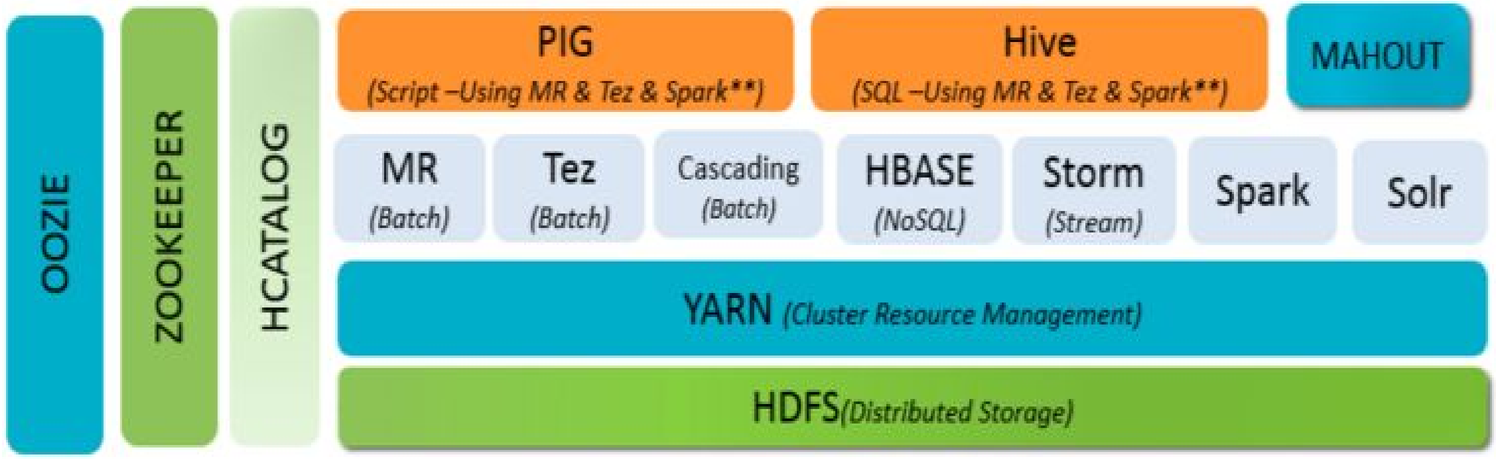
Gemini HIVE Workshop

02-17-2016

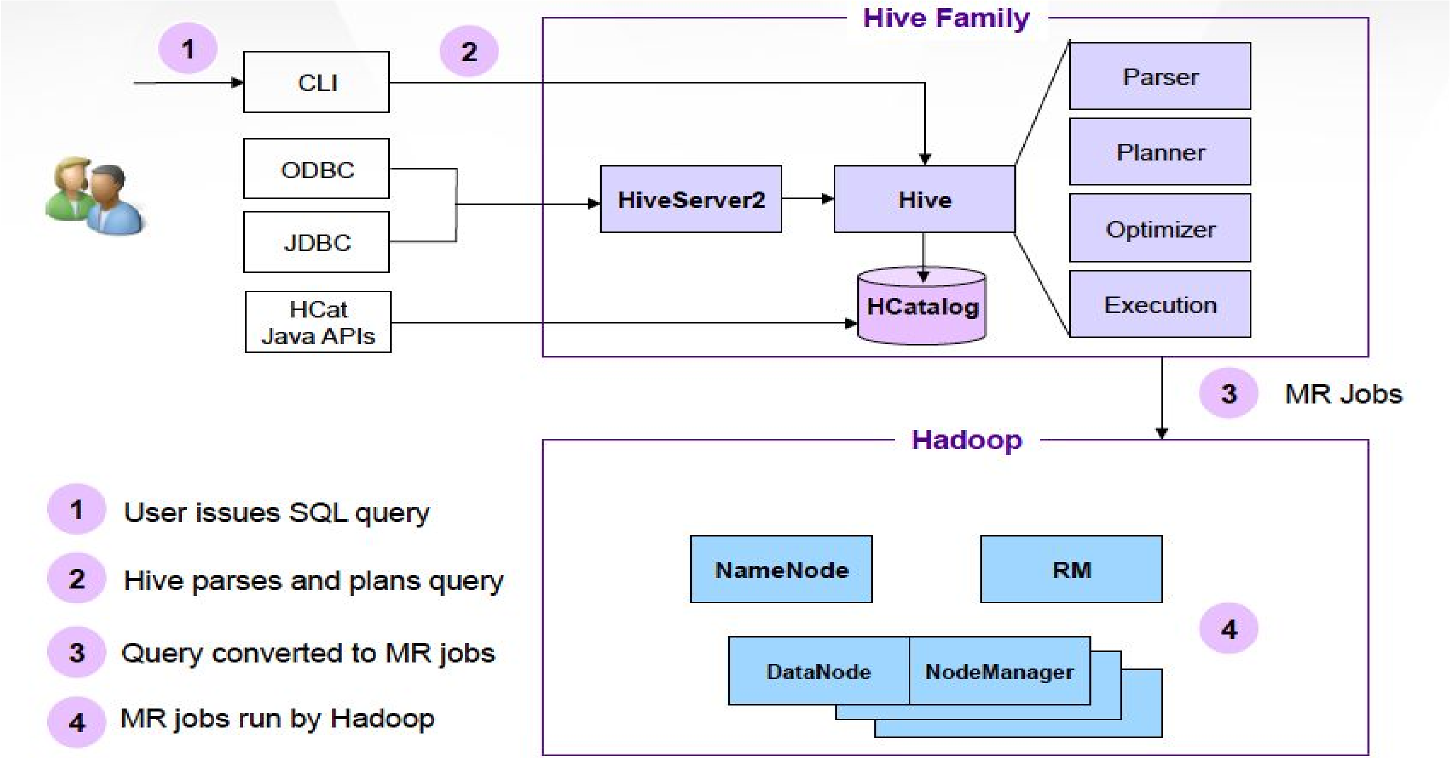
Hadoop Eco System



What is HIVE?

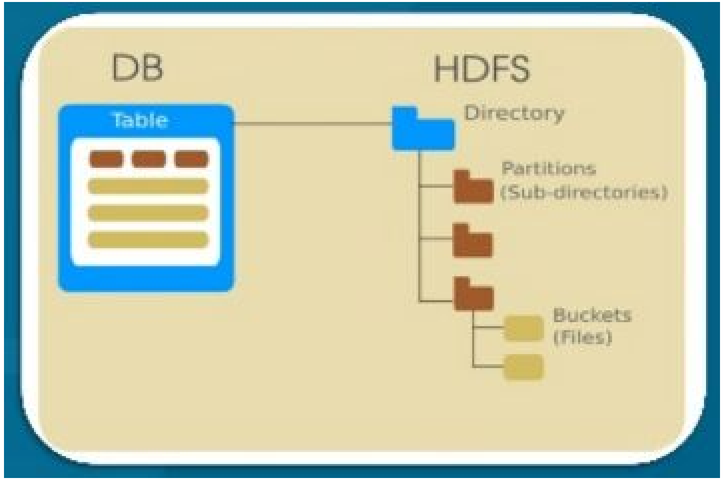
- A system for querying and managing structured data built on Hadoop
  - Structured data with rich types (structs, lists and maps)
  - HDFS for storage
  - Uses Map-Reduce for execution
- Provides SQL-like query language named HiveQL - minimal learning curve
- HIVE compile SQL queries into MapReduce jobs and run the jobs in the hadoop cluster
- Supports user-defined functions, scripts and customized I/O support

Hive Components

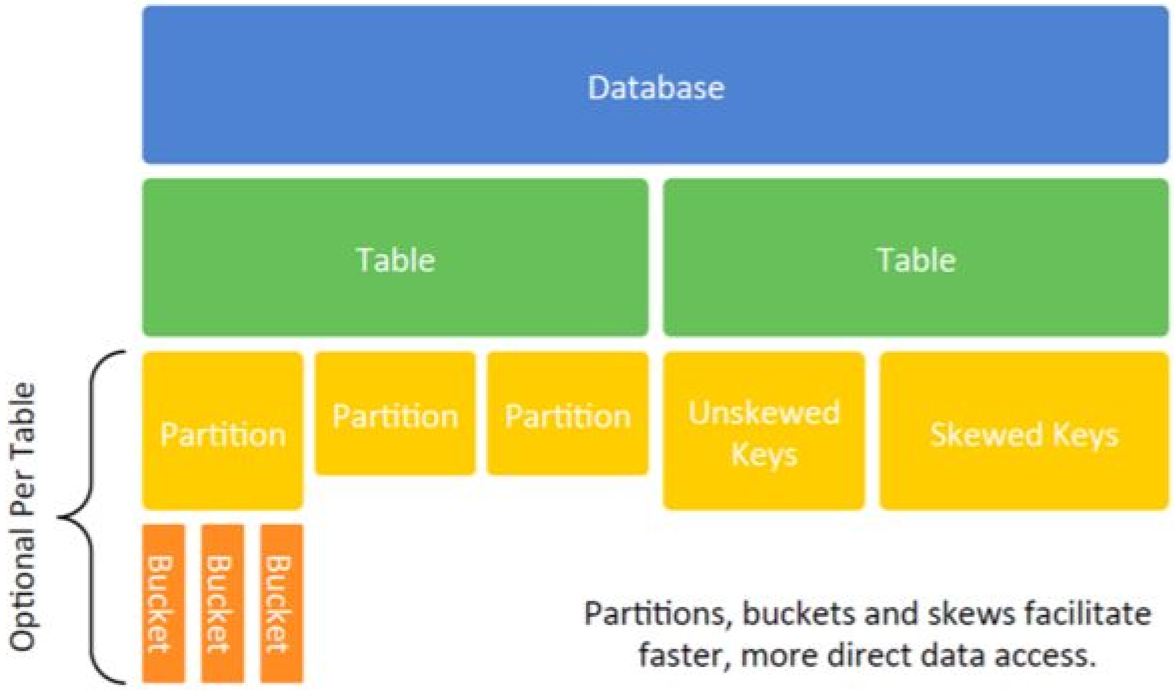


HIVE Data Model

- HIVE structure data into a well defined database concept
  - Tables, columns, and rows, partitions, buckets etc.
- Tables
  - Types columns (int, float, string, date, boolean)
  - Supports array/map/struct for JSON like data
- Partitions
  - Example: range partition tables by date
- Buckets
  - Buckets allow to split partitions
    - Allowing even more focused queries
  - Hash partition within ranges
  - Useful for sampling and join optimization



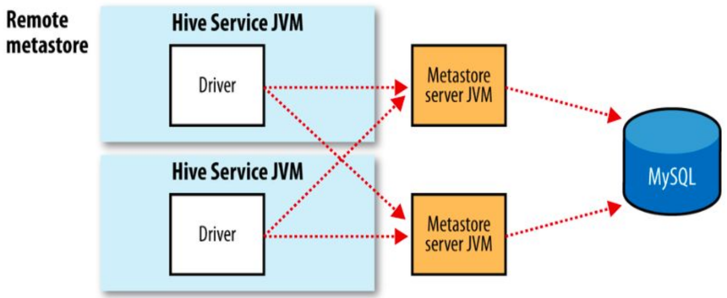
HIVE Data Model



HIVE Data Physical Layout

- Warehouse directory in HDFS
- Table row data is stored in warehouse subdirectories
- Partition creates subdirectories within table directories
- Actual data is stored in flat files
  - Control char-delimited text or Sequence Files
  - With custom Serializer/Deserializer (SerDe), files can use arbitrary format
- Queries with partition columns in WHERE clause will scan through only a subset of data

