Advanced HiveQL

Ahmad Alkilani
www.pluralsight.com

pluralsight
hardcore developer training

Outline

- **Bucketing**
- **Sampling Data**
  - Bucket sampling
  - Block sampling
- **Joins**
  - Types of joins
  - Joins in depth
  - Join optimizations
- **Distributed Cache**
- **Advanced Hive Functions**
  - Table valued functions (UDTFs)
  - Lateral view
- **Extending Hive**
  - Creating our own UDF
  - Transformation script using Streaming
- **Windowing and Analytical/Ranking Functions**

Bucketing

- **Tables or Partitions can be bucketed**
- **Bucketing is an approach to distribute or cluster table data**
  - More efficient sampling
  - Better performance with Map-side joins
  - Used with partitioning or w/o when partitioning doesn't work for your data set
- **Buckets can also be sorted**
  - Sort-Merge-Bucket (SMB) joins

```
CREATE TABLE t1 (a INT, b STRING, c STRING)
CLUSTERED BY (b) INTO 256 BUCKETS
```

```
CREATE TABLE t1 (a INT, b STRING, c STRING)
PARTITIONED BY (dt STRING)
CLUSTERED BY (b) SORTED BY (c) INTO 64 BUCKETS
```

Bucketing

- Hive doesn't control or enforce bucketing on data loaded into table
- 2 approaches

set mapred.reduce.tasks = 64;

```
set mapred.reduce.tasks=64;
INSERT OVERWRITE TABLE t1
SELECT a, b, c FROM t2 CLUSTER BY
    b;
```

set hive.enforce.bucketing=true;

```
set hive.enforce.bucketing=true;
INSERT OVERWRITE TABLE t1
SELECT a, b, c FROM t2
```

- Number of reducers and hence number of output files equals the number of buckets.

- Sampling data becomes a simple task.

Bucket Sampling

- Hive supports sampling data from tables
- Can be applied to any table

Bucket Sampling Syntax

```
SELECT * FROM source TABLESAMPLE (BUCKET x OUT OF y [ON colname]);
```

```
SELECT * FROM page_views TABLESAMPLE (BUCKET 3 OUT OF 64 ON userid);
SELECT * FROM page_views TABLESAMPLE (BUCKET 3 OUT OF 64 ON rand());
```

```
CREATE TABLE page_views (userid INT, page STRING, views INT)
PARTITIONED BY (dt STRING)
CLUSTERED BY (userid) SORTED BY (dt) INTO 64 BUCKETS
```

Block Sampling

- Based on HDFS blocks (64/128/256 etc..)
- Percentage of data size (notice this is not # of rows)
- Returns at least the percentage specified
- Doesn't always work
  - Depends on compression and input format (CombineHiveInputFormat)

Bucket Sampling Syntax

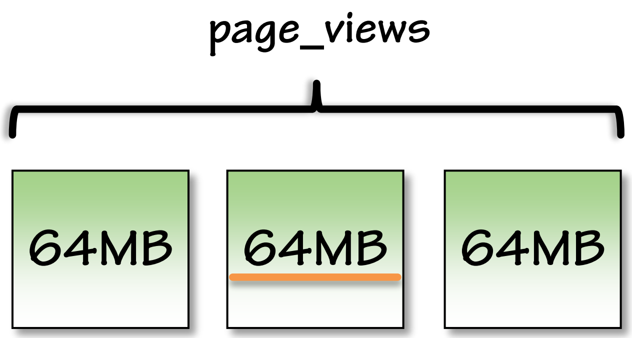
```
SELECT * FROM source TABLESAMPLE (n PERCENT);
```

```
SELECT * FROM source TABLESAMPLE (xM);
```

```
SELECT * FROM page_views TABLESAMPLE (0.1 PERCENT);
SELECT * FROM page_views TABLESAMPLE (90M);
```

```
SELECT * FROM source TABLESAMPLE (10 ROWS);
```

