmed M 5
cheese Spicy Garli 5</td></tr>
<tr><td>alter table <tablename> change column <oldcolumname> <old/newcolumname> <datatype> after <columname>;</td><td>Alter the table freshprods</td><td>hive> alter table freshprods change column
. . > title title string after id;
OK
No rows affected (1.58 seconds)</td></tr>
<tr><td>describe <tablename>;</td><td>Check the schema of freshprods</td><td>hive> describe freshprods;
OK
id string
title string
cost float
expiry_date date Expiry date of fresh produce
4 rows selected (0.184 seconds)</td></tr>
<tr><td>load data local inpath <'filename.csv'> into table <tablename></td><td>Load the data from freshproducts.csv into fresprods.</td><td>hive> load data local inpath 'freshproducts.csv'
. . > into table freshprods;
Loading data to table mca.freshprods
OK
No rows affected (3.898 seconds)</td></tr>
<tr><td>select * from <tablename></td><td>Check the contents of the table.</td><td>hive> select * from freshprods;
OK
broccolli Broccoli 5.0
spinach Spinach 7.0
carrot Local Carrots 4.0
potato Idaho Potatoes 4.0
milk Skimmed Milk 5.0
cheese Spicy Garlic 5.0
6 rows selected (4.297 seconds)</td></tr>
<tr><td colspan="3">Using this command a copy is made into the Hive warehouse directory if this is a managed table.
The contents of the csv file has been copied into Hive's warehouse.
The keyword local in indicates that the file is present on the local file system and not on HDFS.</td></tr>
<tr><td>hadoop fs -ls /user/hive/warehouse/ <database>.db</td><td>The path of the file is the hive installation directory.</td><td>C:\BigData>hadoop fs -ls /user/hive/warehouse/mca.db
Found 3 items
drwxr-xr-x - raoal supergroup 0 2019-11-13 18:08 /user/hive/warehouse/mca.db/customers
drwxr-xr-x - raoal supergroup 0 2019-11-19 13:04 /user/hive/warehouse/mca.db/freshprods
drwxr-xr-x - raoal supergroup 0 2019-11-13 23:04 /user/hive/warehouse/mca.db/orders</td></tr>
</table>

hadoop fs -cat /user/hive/warehouse/ <database>.db/ <tablename>/*	Check the contents	C:\BigData>hadoop fs -cat /user/hive/warehouse/mca.db/freshprods/* broccolli, Broccoli,5 spinach, Spinach,7 carrot, Local Carrots,4 potato, Idaho Potatoes,4 milk,Skimmed Milk,5 cheese,Spicy Garlic,5
hadoop fs -copyFromLocal <filename>.csv /<hdfs directory>/ hadoop fs -ls /<hdfs directory>/	Copy this file into HDFS and check whether it is copied or not.	C:\hive\apache-hive-2.1.0-bin\bin>hadoop fs -copyFromLocal freshproducts.csv /data/ C:\hive\apache-hive-2.1.0-bin\bin>hadoop fs -ls /data/ Found 2 items -rw-r--r-- 1 raoal supergroup 138 2019-11-20 08:15 /data/freshproducts.csv -rw-r--r-- 1 raoal supergroup 123 2019-11-15 12:49 /data/products.csv
load data inpath '/<hdfs directory>/<filename.csv>' into table <tablename>;	Load the same data again but from the HDFS directory.	hive> load data inpath '/data/freshproducts.csv' into table freshprods; Loading data to table mca.freshprods OK No rows affected (10.98 seconds)
	Check the contents using select command.	hive> select * from freshprods; OK broccolli Broccoli 5.0 spinach Spinach 7.0 carrot Local Carrots 4.0 potato Idaho Potatoes 4.0 milk Skimmed Milk 5.0 cheese Spicy Garlic 5.0 broccolli Broccoli 5.0 spinach Spinach 7.0 carrot Local Carrots 4.0 potato Idaho Potatoes 4.0 milk Skimmed Milk 5.0 cheese Spicy Garlic 5.0 12 rows selected (2.058 seconds)
The new data is appended to the existing table.		
hadoop fs -ls /user/hive/warehouse/ <database>.db/ <tablename>	Explore the warehouse to see how things have changed.	C:\hive\apache-hive-2.1.0-bin\bin>hadoop fs -ls /user/hive/warehouse/mca.db/freshprods Found 2 items -rwxr-xr-x 1 raoal supergroup 138 2019-11-19 13:04 /user/hive/warehouse/mca.db/freshprods/freshproducts.csv -rwxr-xr-x 1 raoal supergroup 138 2019-11-20 08:15 /user/hive/warehouse/mca.db/freshprods/freshproducts_copy_1.csv
hadoop fs -ls /<hdfs directory>/		C:\hive\apache-hive-2.1.0-bin\bin>hadoop fs -ls /data/ Found 1 items -rw-r--r-- 1 raoal supergroup 123 2019-11-15 12:49 /data/products.csv
The file has been moved from the HDFS directory to the warehouse directory.		