MetaStore

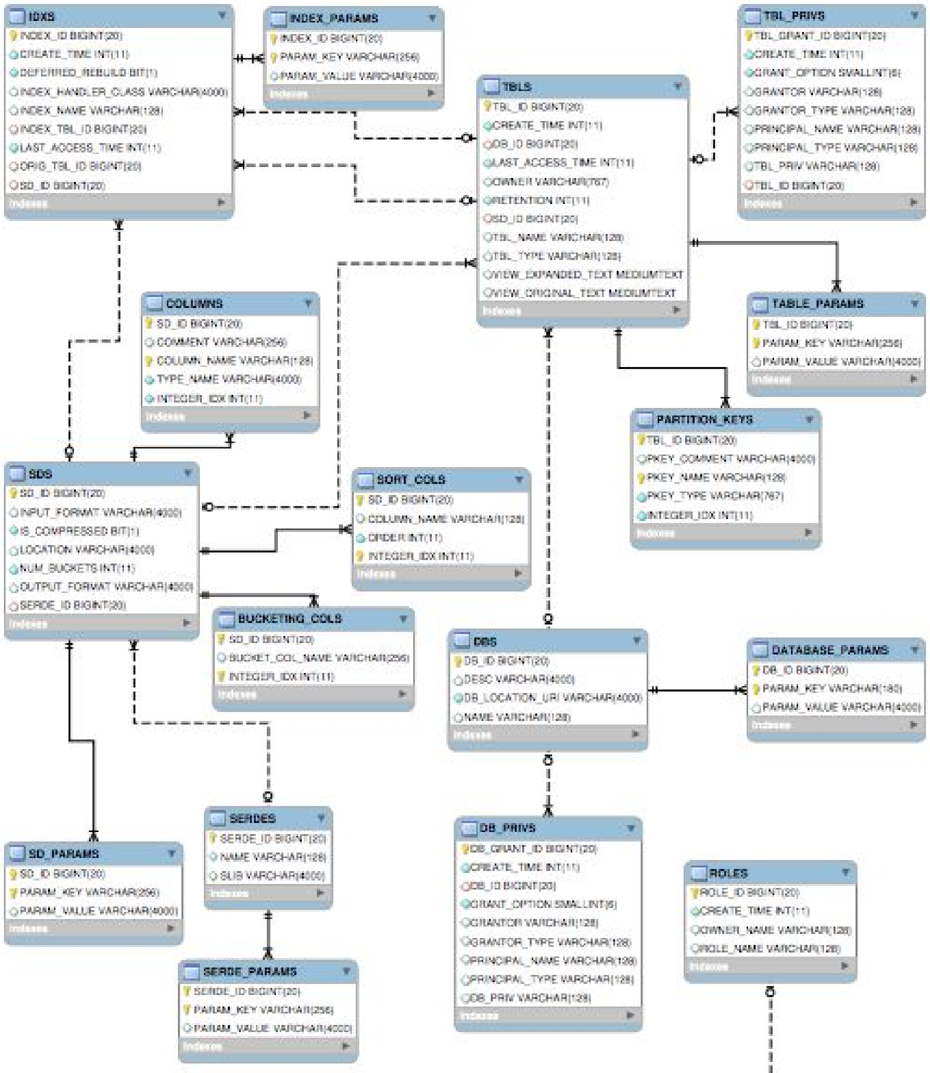


- Stores Table/Partition Properties
  - Table Schema and SerDe Library
  - Table location on HDFS
  - Logical Partitioning Keys and Types
  - Partition Level MetaData
  - Statistics about the database
- Thrift API
  - Remote: hive.metastore.uris=thrift://phazonblue-hcat.ygrid.vip.gq1.yahoo.com:50513
- Metadata stored in any SQL backend
- HiveQL Data Definition Language (DDL) statements such as CREATE TABLE updates megastore

MetaStore Schema

https://issues.apache.org/jira/secure/attachment/12471108/HiveMe

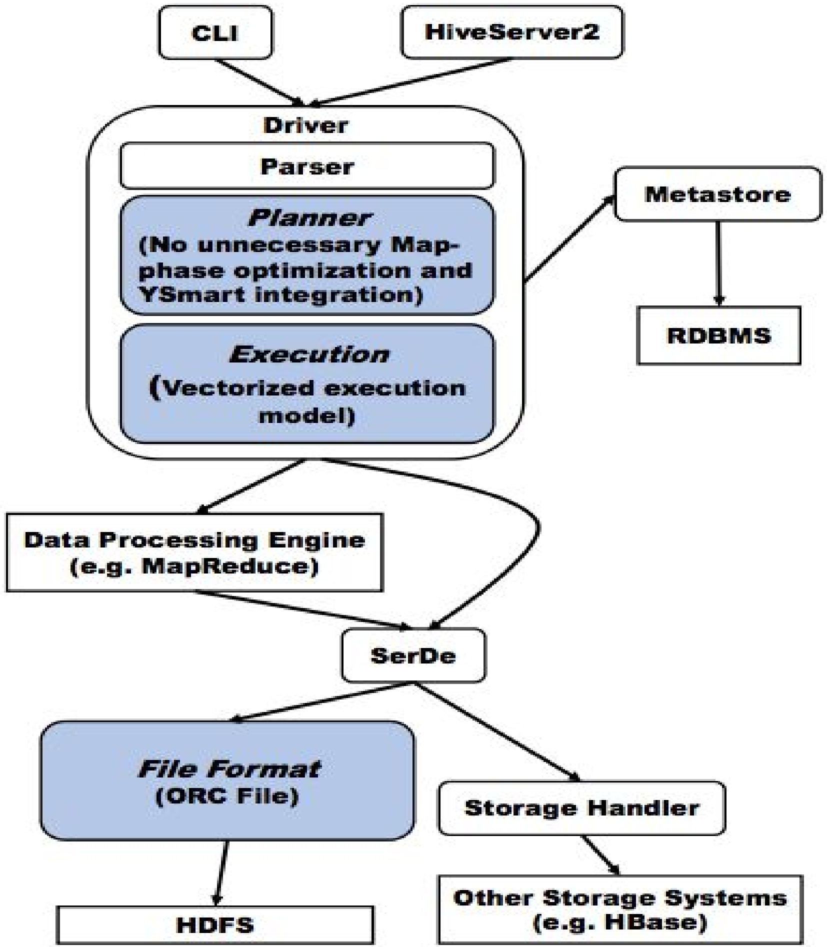
- Database
  - Database Params
  - Tables
    - Partition Keys
    - Table Params
    - Table Privileges
    - Storage Descriptor
      - Storage Params
      - Columns
      - Ser/De lib information
      - Bucketing columns
      - Sorting columns
  - Index (later)



HIVE Tables

- External Tables
  - Data is kept in the HDFS path specified by LOCATION keyword
  - Data is not managed by Hive
  - Drop table will not delete the data
    - CREATE EXTERNAL TABLE external_table (dummy STRING) **LOCATIO** **'/user/srinathm/external_table'**;
    - LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE external_table;
- Managed Tables/Internal Tables
  - Data is fully managed by HIVE
  - Data is kept in its warehouse directory (database directory)
    - CREATE TABLE managed_table (dummy STRING);
    - LOAD DATA INPATH '/user/tom/data.txt' INTO table managed_table;

Hive Architecture



Storage Formats

```
CREATE TABLE ...
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\001'
  COLLECTION ITEMS TERMINATED BY '\002'
  MAP KEYS TERMINATED BY '\003'
  LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

- Two dimensions that govern table storage in Hive
  - Row Format
    - Dictates how rows, and the fields in a particular row, are stored
    - Row format is defined by a SerDe (Serializer-Deserializer)
    - SerDe will deserialize a row of data from the bytes in the file to Hive objects
    - Hive uses a SerDe called LazySimpleSerDe for this delimited format
  - File Format
    - ROW FORMAT is not specified, controlled by the underlying binary file format
    - Divided into row-oriented formats and column-oriented formats
    - Row-oriented Formats - Avro, Sequence Files
    - Column-oriented Formats - Parquet, RCFile, and ORCFile
      - STORED AS ‘ORC’

File Formats, Record Formats -- Examples

- File Formats
  - Uses the InputFormat when reading data from the table
  - Uses the OutputFormat when writing data to the table
- Record Formats
  - SerDe encapsulates the logic for converting the unstructured bytes in a record into a record
- STORED AS TEXTFILE
  - inputFormat:org.apache.hadoop.mapred.TextInputFormat,
  - outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
  - ROWFORMAT SERDE ...
- STORED AS SEQUENCEFILE
  - inputFormat:org.apache.hadoop.mapred.SequenceFileInputFormat
  - outputFormat:org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
  - ROWFORMAT SERDE ...

Storage Descriptor

- Determines which input/output format and SerDe to use
- Each partition has its own storage descriptor
    - Can change file format and serialization formats at any time without affecting previous ones
- Hive creates an instance of InputFormat for partition
    - Uses it to read bytes from disk and splitting into records
    - Defines Input Data Split
- Hive creates an instance of SerDe and initializes with provided properties
    - Each record’s worth of bytes given to SerDe to create a record

ORCFile - Columnar Storage for HIVE

- Optimized Row Columnar Format (ORC)
  - Allows predicates to be pushed down to storage layer
  - Example :
    - SELECT COUNT(*) FROM CUSTOMER WHERE CUSTOMER.state = ‘CA’;
    - ORCFile Reader will only return rows that actually match where predicates
- ORC is an optimized, compressed, columnar storage format
  - Only needed columns are read
  - Blocks of data can be skipped using indexes and predicate pushdown
- Table Creation with ORC Format
  - CREATE TABLE ORCFileFormatExample (…) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS ORC tblproperties ("orc.compress"="GLIB");

ORCFile Format (1)

- Breaks rows into row groups (stripes)
  - Applies columnar compression and indexing within these row groups
  - Each column is contiguously stored for all rows

Stride Index

First 10,000 Rows

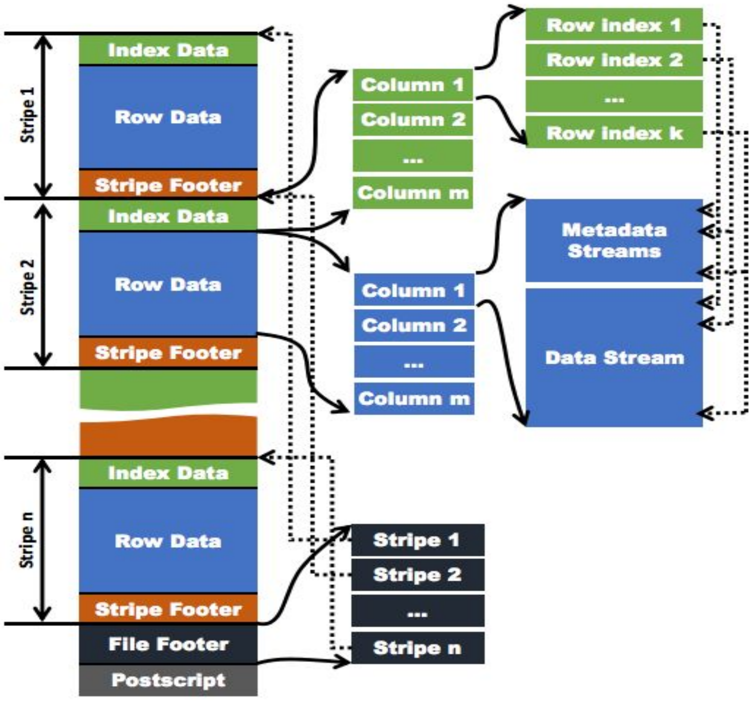
ID (min = 1, max=10000)	Name (dictionary, min, max)	State (dictionary, min, max)
1	Bob	NJ
2	Larry	CA
3	Sue	TX

Second 10,000 Rows

ID (min = 10001, max=20000)	Name (dictionary, min, max)	State (dictionary, min, max)
10001	Steve	OR
10002	Alan	ND
10003	Mary	FL

ORCFile Format (2)

- Divide rows into stripes
- In each stripe, data values are stored by column by column
    - Encoding specific to column type
- Stats for each column
    - Count, min, max and sum
- Data statistics at three levels
    - File level column stats in File footer
    - Stripe level column stats
    - Index groups - group of column values
- Postscript
    - Contains compression parameters



ORCFile - Stripe Structure

- Data - One section per stripe (each stripe - 250MB)
    - Composed of multiple streams for each column (depending on column type)
    - String columns have 4 streams
        - Present bit stream - Is the value non-null?
        - Dictionary Data - the bytes for strings
        - Dictionary length - the length of each entry
        - Row data - the row values
    - Integer columns have 2 streams
        - Present bit stream, Data Stream - stream of integers
- Index - (per stripe)
        - Position in each stream, Min and Max for each column (bloom filter in future)
- Footer
    - Directory of stream locations, Encoding of each column

ORCFile - Benzene.dailydata

- hive> DESCRIBE extended benzene.daily_data
    - Location: hdfs://phazonblue-nn1.blue.ygrid.yahoo.com:8020/data/FETL/benzene_daily_data
    - inputFormat:**org.apache.hadoop.hive.ql.io.orc.OrcInputFormat**
    - outputFormat:**org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat**
- orcfiledump utility - Example
    - hive --orcfiledump hdfs://phazonblue-nn1.blue.ygrid.yahoo.com:8020/data/FETL/benzene_daily_data/20160205/on/search/desktop/web/view/part-r-03584

ORCFile Dump Utility

```
[cb_marketplace@gwbl444n07 ~]$hive --orcfiledump hdfs://phazonblue-nn1.blue.ygrid.yahoo.com:8020/
                    data/FETL/benzene_daily_data/20160205/on/search/desktop/web/view/part-r-03584
Rows: 1270982
Stripe Statistics:
Stripe Statistics:
  Stripe 1:
    Column 0: count: 27909 hasNull: false
    Column 1: count: 27909 hasNull: false true: 27909
    Column 2: count: 27909 hasNull: false true: 29
    ...
 Stripe 2:
   Column 0: count: 45239 hasNull: false
   Column 1: count: 45239 hasNull: false true: 45239
   Column 2: count: 45239 hasNull: false true: 38
   ...