10 rows per input split



Joins

- **Join Types**
  - JOIN (Inner Join)
  - LEFT, RIGHT, FULL [OUTER] JOIN
  - LEFT SEMI JOIN
  - CROSS JOIN

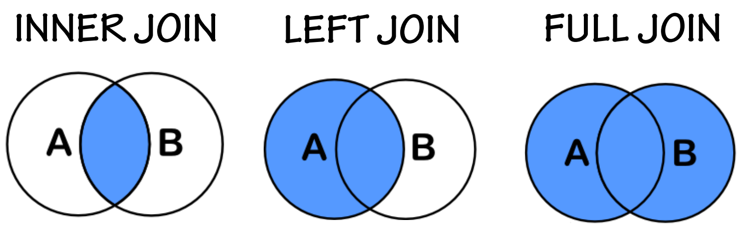
- **Equality joins only (equi-joins)**

- **Multiple tables can be joined in the same query**

Joins

JOIN (Inner Join)

```
SELECT a.val, b.val FROM a JOIN b ON (a.key = b.key);
```



LEFT, RIGHT, FULL [OUTER] JOIN

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key = b.key) JOIN c ON (c.key =
      a.key);
```

LEFT SEMI JOIN

```
SELECT a.val FROM a WHERE a.key IN (SELECT b.key FROM b) - Not Supported
SELECT a.val FROM a WHERE EXISTS (SELECT 1 FROM b WHERE b.key = a.key) - Not Supported


SELECT a.val FROM a LEFT SEMI JOIN b ON (a.key = b.key);
```

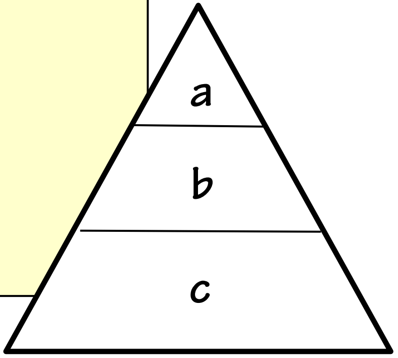
CROSS JOIN

