Curso de extensão em Data Science

GERÊNCIA DE INFRAESTRUTURA PARA BIG DATA

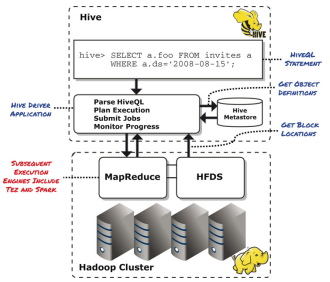
Prof. Tiago Ferreto – tiago.ferreto@pucrs.br

ESCOLA POLITÉCNICA

HIVE

Hive Introduction

- Started in 2010 at Facebook
- Goal: provide a high-level interface to Hadoop MapReduce using SQL-like
  - HiveQL (Hive Query Language) – implements a subset of SQL-92 with extensions
- Process
  - Hive parses the query in HiveQL and generates Java MR operations
  - MR are submitted to the Hadoop cluster
  - Hive monitors and returns results to client (or writes to HDFS)

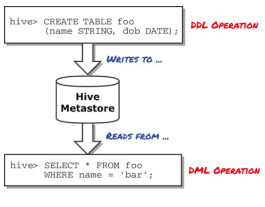


Hive Objects

- Hive implements a tabular abstraction to objects in HDFS
  - Directories and files are represented as tables with predefined columns with datatypes
  - Tables are created using SQL DDL (Data Definition Language) commands and accessed using SQL DML (Data Manipulation Language) statements
- Differences between Hive and a conventional database platform
  - UPDATE is implemented as a coarse-grained transformation
    - Since HDFS is an immutable filesystem, modifications to tables are not allowed
  - No transactions, no journaling, no rollbacks, no real transaction isolation level
  - No declarative referential integrity (DRI), no primary keys, no foreign keys
  - Incorrectly formatted data are represented to the client as NULL

Hive Metastore

- Mapping of tables to directory locations in HDFS, columns and definitions are maintained in Hive metastore
  - Hive Metastore is a relational database accessed by the Hive client
- Object definitions include input/output formats of files (tables) and serialization/deserialization functions to define how records and fields are extracted
- Hive Metastore is usually implemented as an embedded Derby database
  - But can also be implemented as a local or remote database (MySQL, PostgreSQL, etc)
- HCatalog - Hive subproject
  - Extend objects created in Hive to other projects (e.g., Apache Pig)



Hive CLI

- Accepts and parses HiveQL input commands
- Common method for performing ad hoc queries
- Used when the Hive client or driver application is deployed to the local machine, including the connection to the metastore

HiveServer2

- Client/server approach to use Hive
- Used in large-scale implementations
  - Connection details to the metastore are kept in the server and cluster access is controlled
- Can act as a multi-session driver application for multiple clients
- Provides a JDBC interface to be accessed by external clients (e.g., Beeline – lightweight CLI

Hive Batch Mode

- Hive also supports non-interactive or batch mode execution
- Example

```
# run all statements in a file
$ hive -f MyHiveQuery.hql
# run an individual statement
$ hive -e "SELECT * FROM mytable"
```


Hive Database and Tables

- Hive objects consist basically of databases and Tables
- Databases – used for organization, authorization, and namespace management
  - Default database is named "*default*"
- Tables – exist in a Hive database
- Database context can be changed using the USE statement
  - `hive> USE bikeshare;`
- Objects can referenced using fully qualified object identifier (database and table name)
  - `hive> SELECT * FROM bikeshare.trips;`

Hive Authorization

- Hive supports basic authorization
  - Determines access levels to objects within a Hive instance
  - Not enabled by default
  - Configured in hive-site.xml file

```
...
<property>
  <name>hive.security.authorization.enabled</name>
  <value>true</value>
</property>
<property>
  <name>hive.security.authorization.createtable.owner.grants</name>
  <value>ALL</value>
</property>
...
```

- Object-level privileges are assigned using the GRANT statement
  - Operations: SELECT, CREATE, ALTER, DROP, ALL, ...
  - Scope: GROUP, ROLE, USER
  - Revoking privileges: REVOKE statement
  - Examples:
    - `hive> GRANT SELECT ON TABLE trips TO USER javen;`
    - `hive> REVOKE SELECT ON TABLE trips FROM USER javen;`

Hive Objects

- Hive supports most common primitive datatypes, and also complex datatypes (struct, map, array)