File Statistics:
  Column 0: count: 1270982 hasNull: false
  Column 1: count: 1270982 hasNull: false true: 1270982
  Column 2: count: 1270982 hasNull: false true: 179871
  ...

Stripes:
  Stripe: offset: 3 data: 91122564 rows: 27909 tail: 1588 index: 9023
    Stream: column 0 section ROW_INDEX start: 3 length 21
    Stream: column 1 section ROW_INDEX start: 24 length 44
    Stream: column 2 section ROW_INDEX start: 68 length 44
    ...
    Stream: column 1 section DATA start: 9026 length 14
    Stream: column 2 section DATA start: 9040 length 106
    Stream: column 3 section DATA start: 9146 length 10976
    Stream: column 3 section LENGTH start: 20122 length 7
    Stream: column 3 section DICTIONARY_DATA start: 20129 length 105
```

Partitioning in HIVE

- Hive tables can be value partitioned
  - Each partition is associated with a folder in HDFS
  - All partitions have an entry in the HIVE catalog (Metastore)
  - Hive optimizer will parse the query for filter conditions and skip unneeded partitions
  - Normally data partitioned by data load
  - Partitioned tables have a subfolder for each partition
- ORC (and other storage formats) support predicate pushdown
  - Query filters are pushed down into the storage handler
  - Blocks of data can be skipped without reading from HDFS based on ORC index
- Partitioning vs Predicate pushdown
  - Predicate pushdown is applied during file reads

Over Partitioning

- Partitioning feature is very useful in hive, but
  - Too many partitions may optimize some queries, but be detrimental for other queries
  - Creates lot of HDFS directories and small files depending on cardinality of partition column
  - Sometimes two level partitioning along different dimensions
    - date followed by geography (pay attention to skewness)
- Benzene.daily_data
  - Partitions based on 6 dimensions (dt, network, pty_family, pty_device, pty_experience, ...)

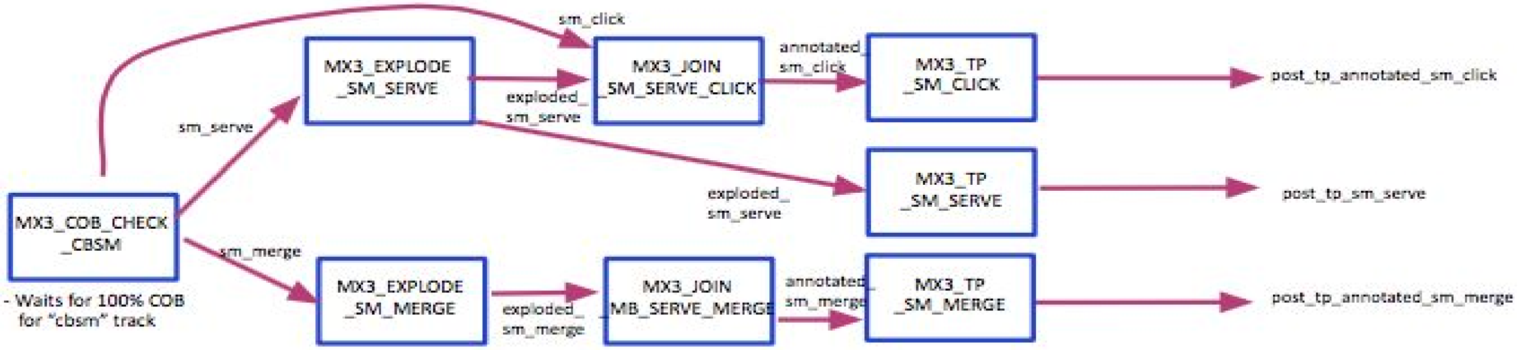
```
hive> describe extended benzene.daily_data;
...

# Partition Information
# col_name              data_type
dt                      string
network                 string
pty_family              string
pty_device              string
pty_experience          string
event_family            string

location:hdfs://phazonblue-nn1.blue.ygrid.yahoo.com:8020/data/FETL/benzene_daily_data
inputFormat:org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
outputFormat:org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
serializationLib:org.apache.hadoop.hive.ql.io.orc.OrcSerde
```

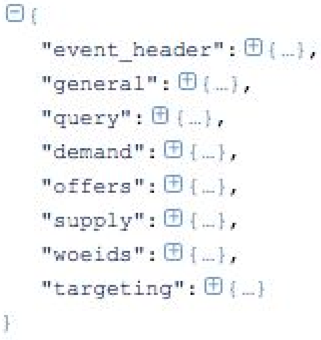
```
hive> show partitions benzene.daily_data;
...
dt=20160205/network=on/pty_family=search/pty_device=desktop/pty_experience=web/event_family=change
dt=20160205/network=on/pty_family=search/pty_device=desktop/pty_experience=web/event_family=interaction
dt=20160205/network=on/pty_family=search/pty_device=desktop/pty_experience=web/event_family=system
dt=20160205/network=on/pty_family=search/pty_device=desktop/pty_experience=web/event_family=unregistered
dt=20160205/network=on/pty_family=search/pty_device=desktop/pty_experience=web/event_family=view
dt=20160205/network=on/pty_family=search/pty_device=mobile/pty_experience=app/event_family=interaction
dt=20160205/network=on/pty_family=search/pty_device=mobile/pty_experience=app/event_family=unregistered
dt=20160205/network=on/pty_family=search/pty_device=mobile/pty_experience=app/event_family=view
dt=20160205/network=on/pty_family=search/pty_device=mobile/pty_experience=web/event_family=change
...
```

MX3 - Detour



- **Search/Merge/Clicks Events - Data Highway Event Collection (Avro Format)**
  - /projects/mx3/prod/dh/complete/cbsm/{SMServe, SMMerge, SMClick}
  - /user/mx3_prod/mx3_core_4_2_18/schemas (Schemas)
- **Serve/Merge events Join (Avro Format)**
  - https://git.corp.yahoo.com/MX3/mx3_sm_joins/blob/master/src/explode_sm_serve_avro.pig
  - https://git.corp.yahoo.com/MX3/mx3_sm_joins/blob/master/src/explode_sm_merge_avro.pig
  - https://git.corp.yahoo.com/MX3/mx3_sm_joins/blob/master/src/join_sm_merge_serve_avro.pig
- **Post TP Merge/Click events**
  - https://git.corp.yahoo.com/MX3/mx3_sm_tp/blob/master/src/post_tp_annotated_sm_merge_avro.pig
  - https://git.corp.yahoo.com/MX3/mx3_sm_tp/blob/master/src/post_tp_annotated_sm_click_avro.pig

Post TP Annotated SM Merge



- Top Level Schema contains multiple sections
    - event_header, general, query, demand, offers, supply, woeids, targeting
    - One serve event contains multiple offers (impressions/ads)
- Explode SMServeEvents and SMMergeEvents
    - One record for each offer
- Join SMMerge with SMServe (receive_time, event_guid and offer_guid)
    - MX3 maintains indexed version of SMServeEvents
    - Does a lookup into these files for each SMMerge Event instead of joining
- Apply TP UDF on SMMerge Events
    - Label each event with tp filter information

Adding Partition Info - Example

- add_partition.sql
  - USE ${HCAT_DB};
  - ALTER TABLE ${HCAT_TABLE} DROP IF EXISTS PARTITION(${PARTITION_NAME}='${PARTITION_VALUE}');
  - ALTER TABLE ${HCAT_TABLE} ADD PARTITION(${PARTITION_NAME}='${PARTITION_VALUE}') LOCATION '${PARTITION_LOCATION}';
- Parameters
  - HCAT_DB=mx3
  - HCAT_TABLE=post_tp_annotated_sm_merge
  - PARTITION_NAME=datestamp
  - PARTITION_VALUE=201602092345
  - PARTITION_LOCATION=hdfs://phazonblue-nn1.blue.ygrid.yahoo.com:8020/projects/mx3/prod/feeds/post_tp_annotated_sm_merge/15m/data/201602092345
- Oozie - Hive action
  - <hive xmlns="uri:oozie:hive-action:0.4">

Table-by-day schema pattern

- Most of fact tables are partitioned by datestamp in Gemini
- 15 min partition files (datestamp string)
    - post_tp_exploded_sm_serve
    - post_tp_annotated_sm_merge
    - post_tp_annotated_sm_click
    - post_tp_annotated_sm_impression
- Aggregation tables
    - supply_preagg_15m
    - supply_preagg_daily
    - dxs_preagg_15m
    - dxs_preagg_daily
- Use datestamp in WHERE clause to scan only needed partitions

```
hive> show partitions mx3.post_tp_exploded_sm_serve;
...
datestamp=201602061500
datestamp=201602061515
datestamp=201602061530
datestamp=201602061545
datestamp=201602061600
datestamp=201602061615
datestamp=201602061630
datestamp=201602061645
Time taken: 1.501 seconds, Fetched: 25742 row(s)
```

Partitioning Strategies

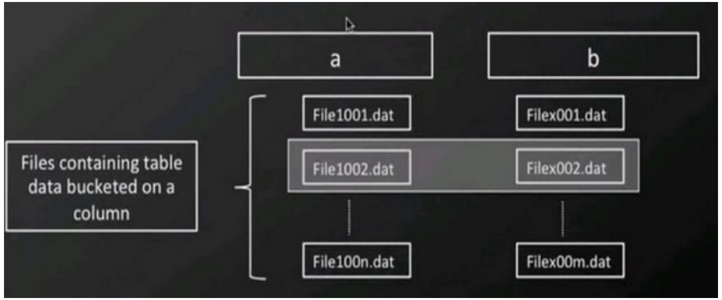
- Partitions by date and time: Use date as partition keys
- Partitions by locations: Use country, territory, state ...
- Partitions by business logic
  - Example: benzene.daily_data
  - dt, network, pty_family, pty_device, pty_experience,

Table Creation

- Creating database
  - `set GEMINI_SEARCH_INSIGHTS_HIVE_DB_PATH=/projects/cb_nctr/marketplace`
  - `CREATE DATABASE IF NOT EXISTS gemini_search_insights LOCATION '${hiveconf:GEMINI_SEARCH_INSIGHTS_HIVE_DB_PATH}'`
- Specify partition columns in create DDL statement
  - `hive> CREATE EXTERNAL TABLE IF NOT EXISTS gemini_search_insight (event_guid string, cb_bucket_id string, ...) PARTITIONED BY (date string);`
- If the table is external table ...
  - Generate data in new folder named as partition_key=value (example: date=20160124)
  - Add partition information to Metastore
    - `hive> ALTER TABLE auctions ADD IF NOT EXISTS PARTITION (date='20151104');`

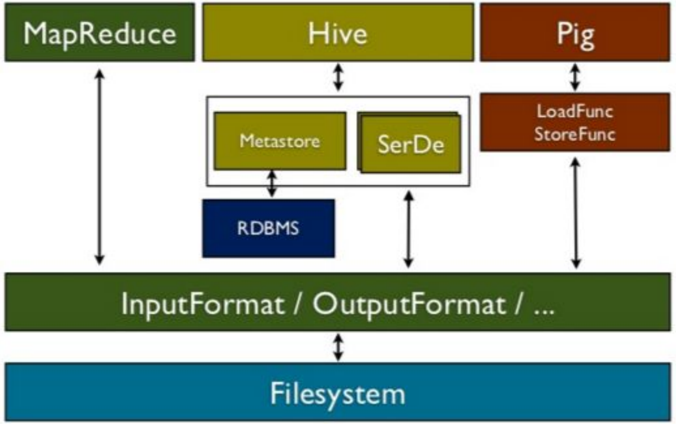
Bucketing

- Bucketing is another technique for decomposing data sets into more manageable parts
    - Records with the same id will always be stored in the same bucket
- Hive tables can be bucketed using the CLUSTERED keyword
    - One file/reducer per bucket
    - Buckets can be sorted
    - Additional advantages like bucket joins and sampling
    - if you have 5 partitions with 4 buckets - will have 20 reducers
- Joins
    - Helps in doing efficient map-side joins
    - Join of two tables which bucketed on same columns



What is HCatalog?

- Central metastore for facilitating interoperability among various Hadoop tools
- Acts as the table and storage management layer
  - Abstracts where or in what format data is stored
  - Pig, MapReduce, and Hive can share data
  - Presents a relational view of the data in HDFS
  - Enables notifications of data availability
- Pig/MapReduce need to remember
  - Where a data set is located
  - What format it has
  - What the schema is