```
SELECT a.*, b.* FROM a CROSS JOIN
      b;
```

Joins - In Depth

STREAM

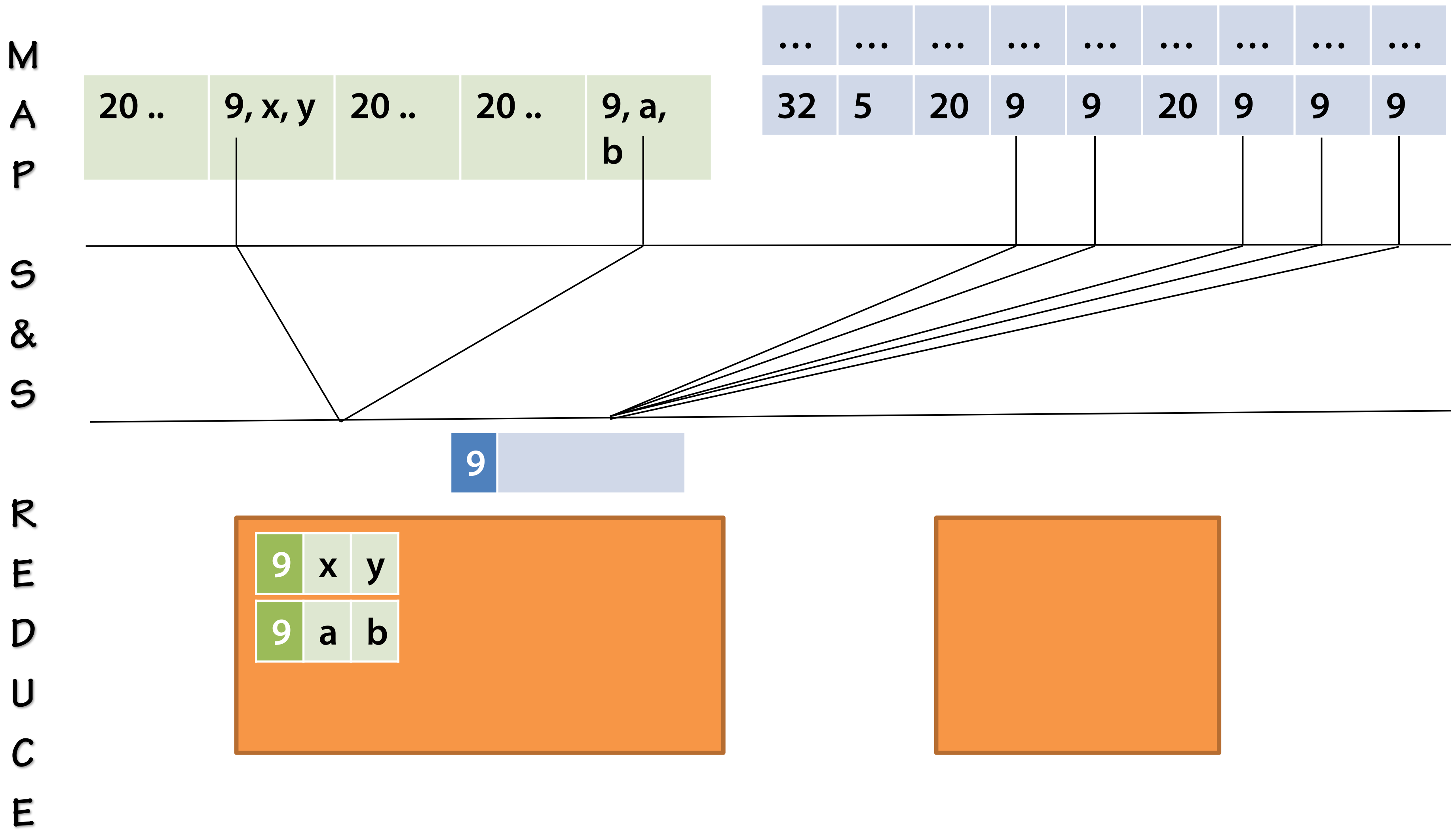
```
SELECT a.*, b.*, c.*
FROM a
LEFT JOIN b ON (a.key = b.key);
JOIN c ON (a.xyz = c.xyz)
```



```
SELECT STREAMABLE(a) a.*, b.*, c.*
FROM a
LEFT JOIN b ON (a.key = b.key);
JOIN c ON (a.xyz = c.xyz)
```

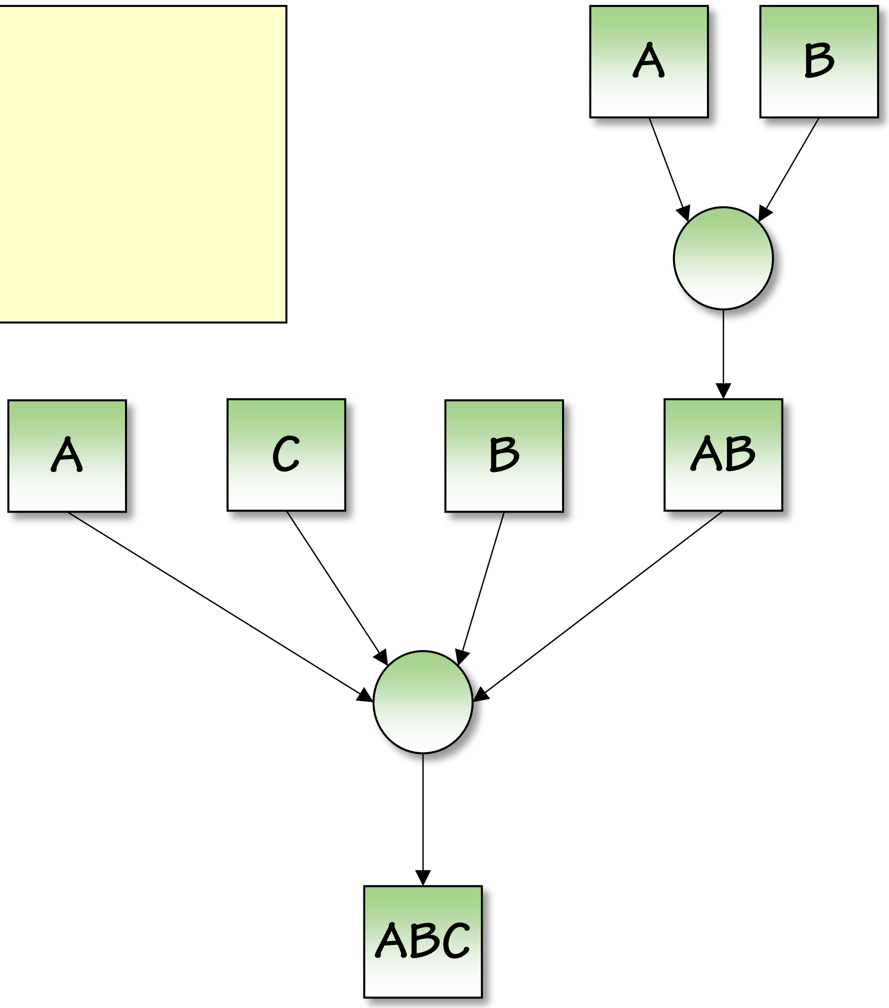
Joins - In Depth

How Joins Work



Joins - Merging MR Jobs