If data is loaded from a file on the local file system a copy will be made in Hive's warehouse. If a file from HDFS is loaded into the table it will be moved to Hive's warehouse directory.		
load data local inpath '<filename.csv>' overwrite into table <tablename>;	To overwrite existing data in a table	hive> load data local inpath 'freshproducts.csv' . . > overwrite into table freshprods; Loading data to table mca.freshprods OK No rows affected (12.671 seconds)
	Check the contents of the table	hive> select * from freshprods; OK broccolli Broccoli 5.0 spinach Spinach 7.0 carrot Local Carrots 4.0 potato Idaho Potatoes 4.0 milk Skimmed Milk 5.0 cheese Spicy Garlic 5.0 6 rows selected (2.181 seconds)

- Inserting from other tables
  - Data can be loaded into Hive from other tables too.

insert into table <tablename> select <field1>, <field2>,…. from <tablename>	Merge the data of both products and freshprods table.	hive> insert into table products . . > select id, title, cost from freshprods;
The overwrite keyword can also be used with this insert command to completely remove the contents of the existing table.		
	Check the contents	hive> select * from products; OK broccolli Broccoli 5.0 spinach Spinach 7.0 carrot Local Carrots 4.0 potato Idaho Potatoes 4.0 milk Skimmed Milk 5.0 cheese Spicy Garlic 5.0 iphone7 iPhone 7 950.0 camera_canon Canon 570x 1000.0 washingmachine_samsung Samsung Swift 400.0 tv_vu Vu 56 Inch 600.0 12 rows selected (1.286 seconds)

ϖ Load multiple tables from single table.

describe <tablename>; alter <tablename> change column <field> <field> <datatype> after <field>;	Use the products table as source. Alter the table such that the id comes before title.	hive> describe products; OK title string id string cost float 3 rows selected (0.975 seconds) hive> alter table products . . > change column title title string . . > after id; OK No rows affected (1.806 seconds)
create table <tablename> (<field1> <datatype>, ……..);	Create a new table called prod_name that has id and name as its attributes.	hive> create table prod_name (id string, name string); OK No rows affected (10.703 seconds)
	Create another table prod_cost that has id, cost as its attributes.	hive> create table prod_cost (id string, cost float); OK No rows affected (0.661 seconds)
from <sourcetable> insert overwrite table <table1> select <field1>,…. insert into table <table2> select <field1>,…	To populate both the tables with one insert statement	hive> from products . . > insert overwrite table prod_name . . > select id, title . . > insert into table prod_cost . . > select id, cost;
	Check the contents	hive> select * from prod_cost; OK broccolli 5.0 spinach 7.0 carrot 4.0 potato 4.0 milk 5.0 cheese 5.0 iphone7 950.0 camera_canon 1000.0 washingmachine_samsung 400.0 tv_vu 600.0 12 rows selected (1.048 seconds)

		hive> select * from prod_name; OK broccolli Broccoli spinach Spinach carrot Local Carrots potato Idaho Potatoes milk Skimmed Milk cheese Spicy Garlic iphone7 iPhone 7 camera_canon Canon 570x washingmachine_samsung Samsung Swift tv_vu Vu 56 Inch 12 rows selected (0.936 seconds)
truncate table <tablename>;	Hive uses truncate command to delete all data within a table.	hive> truncate table freshprods; OK No rows affected (0.811 seconds)
	Check the contents	hive> select * from freshprods; OK No rows selected (1.544 seconds)

Deleting and updating data in hive

- Hive tables do not support row level deletes and updates by default
- It is possible to get ACID compliant* Hive tables by setting up special properties in hive-site.xml

Partitioning and bucketing of data

- Splits data into smaller, manageable parts
- Enables performance optimizations

- **Partitioning**
  - Data may be naturally split into logical units
  - Example
    - Customers portioned based on where they lived.
  - Each of these units will be stored in a different directory
  - State specific queries will run only on data in one directory
  - Splits may not of the same size

Bucketing

- δ Size of each split should be the same
- δ Hash of a column value - address, name, timestamp anything
- δ Each bucket is a separate file
- δ Makes sampling and joining data more efficient