```
PostTpServeImpressions = LOAD '$POST_TP_SERVE_SM_EVENTS_PATH/data/$DATE/part*'
   USING com.yahoo.yzip.hadoop.pig.YZipPigSchemaStorage(
   '$POST_TP_SERVE_SM_EVENTS_PATH/schema/$SCHEMA_DATE/post_tp_exploded_sm_serve.schema');
```

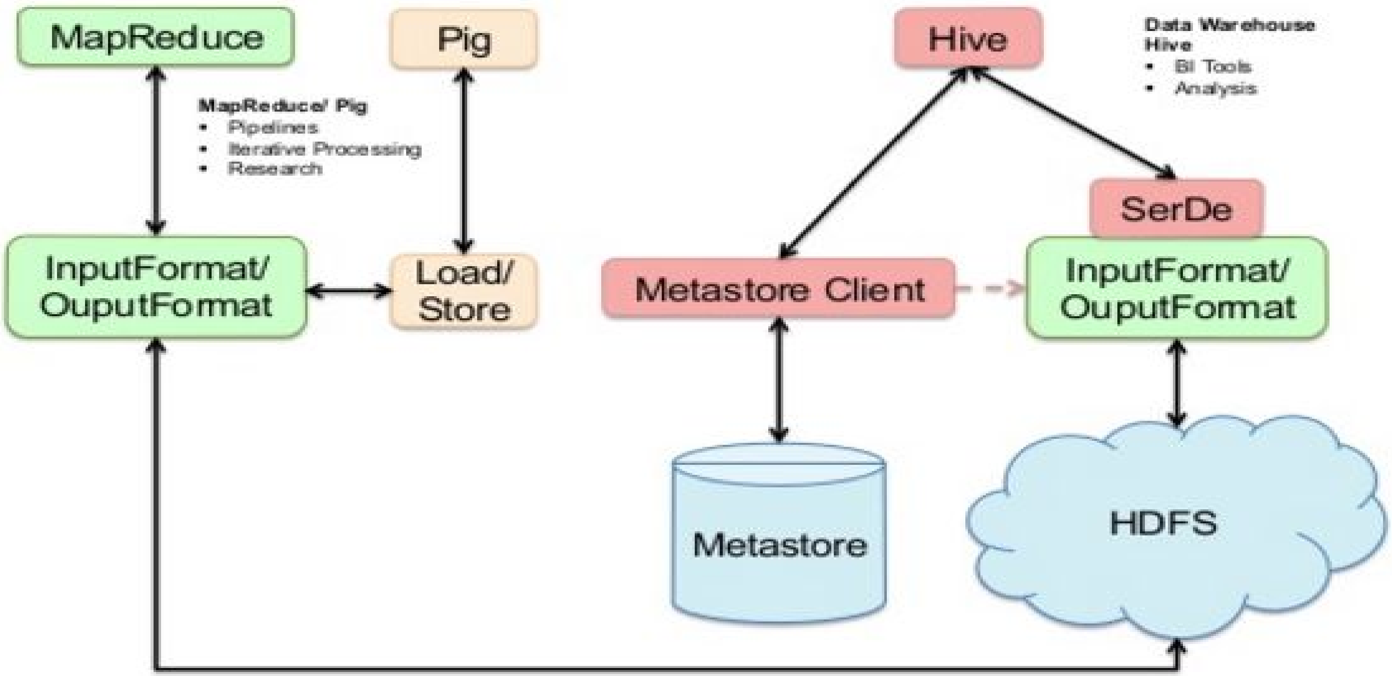
MR, PIG and HIVE Comparison

Feature	**MapReduce**	**Pig**	**Hive**
Record Format	Key Value Pairs	Tuple	Record
Data Model	User Defined	int,float,string,bytes, maps, tuples, bags	int,float,string,maps, structs,lists
Schema	Encoded in app	Declared in script or read by loader	Read from metadata
Data Location	Encoded in app	Declared in script	Read from metadata
Data Format	Encoded in app	Declared in script	Read from metadata

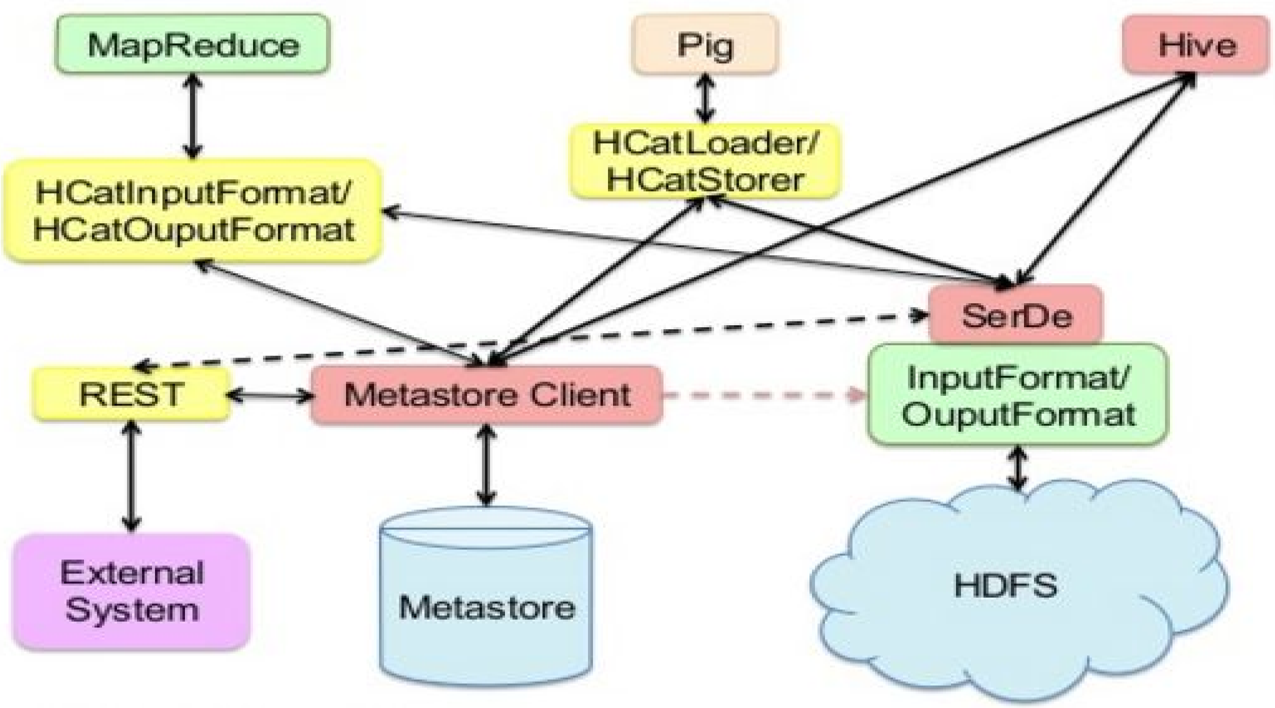
HCatalog Goals

- Provide an abstraction on top of datasets stored in HDFS
  - Just use the name of the dataset not the path
  - Data location, format and schema can be changed - Transparent to consumers
- Enable data discovery
  - Store all datasets (and properties) in HCatalog
- Provide notifications for data availability
  - Process new data immediately when it appears
  - Oozie or custom java code can wait for these events and schedules tasks based on them
- HCatalog enables non-HIVE projects to access HIVE tables

Hadoop - One platform, Many tools



Opening up MetaData to MR & PIG

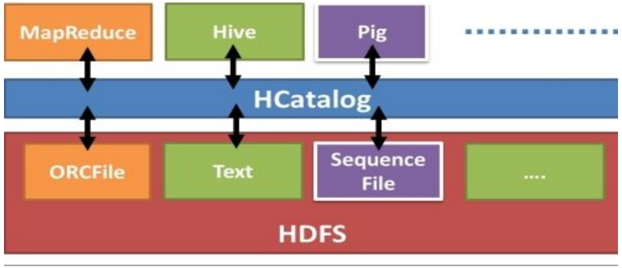


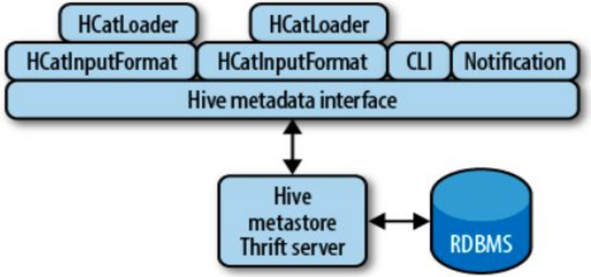
MR, PIG with HCatalog

Feature	**MapReduce+ HCatalog**	**Pig+HCatalog**	**Hive**
Record Format	Record	Tuple	Record
Data Model	int, float, string, maps, structs, lists	int, float, string, bytes, maps, tuples, bags	int, float, string, maps, structs, lists
Schema	Read from metadata	Read from metadata	Read from metadata
Data Location	Read from metadata	Read from metadata	Read from metadata
Data Format	Read from metadata	Read from metadata	Read from metadata

HCatalog Way

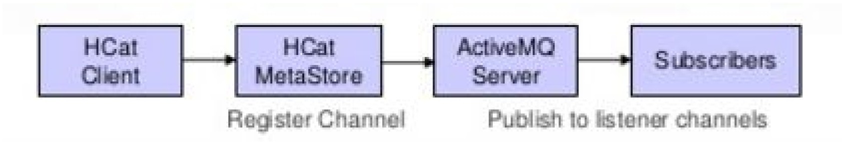
- Hive reads data location, format and schema from metadata
  - Managed by Hive metastore
- MR consists of HCatInputFormat and HCatOutputFormat
- HCatalog way in PIG
  - Consists of HCatLoader and HCatStorer
  - Indicates which partitions to scan by following the LOAD statement with FILTER



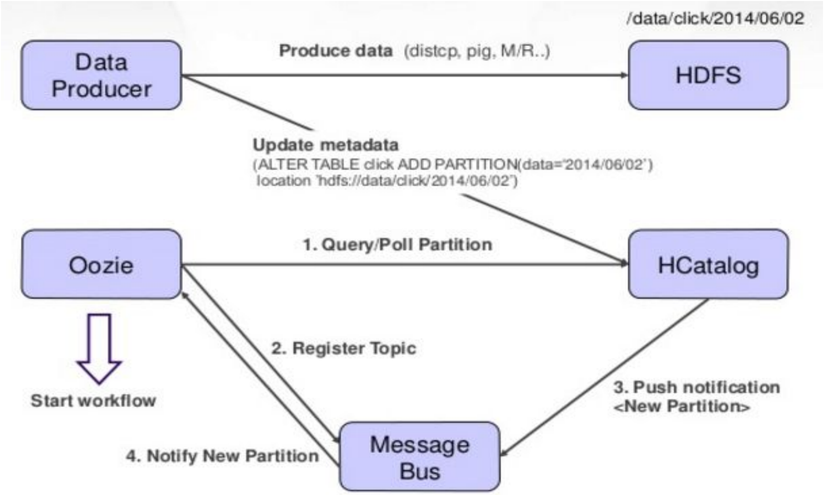