```
SELECT a.*, b.*, c.*
FROM a
LEFT JOIN b ON (a.key = b.key)
INNER JOIN c ON (a.key = c.key);
```



Map-side Joins

- All tables involved in a join are small enough to fit into memory except 1 which is streamed through the mapper

- Hash table is used

```
SELECT MAPJOIN(b) a.*, b.* FROM a
JOIN b ON (a.key = b.key);
```

- No Full or Right Outer Joins
- No UNIONs between multiple queries

```
set hive.auto.convert.join=true
SELECT a.*, b.* FROM a
JOIN b ON (a.key = b.key);
```

Map-side Joins for Bucketed Tables

Buckets can be joined with each other (Map-side Join) when:

- Tables being joined are bucketed on join columns (Clustered)
- Number of buckets in one table is a multiple of the number of buckets in the other table
- Set `hive.optimize.bucketmapjoin=true`

Sort Merge Join

- Tables being joined are bucketed on join columns (Clustered)
- They have the same number of buckets
- Buckets are also sorted
- Set:

```
hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;
hive.optimize.bucketmapjoin = true;
hive.optimize.bucketmapjoin.sortedmerge = true;
```

Distributed Cache

- **An approach used by MapReduce to distribute files across data nodes**
- **Provides a means for data nodes to access files local to the data node itself (cached copy)**
- **Typically used with**
  - Text files
  - Archives (compressed files)
  - Jars and other program files
- **Used to distribute hash archive of a table for Map-Side joins**