Complex data types

- ϖ Array
- ϖ Map
- ϖ Struct
- ϖ Union – Rarely uses, incomplete support in Hive

Array

- ϖ Collection data type
- ϖ No fixed size
- ϖ Entities of the same type
- ϖ Only arrays of primitive types allowed

Demo

create table <tablename> (<field1> array<datatype>….);	Create tables with arrays.	hive> create table mobilephones (. . > id string, . . > title string, . . > cost float, . . > colors array<string>, . . > screen_size array<float> . . >);
	Check the schema	hive> describe mobilephones; OK id string title string cost float colors array<string> screen_size array<float> 5 rows selected (1.761 seconds)
insert into table <tablename> select “stringvalue”, number, array(“stringvalue”,…..),	Insert data into table with arrays	hive> insert into table mobilephones . . > select "redminote7", "Redmi Note 7", 300, . . > array("white", "silver", "black"), array(float(4.5)) . . > UNION ALL . . > select "motoGplus", "Moto G Plus", 200, array("black", "gold"), . . > array(float(4.5), float(5.5));

array(<datatype>(number),.....);		
	To select individual arrays	hive> select id, colors from mobilephones; OK
Select arrayname[<index>];	To select the first element of the colors array	hive> select id, colors[0] from mobilephones; OK
create table <tablename>(<field1> <datatype>,..... <arrayname> array<datatype>, ...) row format delimited fields terminated by ',' collection items terminated by '#';	Create a table from a file.	hive> create table mobilephones (. . > id string, . . > title string, . . > cost float, . . > colors array<string>, . . > screen_size array<float> . . >) . . > row format delimited fields terminated by ',' . . > collection items terminated by '#'; OK No rows affected (2.299 seconds)
To identify the collection items, use the keyword collection with the termination value of #.		
	The file contents are	samsungj7 Samsung J7 250 red#blue#black 5.5 oneplusthree One Plus Three 450 gold#silver 4.5#5.5
load data local inpath <'filenames.csv'> into table <tablename>;	Load the data using the load command.	hive> load data local inpath 'mobilephones.csv' . . > into table mobilephones; Loading data to table mca.mobilephones OK No rows affected (2.556 seconds)
select * from <tablename>;	To check if it is loaded or not	hive> select * from mobilephones; OK samsungj7 Samsung J7 250.0 [" red","blue","black"] [5.5] oneplusthree One Plus Three 450.0 [" gold","silver"] [4.5,5.5] 2 rows selected (1.782 seconds)

Map

- Unordered collection of pairs
- No fixed size
- Every entity is a key, value pair
- Value is accessed using a unique key
- Keys and values have their own data types
  - Key can be integer, value can be a struct
  - Key can be a string, value can be an integer.

Demo

<table>
<tr>
<td>create table
<tablename>
(
<field1>
<datatype>,
……..
<fieldn>
map<<keydatatype>,
<valuedatatype>>)
row format delimited fields terminated by ‘,’
collection items terminated by ‘#’
map keys terminated by ‘:’;</td>
<td>Create the table with the map data type.
The key is of type string
The value is of type boolean
The map keys are delimited by :.</td>
<td>

```
hive> create table mobilephones (
. . . > id string,
. . . > title string,
. . . > cost float,
. . . > colors array<string>,
. . . > 
. . . > screen_size array<float>,
. . . > features map<string, boolean>,
. . . > )
. . . > row format delimited fields terminated by ','
. . . > collection items terminated by '#'
. . . > map keys terminated by ':';
OK
No rows affected (2.467 seconds)
```

</td>
</tr>
<tr>
<td>The file contains the following data</td>
<td>

samsungj7	Samsung J7	250	red#blue#black	5.5	camera:true#dualsim:false
oneplusthree	One Plus Three	450	gold#silver	4.5#5.5	autofocus:true

</td>
</tr>
</table>

load data local inpath <'filename.csv'> into table <tablename>;	Load the data	hive> load data local inpath 'mobilephones.csv' . . > into table mobilephones; Loading data to table mca.mobilephones OK No rows affected (1.424 seconds)
Arrays are identified by square brackets; map is identified by curly braces	hive> select * from mobilephones; OK samsungj7 Samsung J7 250.0 [" red","blue","black"] [5.5] {"camera":true,"dualsim":false} oneplusthree One Plus Three 450.0 [" gold","silver"] [4.5,5.5] {"autofocus":true} 2 rows selected (0.843 seconds)	
select <mapname> from <tablename>;	Accessing maps	hive> select id, features from mobilephones; OK samsungj7 {"camera":true,"dualsim":false} oneplusthree {"autofocus":true} 2 rows selected (3.285 seconds)