```
-- Load BenzeneData using HCatLoader
BenzeneData = LOAD 'benzene.daily_data' USING org.apache.hive.hcatalog.pig.HCatLoader();
```

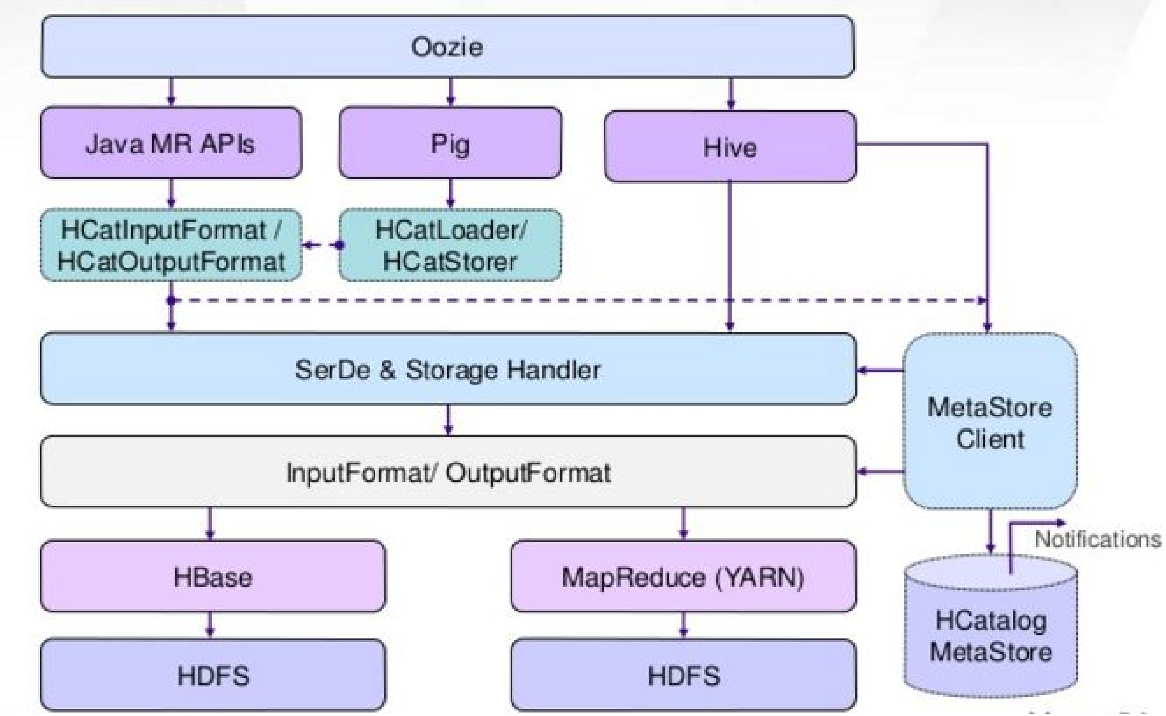
HCatalog Notification



- HCatalog uses JMS (Active MQ) notifications that can be sent for add_database, add_partition, drop_partition, drop_table etc.
- Oozie, Hcatalog and Messaging Integration



HCatalog Interoperability



HIVE Data Types

- Supported Column Data Types
  - Primitive Data Types
    - `tinyint, smallint, int, bigint, float, double, boolean, string`
  - Arrays - Collections of related items of same scalar data type
    - `alternate_names array[string]`
  - Structs - Object like collections
    - `address struct<street:string, city:string, state:string, zipcode:int>`
  - Maps - Key/Value pair collections
    - `preferences map<pref_code string, pref_value string>`

HIVE Query Language

- Basic SQL
  - From clause subquery
  - ANSI JOIN
  - Multi-Table Insert
  - Multi group-by
  - Sampling

SQL Semantics
SELECT, LOAD, INSERT from query
Expressions in WHERE and HAVING
GROUP BY, ORDER BY, SORT BY
CLUSTER BY, DISTRIBUTE BY
Sub-queries in FROM clause
GROUP BY, ORDER BY
ROLLUP and CUBE
UNION
LEFT, RIGHT and FULL INNER/OUTER JOIN
CROSS JOIN, LEFT SEMI JOIN
Windowing functions (OVER, RANK, etc.)

SQL Datatypes
INT
TINYINT/SMALLINT/BIGINT
BOOLEAN
FLOAT
DOUBLE
STRING
BINARY
TIMESTAMP
ARRAY, MAP, STRUCT, UNION
DECIMAL

Detour - What is gemini_search_insights.auctions ?

- Created auctions table to make ad-hoc analysis easier
- What does it contain?
  - Both Gemini and Bing impressions joined with clicks partitioned by day
  - Daily workflow to generate auctions - Adds a partition to hive table daily
  - Exploded various blobs as first class fields (ex: clkb blob, traffic shaping etc)
  - Joined with various dimension tables (ex: advertisers to extract mdm id)
  - Mapped Bing ads with their advertiser, campaign and adgroup information
  - Easy to compare KPI bing vs gemini at MDM, advertiser and keyword level
  - Many other convenient attributes got extracted
- Source
  - https://git.corp.yahoo.com/srinathm/mat/blob/master/monetization/pig/CBSearchAuctionsMacros.pig#L11
  - https://git.corp.yahoo.com/srinathm/mat/blob/master/monetization/pig/CBCoreEventsMacros.pig

Top K Queries By Volume

- Top 10 queries for each MDM by volume
- Pay attention to Rank function
- Over Clause
  - Defines window
  - PARTITION BY clause is used to reduce the scope of the window
  - rank() applied for each partition
  - PARTITION BY clause allows you to specify different partitions in the same select statement

```sql
SELECT *
FROM (
    SELECT
        advertiser_mdm_id as advertiser_mdm_id,
        cb_canon_user_query as cb_canon_user_query,
        rank() over (
            PARTITION BY advertiser_mdm_id
            ORDER BY COUNT(distinct event_guid) DESC
        ) as rank_1,
        COUNT(distinct event_guid) as bsearches,
        COUNT(*) as imps,
        SUM(click) as clicks,
        ROUND(SUM(clicks_advertiser_cost_usd), 2) as revenue,
        ROUND(SUM(clicks_advertiser_cost_usd)/SUM(click), 2) as ppc,
        ROUND(SUM(click)/COUNT(*), 2) as ctr
    FROM gemini_search_insights.auctions
    WHERE
        date >= '20160211' AND date <= '20160217'
        AND ad_listing_type = 'curveball'
    GROUP BY
        advertiser_mdm_id, cb_canon_user_query
    HAVING bsearches > 1000
) a
WHERE rank_1 < 10
ORDER BY advertiser_mdm_id, rank_1;
;
```

Top MDMs By Revenue

- Easy to compute aggregation functions like count(), sum() over group by clause
- How about percentages with respect to total revenue?

```sql
SELECT
    advertiser_mdm_id as advertiser_mdm_id,
    COUNT(distinct event_guid) as bsearches,
    COUNT(*) as imps,
    SUM(click) as clicks,
    ROUND(SUM(clicks_advertiser_cost_usd), 2) as revenue,
    ROUND(SUM(clicks_advertiser_cost_usd)/SUM(click), 2) as ppc
FROM gemini_search_insights.auctions
WHERE
    date >= '20160201' AND date <= '20160207'
    AND ad_listing_type = 'curveball'
GROUP BY
    advertiser_mdm_id
HAVING bsearches > 10000
ORDER BY revenue DESC
;
```

1	advertiser_md	bsearches	imps	clicks	revenue	ppc
2	4672	131,839,780	183,678,773	2,736,682	$454,461	$0.17
3	5223	42,553,452	43,350,541	1,776,981	$318,515	$0.18
4	97247	1,531,233	1,598,970	254,572	$169,526	$0.67
5	892	1,237,560	1,536,939	120,930	$124,184	$1.03
6	2166	6,272,742	7,146,341	145,313	$107,860	$0.74
7	15338	1,288,312	1,301,267	79,495	$95,186	$1.20
8	11358	1,987,098	2,004,210	137,950	$87,019	$0.63
9	1498	6,602,148	6,982,984	253,724	$85,169	$0.34
10	538344	6,390,297	7,979,936	334,607	$84,433	$0.25
11	2255	132,323	133,070	2,791	$82,999	$29.74

Top MDMs By Revenue Percentage

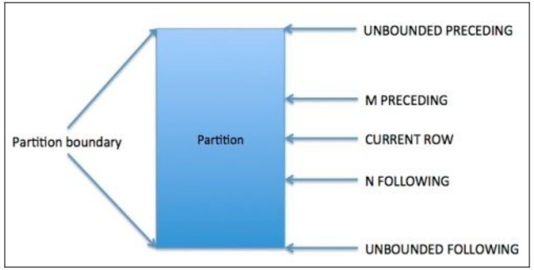
- Pay attention to multiple over clauses in inner select
- Over() computes totals
- Over(partition by mdm) computes totals at mdm level
- Multiple partition clauses in a single select statement

```sql
SELECT
    advertiser_mdm_id as advertiser_mdm_id,
    mdm_clicks as mdm_clicks,
    mdm_revenue as mdm_revenue,
    total_clicks as total_clicks,
    total_revenue as total_revenue,
    ROUND(mdm_clicks/total_clicks, 2) as clicks_percentage,
    ROUND(mdm_revenue/total_revenue, 2) as revenue_percentage
FROM (
    SELECT
        advertiser_mdm_id as advertiser_mdm_id,
        SUM(click)
            OVER(PARTITION BY advertiser_mdm_id) as mdm_clicks,
        SUM(clicks_advertiser_cost_usd)
            OVER(PARTITION BY advertiser_mdm_id) as mdm_revenue,

        SUM(click) OVER() as total_clicks,
        SUM(clicks_advertiser_cost_usd) OVER() as total_revenue

    FROM gemini_search_insights.auctions
    WHERE
        date >= '20160201' AND date <= '20160207'
        AND ad_listing_type = 'curveball'
) r
ORDER BY revenue_percentage DESC
LIMIT 25;
```

How about cumulative totals?



- Getting top 10% queries by volume from each MDM
- Pay attention to computation of cumulative bidded searches

```
SELECT * FROM (
SELECT
    advertiser_mdm_id as advertiser_mdm_id,
    cb_canon_user_query as cb_canon_user_query,
    query_rank as query_rank,
    bsearches as bsearches,
    cum_bsearches as cum_bsearches,
    total_bsearches as total_bsearches,
    ROUND(cum_bsearches/total_bsearches, 5) as cum_bsearches_ratio
FROM (
    SELECT
        advertiser_mdm_id as advertiser_mdm_id,
        cb_canon_user_query as cb_canon_user_query,
        COUNT(distinct event_guid) as bsearches,

        SUM(COUNT(distinct event_guid)) OVER (
            PARTITION BY advertiser_mdm_id ORDER BY COUNT(distinct event_guid) DESC
            ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
        ) as cum_bsearches,

        SUM(COUNT(distinct event_guid)) OVER (PARTITION BY advertiser_mdm_id) as total_bsearches,

        rank() over (
            PARTITION BY advertiser_mdm_id ORDER BY COUNT(distinct event_guid) DESC
        ) as query_rank

        FROM gemini_search_insights.auctions
        WHERE
            date >= '20160211' AND date <= '20160217'
            AND ad_listing_type = 'curveball'
        GROUP BY
            advertiser_mdm_id, cb_canon_user_query
) query_ranked
) o
WHERE cum_bsearches_ratio <= 0.1
ORDER BY advertiser_mdm_id, query_rank
;
```

Windowing Functions Summary

- Concept
  - Partition Rows
  - Within each partition order rows
  - For each row, define window
  - Apply function on all rows in window
- Observations
  - It produces one output row for every input row
  - Can consume multiple rows to produce one row
- Applications
  - Ranking, Sliding Window aggregates, Lead/Lag analysis

Windowing - ORDER BY, ROWS

- If ORDER BY is not specified in OVER clause, entire partition is used for window frame
- If ROWS/RANGE is not specified but ORDER BY is specified, RANGE UNBOUNDED PRECEDING AND CURRENT ROW is used as default for window frame
  - **SELECT avg(foo) OVER (PARTITION BY bar ORDER BY foo) is not equivalent to SELECT avg(foo) OVER (PARTITION BY bar)**
  - **Unordered window frame range defaults to the whole partition, but the ordered window frame ranges from the first partition row (based on the order clause) to the current partition row.**
- More than one window function can be used in a single query with a single FROM clause.
- The OVER clause for each function can differ in partitioning and ordering.

Latest URL for each Session

- Using Inner JOIN

```
CREATE TABLE clicks (
    timestamp date, sessionID string,
    url string, source_ip string
) STORED as ORC tblproperties ("orc.compress" = "SNAPPY");
```

```
SELECT clicks.*
FROM clicks
JOIN
(
    SELECT sessionID, max(timestamp) as max_ts
    FROM clicks
    GROUP BY sessionID
) latest
ON clicks.sessionID = latest.sessionID and
clicks.timestamp = latest.max_ts;
```

- Using Rank Function

```
SELECT * FROM
(
    SELECT *,
        RANK() OVER (PARTITION BY sessionID ORDER BY timestamp DESC) as rank
    FROM clicks
) ranked_clicks
WHERE ranked_clicks.rank=1;
```

Another Simple Example

- Get balance after every transaction

```sql
SELECT actid, tranid, val,
  SUM(val) OVER(PARTITION BY actid
                ORDER BY tranid
                ROWS BETWEEN UNBOUNDED PRECEDING
                         AND CURRENT ROW) AS balance
FROM dbo.Accounts;
```

```sql
-- Set-Based Solution Using Subqueries
SELECT actid, tranid, val,
  (SELECT SUM(S2.val)
   FROM dbo.Accounts AS S2
   WHERE S2.actid = S1.actid
     AND S2.tranid <= S1.tranid) AS balance
FROM dbo.Accounts AS S1;