```
ADD FILE mydata.txt;
```

```
ADD ARCHIVE sendme.zip;
```

```
ADD JAR myprogram.jar;
```

```
LIST FILES|JARS|ARCHIVES [filepath];
```

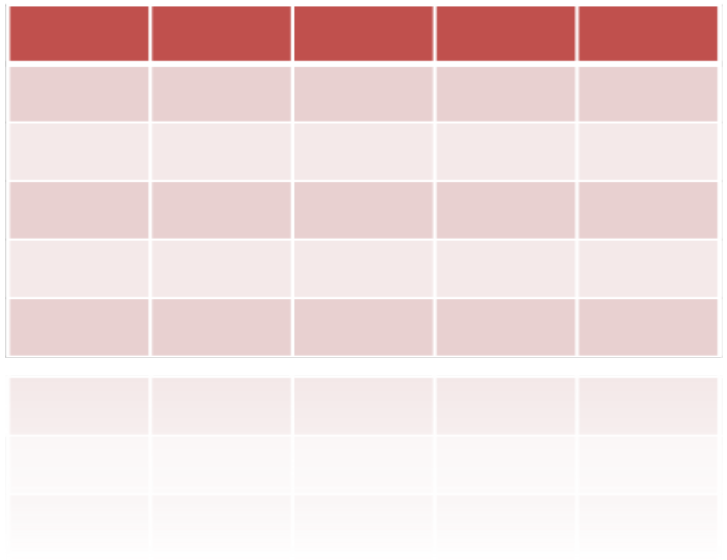


Table-Generating Functions (UDTF)

Advanced Hive Functions

Built-in Table Generating Functions

movie_id	title	actors			
620	The King's Speech	Colin Firth	Geoffrey Rush	Helena Bonham Carter	Freya Wilson
621	Elysium	Matt Damon	Jodie Foster	Sharlto Copley	

Explode()

- Takes array as input
- No other expressions allowed in SELECT
- Can't be nested
- GROUP BY / CLUSTER BY / DISTRIBUTE BY / SORT BY not supported
- Explodes elements of array as separate rows

Colin Firth	Geoffrey Rush	Helena Bonham Carte	Freya Wilson

```
SELECT explode(actors) AS a FROM
    movies;
```

Array<int> metric
[8,9,10]
[6,3]

(int) myNewCol
8
9
10
6
3

Lateral View

- Takes UDTF function as input
- Provides virtual table for accessing combined results

```
SELECT a, b, columnAlias
FROM baseTable
LATERAL VIEW UDTF(expression) tableAlias AS columnAlias;
```

```
SELECT a, b, col1, col2
FROM baseTable
LATERAL VIEW UDTF(x) t1 AS col1
LATERAL VIEW UDTF(col1) t2 AS col2;
```

Lateral View

movie_id	title	actors			
620	The King's Speech	Colin Firth	Geoffrey Rush	Helena Bonham Carter	Freya Wilson
621	Elysium	Matt Damon	Jodie Foster	Sharlto Copley	

```
SELECT movie_id, title, actor
FROM movies LATERAL VIEW explode(actors) actorTable AS actor;
```

movie_id	title	actor
620	The King's Speech	Colin Firth
620	The King's Speech	Geoffrey Rush
620	The King's Speech	Helena Bonham Carter
620	The King's Speech	Freya Wilson
621	Elysium	Matt Damon
621	Elysium	Jodie Foster
621	Elysium	Sharlto Copley

Outer Lateral Views

movie_id	title	actors		
620	Movie A			
621	Elysium	Matt Damon	Jodie Foster	Sharlto Copley

```
SELECT movie_id, title, actor
FROM movies LATERAL VIEW OUTER explode(actors) actorTable AS actor;
```

movie_id	title	actor
620	Movie A	NULL
621	Elysium	Matt Damon
621	Elysium	Jodie Foster
621	Elysium	Sharlto Copley

Writing your own functions

Extending Hive

Creating a UDF

- **Import necessary packages**
  - import org.apache.hadoop.hive.ql.exec.UDF;
  - import org.apache.hadoop.hive.ql.exec.Description;

  Anything you need as part of your UDF
  - import org.apache.hadoop.io.Text
  - import java.util.*;

- **Add annotations**
  - Description, Deterministic, Stateful, DistinctLike

- **Extend the UDF class**

- **Provide an implementation of the evaluate function possibly with multiple overloads**

Creating a UDF

- **Import necessary packages**
  - import org.apache.hadoop.hive.ql.exec.UDF;
  - import org.apache.hadoop.hive.ql.exec.Description;

  Anything you need as part of your UDF
  - import org.apache.hadoop.io.Text
  - import java.util.*;

- **Add annotations**
  - Description, Deterministic, Stateful, DistinctLike

- **Extend the UDF class**

- **Provide an implementation of the evaluate function possibly with multiple overloads**

Creating a UDF

- **Compile and package code**

```
javac -target 1.6 -cp $(ls /usr/lib/hive/lib/hive-
   exec*.jar):/usr/lib/hadoop/hadoop-core.jar com/pluralsight/udf/MyReverse.java
```