Struct

- Logical grouping of data
- Can have different data types
- Can hold any number of values
- Each value referenced by a name

Demo

create table <tablename>(<field> <datatype>,….. <field> struct<<field>:<datatype>, <field>: <datatype>,…..> row format delimited fields terminated by '\,';	Create the table mobilephones Information holds specific info about the phone. The info is battery life and kind of camera.	hive> create table mobilephones (. . > id string, . . > title string, . . > cost float, . . > colors array<string>, . . > screen_size array<float>, . . > features map<string, boolean>, . . > information struct<battery:string, camera:string> . . >) . . > row format delimited fields terminated by ',', . . > collection items terminated by '#' . . > map keys terminated by ':'; OK No rows affected (0.805 seconds)
The file content is as	samsungj7 Samsung J7 250 red#blue#black 5.5 camera:true#dualsim:false 24 hours#2MP oneplusthree One Plus Three 450 gold#silver 4.5#5.5 autofocus:true 12 hours	

shown. The first data value goes to the battery field and second goes to the camera field.		
describe <tablename>;	Check the schema of the table.	hive> describe mobilephones; OK id string title string cost float colors array<string> screen_size array<float> features map<string,boolean> information struct<battery:string,camera:string> 7 rows selected (0.217 seconds)
load data local inpath <'filename.csv'> into table <tablename>;	Load the data	hive> load data local inpath 'mobilephones.csv' . . > into table mobilephones; Loading data to table mca.mobilephones OK No rows affected (1.558 seconds)
Display the data	hive> select * from mobilephones; OK samsungj7 Samsung J7 250.0 [" red","blue","black"] [5.5] {"camera":true,"dualsim":false} {"battery":"24 hours","camera":"2MP"} oneplusthree One Plus Three 450.0 [" gold","silver"] [4.5,5.5] {"autofocus":true} {"battery":"12 hours","camera":null} 2 rows selected (1.127 seconds)	
Select the map and structure details.	hive> select features, information from mobilephones; OK {"camera":true,"dualsim":false} {"battery":"24 hours","camera":"2MP"} {"autofocus":true} {"battery":"12 hours","camera":null} 2 rows selected (2.172 seconds)	
Select individual values in a struct.	hive> select features['camera'], information.battery . . > from mobilephones; OK true 24 hours 12 hours 2 rows selected (2.035 seconds)	

Built-in functions

- UDF
  - User-defined Functions
- UDAF
  - User-defined Aggregate Functions
- UDTF
  - User-defined Table-generating Functions

UDF

- Works on a single row
- Outputs a single row
  - trim(), concat(), length() round(), floor()
- https://cwiki.apache.org/confluence/display/Hive/languageManual+UDF

UDAF

- Works on multiple rows
- Outputs a single row
  - count(*), sum(), avg()

UDTF

- Works on a single row
- Outputs multiple rows
  - explode(), posexplode()

Table-generating functions

- explode()
  - Flatten the data in arrays and maps

Manager	SubordinateList
Larry	[Sundar, Eric, Jon]
Sergey	[Ruth, Urs]
Sundar	[Susan, Alan, Lazlo]

Subordinate
Sundar
Eric
John
Ruth
Urs
Susan
Alan
Lazlo

Employee	Details
Larry	{"office": "271B", "numReports": 8, "salary": 1}
Sergey	{"office": "271B", "numReports": 5, "salary": 1}
Sundar	{"office": "285", "numReports": 12}

Key	Value
office	271B
numReports	8
salary	1
office	271B
numReports	5
salary	1
office	285
numReports	12

Demo

select explode(<fieldname>) as <alias name> from <tablename>	Use mobilephones To explode the values of	`hive> select explode(colors) as variants from mobilephones;`

	colors array	OK red blue black gold silver 5 rows selected (4.227 seconds) hive> select colors from mobilephones; OK [" red","blue","black"] [" gold","silver"] 2 rows selected (2.621 seconds)
select explode(<fieldname>) as (<field1>, <field2>,....) from <tablename>;	Explode a map data type	hive> select explode(features) as (feature, present) . . > from mobilephones; OK camera true dualsim false autofocus true 3 rows selected (0.813 seconds)

- To see that every row of the flattened row is associated with an index value use posexplode.
- hive> select posexplode(colors) as (index, variants) from mobilephones;
- OK
  0 red
  1 blue
  2 black
  0 gold
  1 silver
  5 rows selected (0.825 seconds)