-- Set-Based Solution Using Joins
SELECT S1.actid, S1.tranid, S1.val,
  SUM(S2.val) AS balance
FROM dbo.Accounts AS S1
  JOIN dbo.Accounts AS S2
    ON S2.actid = S1.actid
   AND S2.tranid <= S1.tranid
GROUP BY S1.actid, S1.tranid, S1.val;
```

FIRST_VAL, LAST_VAL windowing functions

- FIRST_VAL and LAST_VAL
    - Example - Landing/Exit page analysis for a given session
    - Pay attention to LAST_VAL windowing rows

```
SELECT landing_page, exit_page, count(*)
FROM (
    SELECT session_id,
        FIRST_VAL(page_type) OVER(PARTITION BY session_id ORDER BY timestamp) as landing_page,
        LAST_VAL(page_type) OVER(PARTITION BY session_id ORDER BY timestamp
                 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) as exit_page,
        RANK() OVER(PARTITION BY session_id ORDER BY timestamp
    FROM user_sessions
) sd
WHERE rs = 1
GROUP BY landing_page, exit_page

```

    - More fun example: Compare your salary with highest salary (FIRST_VAL) at your level

Sessionization using windowing function

```sql
SELECT *,
  user_id || '_' || SUM(new_session)
  OVER (PARTITION BY user_id ORDER BY timestamp) AS session_id
FROM (
    SELECT *,
        CASE
            WHEN timestamp - LAG(timestamp)
               OVER (PARTITION BY user_id ORDER BY mytimestamp) >= 30 * 60
            THEN 1 ELSE 0
         END as new_session
    FROM user_session_data
) sd
```

- Creates new_session as 1 whenever time_diff with previous row >= 30*60
- Finally creates session_id (<user_id>_1, <user_id>_2 ..)
    - Pay attention to SUM(new_session) OVER (PARTITION BY user_id ORDER by timestamp)
    - Windowing frame (unbounded rows preceding and current row) - Cumulative Sum
- Pay attention to usage of CASE statement in SELECT clause

UDF vs UDAF vs UDTF

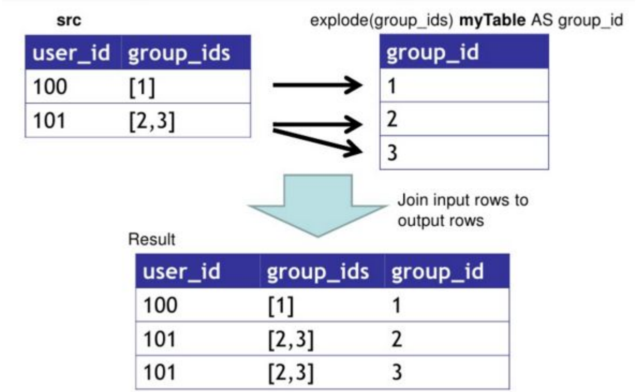
- User Defined Functions
  - One-to-one mapping, Single row to Single row
  - concat(“a”, “b”)
- User Defined Aggregate Functions
  - Many-to-one mapping, Multiple Rows to Single Row
  - sum(num_ads)
- User defined Table Generating Functions
  - One to many mapping, Single row to Multiple Rows
  - explode ([1, 2, 3])

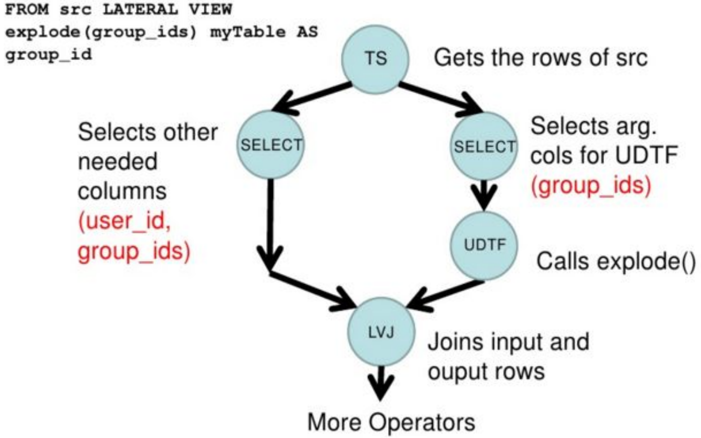
UDTF

- explode(Array<?> arg)
  - Converts an array into multiple rows with one element per row (Like FLATTEN in PIG)
  - select EXPLODE(group_ids) as group_id from src
    - group_ids is an array of elements [1, 2, 3] => results 3 rows
- Transform syntax limited to a single expression
- Use Lateral View for multiple expressions

Lateral View

- Example Query
  - SELECT src.*, myTable.* FROM src, LATERAL VIEW explode(group_ids) lv AS group_id
  - Multiple LATERAL VIEWS can be used to explode multiple fields in src
  - NULL value in group_ids can result zero rows (like in PIG)





Benzene Audience Data Feed (benzene.daily_data)

- Benzene Data Feed
  - Data describing the information presented to the users (view events)
  - Users’ interactions performed within the product
- Interesting Fields
  - page_info - map
    - Meta data about what was shown to the user - Search Query, Num Ads Shown etc
  - view_info - array<map>
    - Meta data about individual objects on the page that the user can interact with - links
  - click_info - map
    - click_info contains the same meta data associated with the object the user interacted

MULTI TABLE Insert Statment

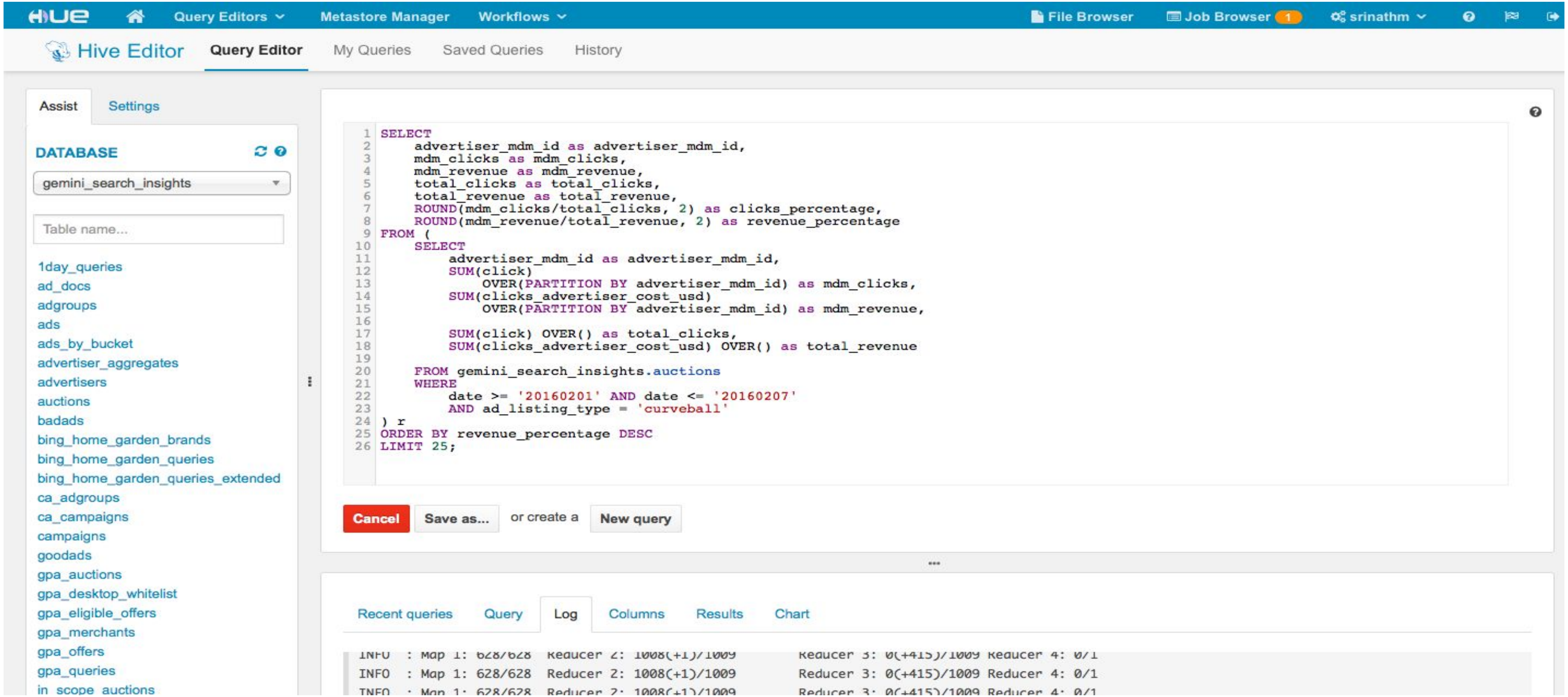
- Multi-Table insert from SELECT
  - FROM (SELECT ...) AS benzene_search_event
  - INSERT OVERWRITE TABLE gemini_pla_db.benzene_searches PARTITION(event_date)
  - SELECT ... DISTRIBUTE BY event_date
  - INSERT OVERWRITE TABLE gemini_pla_db.benzene_pla_views PARTITION(event_date)
  - SELECT ... DISTRIBUTE BY event_date
  - https://git.corp.yahoo.com/srinathm/mat/blob/master/monetization/hive/benzene_pla.hql#L142

- Pay attention to Dynamic Partition feature
  - Can infer partitions to be created from query
  - The static partition keys must come before the dynamic partition keys (event_date as last)
  - `set hive.exec.dynamic.partition=true;`

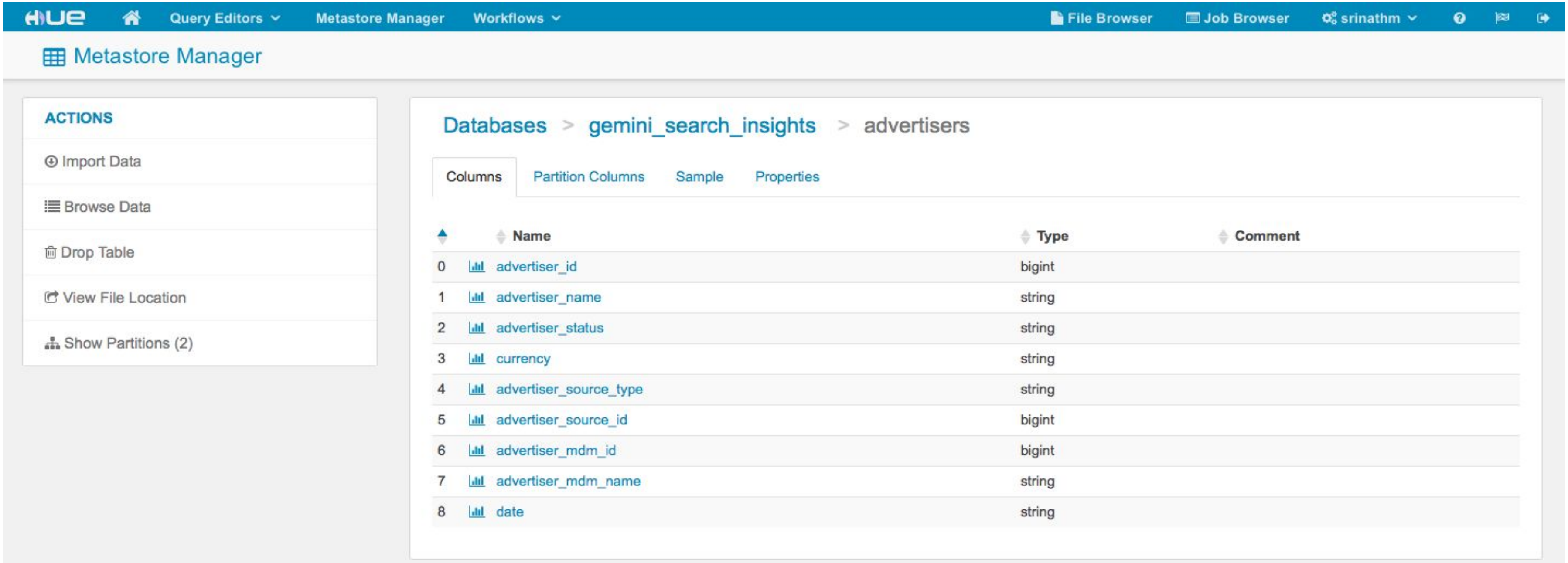
Loading Data into Tables

- Loading data into managed tables
  - First creates directory from partition information and then copies data into it
  - LOAD DATA LOCAL INPATH ‘...’
  - OVERWRITE INTO TABLE ... PARTITION (...)
- Creating tables and loading data in one query
  - CREATE TABLE ... AS SELECT ...

Hue Hive Editor (http://yo/hue.pb)



hue - Metastore Manager (http://yo/hue.pb)



ORDER, SORT, DISTRIBUTE, CLUSTER

- ORDER BY col_1
  - Guarantees global ordering
  - Only one reducer to sort the final output
- SORT BY col_1
  - Orders data at each of N reducers, but each reducer can receive overlapping ranges of data
  - End up with N sorted files with overlapping ranges
- DISTRIBUTE BY col_1
  - Ensures each of N reducers gets non-overlapping ranges of x (No sorting)
  - End up with N unsorted files with non-overlapping ranges.
- CLUSTER BY col_1
  - Ensures each of N reducers gets non-overlapping ranges, then sort at each of these ranges

Bucketed Tables

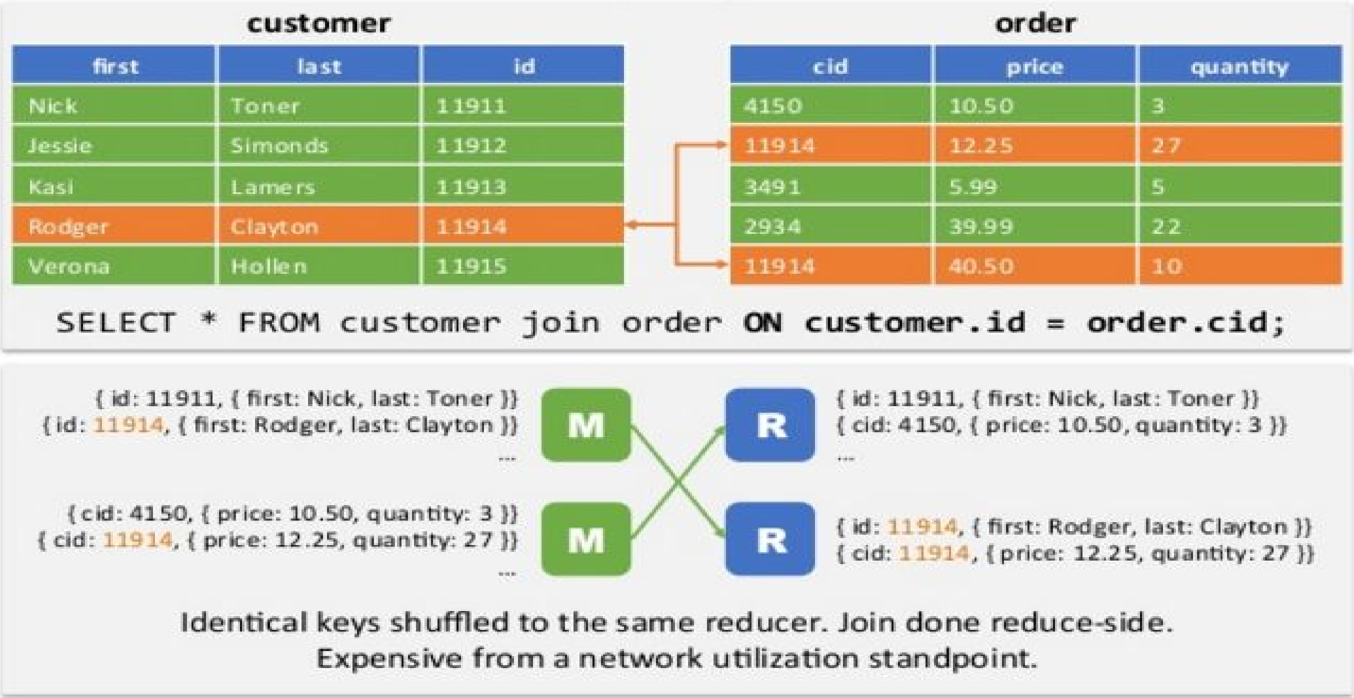
- Creating Bucketed Tables
  - CREATE TABLE keywords_by_bucket(...) PARTITIONED BY (date string)
  - CLUSTERED BY (adgroup_id) INTO 2048 BUCKETS;
- Bucketing
  - Determined by hash_function(bucketing_col) mod num_buckets
  - hive.enforce.bucketing = true (must be set at the time of inserting records)
- Inserting Records
  - LOAD DATA INPATH

    '/projects/cb_nctr/marketplace/demand_snapshot/keywords/date=20160209/' OVERWRITE INTO TABLE gemini_search_insights.keywords_by_bucket PARTITION(date='20160209')
  - set mapreduce.reduce.tasks = 2048 (can also be used to force 2048 buckets)

JOIN Strategies

- Inner Side Join
  - Left Outer Join, Right Outer Join, Full Outer Join, Left Semi Join
- Map Side Join
- Bucket Map Join
- Sort Merge Bucket Join
- Sort Merge Bucket Map Join
- Skew Join

Shuffle Joins in Map/Reduce - Refresh

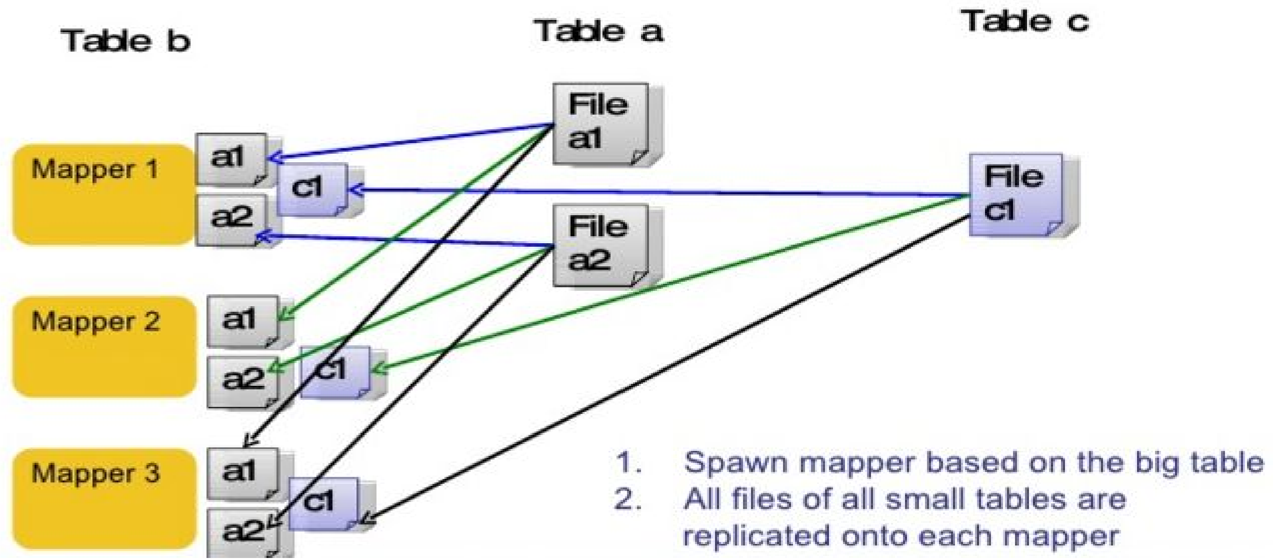


Map Join - (Replicated Join in PIG)

- Start schemas use dimension tables small enough to fit in RAM
- Small tables held in memory by all nodes
- Single pass through large table
- Largest table can be streamed through the mappers while the small tables are cached in memory
  - set hive.auto.convert.join=true (must be set to activate this type of join)
  - hive.mapjoin.smalltable.filesize=25000000 (configure small file size)

Map Join Example

SELECT a.*, b.*, c.* a JOIN b on a.key = b.key JOIN C on a.key=c.key;



Bucket Join

- Cluster and sort by join key in both tables (equal number of buckets)
  - CREATE TABLE ORDER (cid int, price float, quantity int) CLUSTERED BY (cid) SORTED BY (cid) INTO 32 BUCKETS
  - CREATE TABLE CUSTOMER (id int, first string, last string) CLUSTERED BY (id) SORTED BY (id) INTO 32 BUCKETS
- Join happens on map side itself
- Another Example: Generate all active ads
  - All possible combinations of <keyword, ad> pairs within each adgroup
  - JOIN ADS and KEYWORDS by adgroup_id (cross product of ads and keywords)

Bucket Join - Use Case

- Keywords - keywords_by_bucket (adgroup)
  - Create keywords clustered by adgroup_id into 1024 buckets
  - *CREATE TABLE IF NOT EXISTS keywords_by_bucket(...) PARTITIONED B CLUSTERED BY (adgroup_id) SORTED BY (adgroup_id) INTO 1024 BUCK FORMAT DELIMITED FIELDS TERMINATED BY '^A' LINES TERMINATEI STORED AS TEXTFILE*;
- Populate data
  - *FROM keywords INSERT INTO TABLE keywords_by_bucket PARTITION(d SELECT advertiser_id, ... DISTRIBUTE BY adgroup_id SORT BY adgroup_id*
- Follow similar steps by for ads - ads_by_bucket (adgroup)
  - We have both keywords and ads partitioned by date and bucketed by adgroup_id (and sorted) into 1024 buckets

Hive Configuration - Bucketed Table Creation

- Bucket Table Creation
    - set hive.enforce.bucketing = true
    - set hive.enforce.sorting=true;
    - set mapreduce.job.reduces=1024; (sames as number of buckets)
    - set hive.execution.engine=mr;
    - set mapreduce.job.acl-view-job=*;
- Running Bucket Map Sort Merge JOIN
    - set hive.auto.convert.sortmerge.join=true;
    - set hive.optimize.bucketmapjoin = true;
    - set hive.optimize.bucketmapjoin.sortedmerge = true;
- Ad Docs Query (Keywords x Ads by AdGroup)
    - *CREATE TABLE ad_docs AS SELECT k.advertiser_id, k.advertiser_name, k.advertiser_status ..., a.ad_id, a.ad_status, a.ad_title, a.ad_description ... FROM keywords_by_bucket k JOIN ads_by_bucket a ON (k.adgroup_id = a.adgroup_id)*

SubQuery - JOIN

- Get Top Ad Groups having highest number of servable ad docs