```
jar -cf myudf.jar com/pluralsight/udf/MyReverse.class
```

- **Tell Hive about the JAR file. Use ADD JAR /path/to/jar/myudf.jar**
  - Adds JAR to distributed cache & classpath

- **Create TEMPORARY FUNCTION and reference class**

What about that TEMPORARY function

- **Function only exists in current user's session**

- **Use the -i option when launching hive from the command line**
  - Provide an initialization file

- **Use the .hiverc file**
  - User's home directory
  - Hive's bin directory /usr/lib/hive/bin/

Distributed Cache, Again?

- Hive functions are added to the distributed cache

- Accessing files on the distributed cache is just a matter of referencing the file

```
File f = new File(“./samplefile.csv”);
```

Hadoop Streaming

- **Customize Hive using a different language**
- **Data is Streamed through standard in/out**
- **TRANSFORM**
- **MAP, REDUCE**
  - Don’t confuse with actual Map and Reduce, these are just syntactical sugar
  - Primarily introduced to minimize the work required to create Reduce code by eliminating boilerplate code
- **Cluster By, Distribute By, Sort By**
  - Essential with streaming for performance
  - Part of the algorithm to solve the problem

TRANSFORM

- **Syntax**

```
SELECT TRANSFORM (col1 [,col2… coln])
USING 'Code File|Program' [AS (list of columns [and casts])]
FROM SourceTable;
```

- **Columns are sent as tab separated string (default)**

- **Null values are replaced with literal “\N”**

- **Specifying list of output columns is optional, if not provided:**
  - First column is the key
  - Remaining string is the value, even if there are multiple tabs (columns)
  - Key column referenced using key

TRANSFORM (2)

```
SELECT TRANSFORM (col1 [,col2… coln])
USING 'Code File|Program' [AS (list of columns [and casts])]
FROM SourceTable;
```