Datatype	Category	Description
TINYINT	Numeric	1-byte signed integer (–128 to 127)
SMALLINT	Numeric	2-byte signed integer (–32,768 to 32,767)
INT	Numeric	4-byte signed integer
BIGINT	Numeric	8-byte signed integer
FLOAT	Numeric	4-byte single precision floating point
DOUBLE	Numeric	8-byte double precision floating point
DECIMAL(p,s)	Numeric	User definable precision and scale
TIMESTAMP	Date/Time	Unix timestamp
DATE	Date/Time	Date (formatted as YYYY-MM-DD)
STRING	String	Character sequence (variable length)
VARCHAR	String	Character sequence (variable length)
CHAR(n)	String	Character sequence (fixed length)
BOOLEAN	Misc	True or False
BINARY	Misc	Raw binary data

CREATE TABLE Statement

```
hive> CREATE TABLE stations
> (
> station_id INT,
> name STRING,
> lat DOUBLE,
> long DOUBLE,
> dockcount INT,
> landmark STRING,
> installation STRING
> )
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
```

```
hive> DESCRIBE stations;
OK
station_id              int
name                    string
lat                     double
long                    double
dockcount               int
landmark                string
installation            string
Time taken: 0.035 seconds, Fetched: 7 row(s)
```

Input/Output Formats and SerDes in Hive

- Hive uses an InputFormat and a SerDe (Serializer/Deserializer) to determine how to read input files and extract records for processing
  - Specified using the STORED AS directive in the CREATE TABLE statement
  - Example: STORED AS TEXTFILE corresponds to the TextInputFormat
- Other InputFormats: SEQUENCEFILE, ORC, PARQUET, AVRO

- Default SerDe used by Hive – LazySimpleSerDe
  - File is delimited by a character (printable or non-printable) indicated in the FIELDS TERMINATED BY clause (defined as Ctrl-A if not present)
- Other SerDe (specified using the ROW FORMAT SERDE directive)
  - RegexSerDe – schema for fixed-width data

- Default OutputFormat → HiveIgnoreKeyTextOutputFormat (used to write data to text files in Hadoop)
- Other OutputFormats: ORC, Parquet, Avro, SequenceFiles, …

Loading data into Hive

- Approaches to load data into tables
  - Put, move, or copy the file(s) into the HDFS directory for the table object
  - Use the Hive LOAD DATA INPATH command for data that exists in HDFS, which uses an underlying HDFS move function
  - Use the Hive LOAD DATA LOCAL INPATH command for data that does NOT EXIST in HDFS, which uses an underlying HDFS put function
- Examples
  - `hive> LOAD DATA INPATH '/bikeshare/stations' INTO TABLE stations;`

  - Use OVERWRITE to replace exiting contents
  - `hive> LOAD DATA INPATH '/bikeshare/stations'`
    `> OVERWRITE INTO TABLE stations;`

  - Data can be loaded during table creation process
  - `hive> CREATE TABLE stations_copy AS`
    `> SELECT * FROM stations;`

Analyzing data with Hive

- Hive Query Language (HiveQL) is based on the SQL-92 specification, with some additional Hive-specific functions
- HiveQL statements can span multiple lines and are terminated by a semicolon.
- Single line comments are supported using the double hyphen (--).
- Typical SQL semantics such as column lists and WHERE clauses are fully supported in Hive
- Example
  - `-- this is a comment`

```
hive> SELECT name, lat, long
> FROM stations
> WHERE landmark = 'San Jose';
```


Hive's Built-in Functions

- Hive includes several built-in functions to perform several operations (mathematical, string and date manipulation, etc)
  - Examples: ROUND, CEIL, FLOOR, RAND, SUBSTRING, LOWER, UPPER, RTRIM, LTRIM, CONCAT, TO_DATE, DAY, MONTH, YEAR
- Many functions are identical to the ones available in the SQL dialects in most popular RDBMS
- If a function is missing, additional UDFs (User Defined Functions) can be written (in Java)
- Examples
  - `hive> SELECT LOWER(name) FROM stations;`

  - DESCRIBE can be used to show help and usage for functions
  - `hive> DESCRIBE FUNCTION RAND;`
    `OK`
    `RAND([seed]) - Returns a pseudorandom number between 0 and 1`

Data grouping and aggregation

```
hive> SELECT landmark, COUNT(*) FROM stations
> GROUP BY landmark;
OK
Mountain View    7
Palo Alto        5
Redwood City     7
San Francisco    35
San Jose         16
Time taken: 19.443 seconds, Fetched: 5 row(s)
```


Joins in Hive

- Hive supports all common join types (INNER JOIN – default, LEFT OUTER JOIN, RIGHT OUTER JOIN and FULL OUTER JOIN)

```
hive> SELECT t.trip_id,
> t.duration, t.start_date, s.name
> FROM stations s
> JOIN trips t ON s.station_id = t.start_terminal;
```