```
SELECT
    k.advertiser_id, k.advertiser_name,
    k.advertiser_mdm_id, k.advertiser_mdm_name, k.adgroup_id,
    keywords_count, ads_count, keywords_count*ads_count as addocs_count
FROM (
    SELECT advertiser_id, advertiser_name,
       advertiser_mdm_id, advertiser_mdm_name, adgroup_id,
       COUNT(*) as keywords_count
    FROM keywords_by_bucket
    GROUP BY
        advertiser_id, advertiser_name,
        advertiser_mdm_id, advertiser_mdm_name, adgroup_id
    ORDER BY keywords_count
) k JOIN
(
   SELECT advertiser_id, advertiser_name,
       advertiser_mdm_id, advertiser_mdm_name, adgroup_id,
       COUNT(*) as ads_count
    FROM ads_by_bucket
    GROUP BY
        advertiser_id, advertiser_name,
        advertiser_mdm_id, advertiser_mdm_name, adgroup_id
    ORDER BY ads_count
) a
ON (k.adgroup_id = a.adgroup_id)
ORDER BY addocs_count DESC
LIMIT 5000
```

k.advertiser_i	k.advertiser_r	k.advertiser_r	k.advertiser_r	k.adgroup_id	keywords_cou	ads_count	addocs_count
1169024	tw.indeed.com	355726	tw.indeed.com	8472739232	63,998	153,795	9,842,572,410
1096970	Kenshoo	510581	Kenshoo	8138978839	8,388	2,596	21,775,248
1096970	Kenshoo	510581	Kenshoo	8138980835	5,983	2,540	15,196,820
57095	RW Lynch_NEW	315256	RW Lynch_NEW	8091521358	11,106	1,130	12,549,780
1096970	Kenshoo	510581	Kenshoo	8138113572	3,464	3,403	11,787,992
911682	Gem_Tablet_En	2393	Gem_Tablet_En	7963008710	33,148	100	3,314,800
911682	Gem_Tablet_En	2393	Gem_Tablet_En	7963007817	44,596	73	3,255,508
40459	Drapers and Dar	368	Drapers and Dar	8154283737	2,914	763	2,223,382
911682	Gem_Tablet_En	2393	Gem_Tablet_En	7963008644	16,884	108	1,823,472
40459	Drapers and Dar	368	Drapers and Dar	8154283789	2,551	521	1,329,071

Join Strategies

Type	Approach	Pros	Cons
Shuffle Join	Join keys are shuffled using map/ reduce and joins performed reduce side.	Works regardless of data size or layout.	Most resource-intensive and slowest join type.
Broadcast Join	Small tables are loaded into memory in all nodes, mapper scans through the large table and joins.	Very fast, single scan through largest table.	All but one table must be small enough to fit in RAM.
Sort-Merge-Bucket Join	Mappers take advantage of co-location of keys to do efficient joins.	Very fast for tables of any size.	Data must be bucketed ahead of time.

Column Sorting to facilitate skipping

- Use ORC file format to facilitate skipping by sorting data
- ORC file format maintains index for a group of records
- Index contains stats for every column (min, max ...) in that group
- If data is sorted, it uses index stats to skip block of records based on query

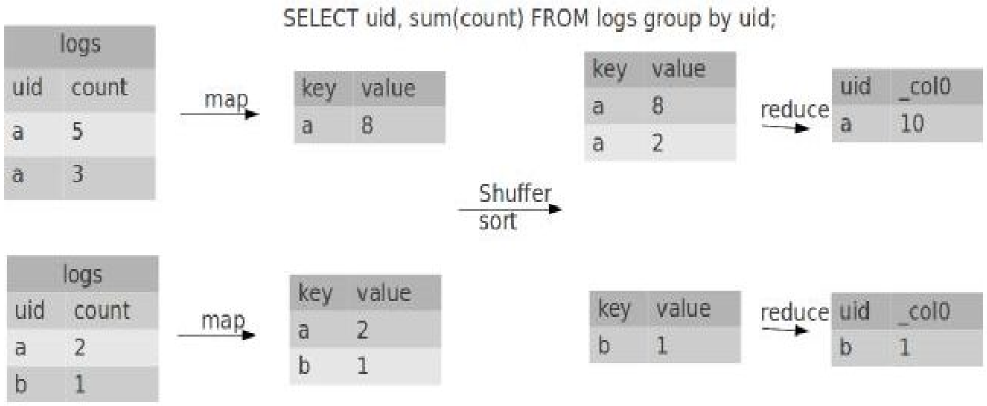
sale

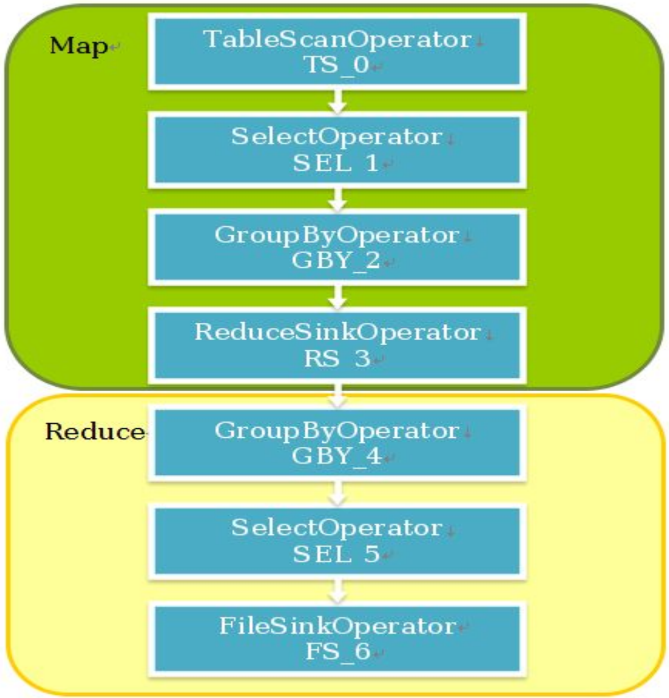
id	timestamp	productsk	storesk	amount	state
10005	2013-06-13T09:03:18	1192	497	$25.73	IL
10002	2013-06-13T09:03:06	4671	606	$67.12	MA
10003	2013-06-13T09:03:08	7224	174	$96.85	CA
10004	2013-06-13T09:03:12	9354	123	$67.76	CA
10001	2013-06-13T09:03:05	10739	359	$52.99	IL
10000	2013-06-13T09:03:05	16775	670	$70.50	CA