```
SELECT TRANSFORM (movie_title)
USING '/bin/sed "s/[^ ][^ ]*/(&)/g"' AS movie_title_parantheses
FROM pluralsight.movies;
```

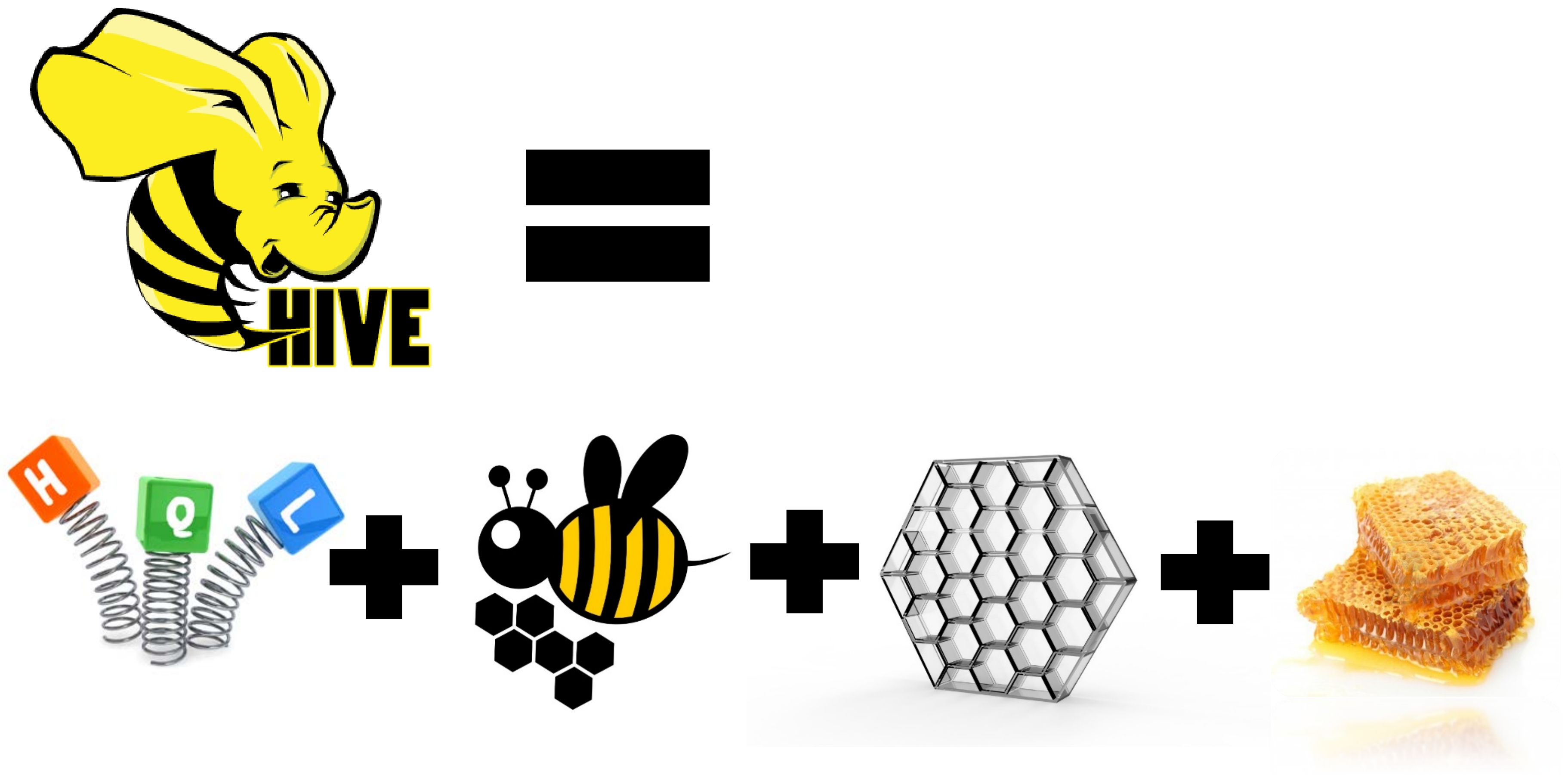
*Example used here was inspired by the O'REILLY book "Programming Hive" and www.grymoire.com/Unix/Sed.html

Windowing and Analytics Functions

- LEAD/LAG
- FIRST_VALUE
- LAST_VALUE
- PARTITION BY
- OVER clause

ID	Basket	Contents	Quantity
1	Susan	Apple	6
2	Susan	Banana	12
3	Mike	Pear	5
4	Mike	Milk	2
5	Mike	Eggs	12
6	John	Cereal	1
7	John	Apple	7
8	John	Milk	3
9	John	Cheese	1
10	John	Broccoli	2

- WINDOW clause to provide window specification
- RANK, ROW_NUMBER, DENSE_RANK
  CUME_DIST, PERCENT_RANK, NTILE



Demo

Putting it all together

Problem : Time On Site

How much time does each user spend on my site?

- **Sort log records based on user**
- **Get the difference between log times of each record**
- **Add it up for each user**

Windowing and Analytics Functions

- LEAD/LAG
- FIRST_VALUE
- LAST_VALUE
- PARTITION BY
- OVER clause

ID	Basket	Contents	Quantity
1	Susan	Apple	6
2	Susan	Banana	12
3	Mike	Pear	5
4	Mike	Milk	2
5	Mike	Eggs	12
6	John	Cereal	1
7	John	Apple	7
8	John	Milk	3
9	John	Cheese	1
10	John	Broccoli	2

- WINDOW clause to provide window specification
- RANK, ROW_NUMBER, DENSE_RANK
  CUME_DIST, PERCENT_RANK, NTILE

Summary

- **Bucketing and Table Sampling**
- **Joins**
  - Join Types
  - Map Side Joins
  - Sort Merge Bucket Join
- **Distributed Cache**
- **Table Valued Functions**
  - Explode
  - Lateral View
- **Extended Hive with a User Defined Function**
- **Hadoop Streaming and Hive Transform**
- **Windowing and Analytics functions**