Data Output with Hive

- Hive supports several methods to persist the output of a query to local or HDFS filesystem
  - INSERT OVERWRITE → sends the query results to another Hive table (a directory in HDFS), overwriting the existing contents
  - INSERT INTO TABLE → sends the query results to another Hive table, appending the output to the existing table (directory) contents
  - INSERT OVERWRITE DIRECTORY → saves the results to a directory in HDFS which may or may not be assigned to a Hive table
  - INSERT OVERWRITE LOCAL DIRECTORY → saves the results to a local directory, not to HDFS

```
hive> INSERT INTO TABLE trips_counts
> SELECT start_terminal , start_station, COUNT(*) AS count
> FROM trips
> GROUP BY start_terminal, start_station
> ORDER BY count
> DESC LIMIT 10;
```


Complex Datatypes in Hive

- Hive supports complex or nested datatypes
  - Common in XML, JSON, and other semi-structured data sources
- ARRAY Datatype
  - Stores an ordered list of elements of the same datatype
  - Uses COLLECTION ITEMS TERMINATED BY clause to determine the delimiter used for reading or deriving elements in the collection

```
hive> CREATE TABLE customers (
> id INT,
> fname STRING,
> lname STRING,
> email STRING,
> orderids ARRAY<INT>)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY '|'
> COLLECTION ITEMS TERMINATED BY ','
> ;
```

```
hive> SELECT orderids[0] FROM customers;
1
```


Complex Datatypes in Hive

- STRUCT Datatype
  - Consists of named fields that can be of the same of different datatypes
  - Uses angle brackets (<>) to associate datatypes to fields

```
hive> CREATE TABLE customers (
    > id INT,
    > fname STRING,
    > lname STRING,
    > email STRING,
    > orderids ARRAY<INT>,
    > email_preferences STRUCT<opcomms:boolean, promos:boolean>)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '|'
    > COLLECTION ITEMS TERMINATED BY ','
    > ;
hive> SELECT email_preferences.opcomms FROM customers;
TRUE
```


Complex Datatypes in Hive

- MAP Datatype
  - Represents key value pairs

```
hive> CREATE TABLE customers (
    > id INT,
    > fname STRING,
    > lname STRING,
    > email STRING,
    > orderids ARRAY<INT>,
    > email_preferences STRUCT<opcomms:boolean, promos:boolean>,
    > address_map MAP<STRING, STRING>)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '|'
    > COLLECTION ITEMS TERMINATED BY ','
    > MAP KEYS TERMINATED BY ':';
hive> SELECT address_map['city'] FROM customers;
Hayward
```

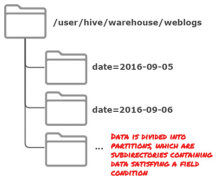

Query management and optimization in Hive

- Hive requires optimization in queries design to improve performance
  - Unlike a traditional relational database platform, Hive does not have indexes and statistics to optimize queries and data access
- An important factor for optimization is the amount of data read by a particular query
- Hive offers two key approaches used to limit or restrict the amount of data that a query needs to read
  - Partitions
  - Buckets

Partitions in Hive

- Used to divide data into subdirectories based upon one or more conditions (typically used in WHERE clauses)
- Used normally for coarse-grained date grouping
- Example

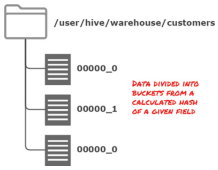
```
hive> CREATE TABLE weblogs (
    > ip STRING,
    > time_local STRING,
    > method STRING,
    > uri STRING,
    > protocol STRING,
    > status STRING,
    > bytes_sent STRING,
    > referer STRING,
    > useragent STRING)
    > PARTITIONED BY (date STRING);
```



Buckets in Hive

- Used to group items based upon a key hash

```
hive> CREATE TABLE customers
    > (cust_id INT,
    > fname STRING,
    > lname STRING,
    > email STRING
    > )
    > CLUSTERED BY (cust_id) INTO 3 BUCKETS;
```



EXPLAIN Statement

- Used to understand the query execution plan of the Hive query optimizer for a specific query

```
hive> EXPLAIN SELECT method, COUNT(*) FROM weblogs GROUP BY method;
STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-0 depends on stages: Stage-1
STAGE PLANS:
Stage: Stage-1
Map Reduce
Map Operator Tree:
TableScan
alias: weblogs
...
```

HIVE – HANDS-ON