```
CREATE TABLE sale (
    id int, timestamp timestamp,
    productsk int, storesk int,
    amount decimal, state string
) STORED AS orc;
INSERT INTO sale AS SELECT * FROM staging SORT BY productsk;
```

ORCFile skipping speeds queries like
WHERE productsk = X, productsk IN (Y, Z); etc.

Explain - Simple example






```
EXPLAIN [EXTENDED|DEPENDENCY|AUTHORIZATION] hive_query
```

EXPLAIN - Multiple Stages

```sql
set hive.execution.engine=mr;
EXPLAIN
SELECT advertiser_id, advertiser_name,
    advertiser_mdm_id, advertiser_mdm_name, adgroup_id,
    COUNT(*) as keywords_count
FROM keywords_by_bucket
GROUP BY
    advertiser_id, advertiser_name,
    advertiser_mdm_id, advertiser_mdm_name, adgroup_id
ORDER BY keywords_count
```

```
STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-2 depends on stages: Stage-1
Stage-0 depends on stages: Stage-2

STAGE PLANS:
Stage: Stage-1
Map Reduce
Map Operator Tree:
    TableScan
    Select Operator
        expressions: advertiser_id (type: bigint), ...
        outputColumnNames: advertiser_id, ...
    Group By Operator
        aggregations: count()
        keys: advertiser_id (type: bigint), ...
        mode: hash
        outputColumnNames: _col0, ...
    Reduce Output Operator
        key expressions: _col0 (type: bigint), ...
        sort order: +++++
        Map-reduce partition columns: _col0 (type: bigint), ...
        value expressions: _col5 (type: bigint)
Reduce Operator Tree:
    Group By Operator
        aggregations: count(VALUE._col0)
        keys: KEY._col0 (type: bigint), ...
        mode: mergepartial
        outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5
        Statistics: Num rows: 53849497 Data size: 12062287380 Basic stats: COMPLETE Column stats: NONE
    File Output Operator
        compressed: true
    table:
        input format: org.apache.hadoop.mapred.SequenceFileInputFormat
        output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
        serde: org.apache.hadoop.hive.serde2.lazybinary.LazyBinarySerDe

Stage: Stage-2
```

Histogram Generation

- histogram_numeric(col, b)
  - Computes a histogram using b non-uniformly spaced bins
  - output - array<struct {'x','y'}>
  - (x,y) coordinates that represent the bin centers and heights
- Lateral View
  - Explode array into rows
- Data (20160209)
  - 183,010,223 AdGroups
  - 978,339,861 Keywords

```sql
SELECT
    CAST(ag_kw_hist.x as int) as bin_center,
    CAST(ag_kw_hist.y as int) as bin_height
FROM (
    SELECT histogram_numeric(keywords_count, 500) as adgroup_kw_hist
    FROM (
        SELECT adgroup_id, COUNT(*) as keywords_count
        FROM keywords_by_bucket
        GROUP BY
            advertiser_id, advertiser_name,
            advertiser_mdm_id, advertiser_mdm_name, adgroup_id
        ORDER BY keywords_count
    ) k
) hist
LATERAL VIEW explode(adgroup_kw_hist) exploded_hist as ag_kw_hist
```

Histogram Algorithm

http://www.jmlr.org/papers/volume11/ben-haim10a/ben-haim10a.pdf

The **update** procedure:
Given a histogram $(p_1, m_1), \ldots, (p_r, m_r)$, $p_1 < \ldots < p_r$ and a point p, the **update** procedure adds p to the set S represented by the histogram.

- If $p = p_i$ for some i, then increment m_i by 1. Otherwise:
- Add the bin $(p, 1)$ to the histogram, resulting in a histogram of $r+1$ bins $(q_1, k_1), \ldots, (q_{r+1}, k_{r+1})$, $q_1 < \ldots < q_{r+1}$.
- Find a point q_i such that $q_{i+1} - q_i$ is minimal.
- Replace the bins (q_i, k_i), (q_{i+1}, k_{i+1}) by the bin

$$\frac{q_i k_i + q_{i+1} k_{i+1}}{k_i + k_{i+1}}, k_i + k_{i+1} \quad .$$

Percentile

```sql
SELECT percentile(keywords_count,
       array(0.25, 0.5, 0.75, 0.8, 0.9, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0))
FROM
(
    SELECT adgroup_id, COUNT(*) as keywords_count
    FROM keywords_by_bucket
    GROUP BY
        advertiser_id, advertiser_name,
        advertiser_mdm_id, advertiser_mdm_name, adgroup_id
    ORDER BY keywords_count
) k
```

- percentile(col, array(...)) computes percentile values of the specified column
- output: array, use explode to flatten to individual rows
- Example: rank ad groups based on number of keywords
  - There are 183M distinct adgroups in keywords_by_bucket table
  - Output: [1.0,1.0,1.0,1.0,4.0,9.0,12.0,18.0,27.0,50.0,134972.0]

Percentil	Keywords Count	AdGroups Count
0.25	1	45,752,556
0.5	1	91,505,112
0.75	1	137,257,667
0.8	1	146,408,178
0.9	4	164,709,201
0.95	9	173,859,712
0.96	12	175,689,814
0.97	18	177,519,916
0.98	17	179,350,019
0.99	50	181,180,121
1	134972	183,010,223

https://docs.google.com/spreadsheets/d/1zlmZsejmKYFih0RsJjWEiky_PSN2cPaOzdhe7elDVdQ/edit#gid=185

Views

- Allows a query to be saved and treated like a table
- Reduces the query complexity by encapsulating nested queries in views
- Just a logical construct (an alias for a query) with no physical data behind it

Views - Example

- Creates a view to represent all MDM participated auctions
- collect_set does not accept non-primitive types, otherwise create a struct(...) instead of concat_ws(...) - more clean
- Simple Pattern - Group BY, Collect, FILTER, Explode followed by SPLIT
- Just use this view as regular table
- Can also parameterize date and mdm id

SELECT * FROM mdm_auctions_view;

```sql
CREATE VIEW IF NOT EXISTS mdm_auctions_view (
    event_guid,
    cb_canon_user_query,
    ad_listing_type,
    page_position,
    advertiser_mdm_id,
    advertiser_acct_id,
    advertiser_name,
    imps_adv_bid_usd,
    imps_advertiser_cost_usd,
    ad_title,
    ad_description,
    bidded_term_text,
    match_type_desc
) AS
    SELECT event_guid, cb_canon_user_query,
        adv_info[0] as ad_listing_type,
        adv_info[1] as page_position,
        cast(adv_info[2] as bigint) as advertiser_mdm_id,
        cast(adv_info[3] as bigint) as advertiser_acct_id,
        adv_info[4] as advertiser_name,
        cast(adv_info[5] as double) as imps_adv_bid_usd,
        cast(adv_info[6] as double) as imps_advertiser_cost_usd,
        adv_info[7] as ad_title,
        adv_info[8] as ad_description,
        adv_info[9] as bidded_term_text,
        adv_info[10] as match_type_desc
    FROM (
    SELECT event_guid, cb_canon_user_query, split(advertiser_info, "###") adv_info FROM (
        SELECT * FROM (
            SELECT
                event_guid,
                cb_canon_user_query,
                collect_set(advertiser_mdm_id) as competing_mdm_ids,
                collect_set(concat_ws("###",
                    ad_listing_type,
                    page_position,
                    cast(advertiser_mdm_id as string),
                    cast(advertiser_acct_id as string),
                    advertiser_name,
                    cast(imps_adv_bid_usd as string),
                    cast(imps_advertiser_cost_usd as string),
                    ad_title,
                    ad_description,
                    advertiser_bidded_phrase_canon_term,
                    match_type_desc
                )) as competing_advertisers_info
            FROM auctions
            WHERE date = '20160211'
            GROUP BY event_guid, cb_canon_user_query
        ) c
        WHERE array_contains(competing_mdm_ids, cast(16141 as bigint)) > 0
    ) competing LATERAL VIEW explode(competing_advertisers_info) cv AS advertiser_info
) advertiser_info_flattened
;
```

Enhanced Aggregation

- Grouping Sets
- Cubes and Rollup

HIVE Components

- Parser
  - Transform HIVE QL into Abstract Syntax Tree (AST)
- Semantic Analyzer - AST to DAG of Map/Reduce Tasks
  - Logical Plan Generator: AST to Operator Trees
  - Optimizer: Operator Trees To Operator Trees
  - Physical Plan Generation: Operator Trees to MapReduce Tasks
- Execution Libraries
  - Operator Implementation, UDF/UDAF/UDTF
  - SerDe, Object Inspector, Metastore
  - FileFormat & Record Reader

HIVE Compiler Overview

- Parser
    - Semantic Analyzer
        - Logical Plan Generation
            - Logical Optimizer
                - Physical Plan Generation
                    - Physical Optimizer