Hive Query Language

Ahmad Alkilani
www.pluralsight.com

pluralsight
hardcore developer training

Outline

- **Data Types**
- **Load and Organize Data**
  - Managed/External Partitioned Tables
  - Dynamic Partition Inserts
- **Single Scan-Multiple Inserts**
- **Hive Functions, Aggregates, Group By, Cube, Rollup, Having**
- **Sorting and Clustering Results**
- **Using the CLI in the real world**
  - Batch mode
  - Variable Substitution

Primitive Data Types

Numeric

- TINYINT, SMALLINT, INT, BIGINT
- FLOAT
- DOUBLE
- DECIMAL – Starting Hive 0.11

Date/Time

- TIMESTAMP starting Hive 0.8
  - Strings must be in format "YYYY-MM-DD HH:MM:SS.fffffffff"
  - Integer types as UNIX timestamp in seconds from UNIX epoch
  - Floating point types same as Integer with decimal precision
- DATE starting Hive 0.12

Misc.

- BOOLEAN
- STRING
- BINARY

Complex/Collection Types

Type	Syntax
Arrays	ARRAY<data_type>
Maps	MAP<primitive_type, data_type>
Struct	STRUCT<col_name : data_type [COMMENT col_comment],
Union Type	UNIONTYPE<data_type, data_type, …>

```
CREATE TABLE movies (
    movie_name string,
    participants ARRAY<string>,
    release_dates MAP<string, timestamp>,
    studio_addr STRUCT<state:string, city:string, zip:string, streetnbr:int, streetname:string,
    unit:string>,
    complex_participants MAP<string, STRUCT<address:string, attributes MAP<string,
    string>>>
    misc UNIONTYPE<int, string, ARRAY<double>>
);
```

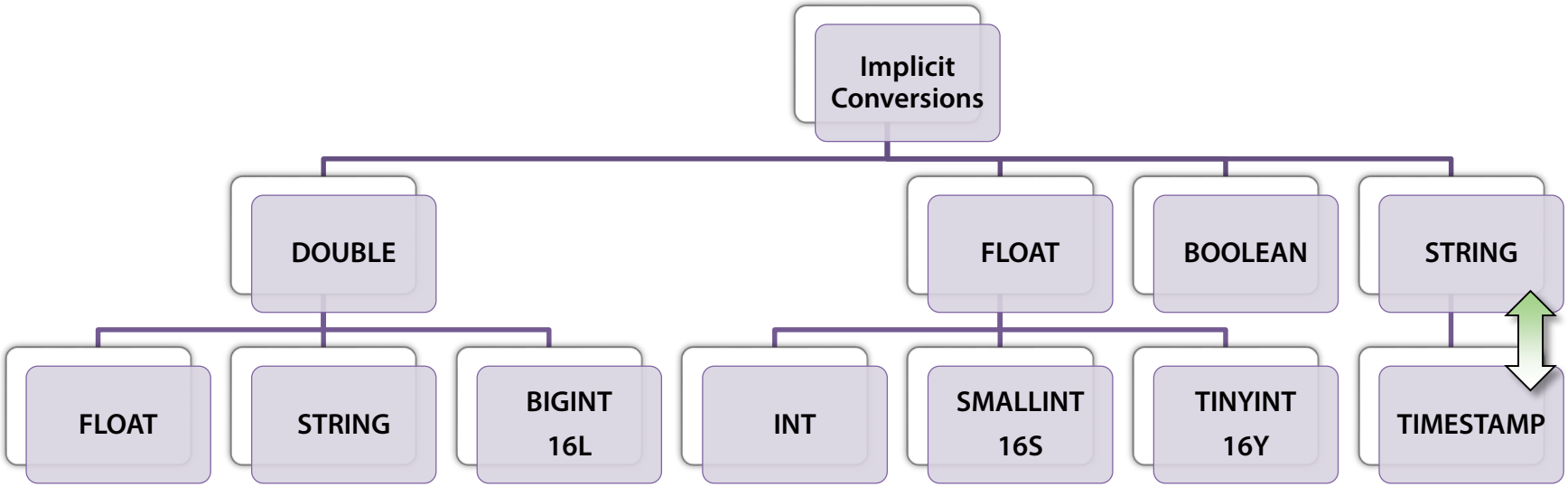
Complex/Collection Types

```
CREATE TABLE movies (
    movie_name string,
    participants ARRAY<string>,
    release_dates MAP<string, timestamp>,
    studio_addr STRUCT<state:string, city:string, zip:string, streetnbr:int, streetname:string,
    unit:string>,
    complex_participants MAP<string, STRUCT<address:string, attributes MAP<string,
    string>>>
    misc UNIONTYPE<int, string, ARRAY<double>>
```

“Inception”	2010-07-16 00:00:00	91505	“Dark Green”	{0:800}
“Planes”	2013-08-09 00:00:00	91505	“Green”	{3:[1.0, 2.3, 5.6]}

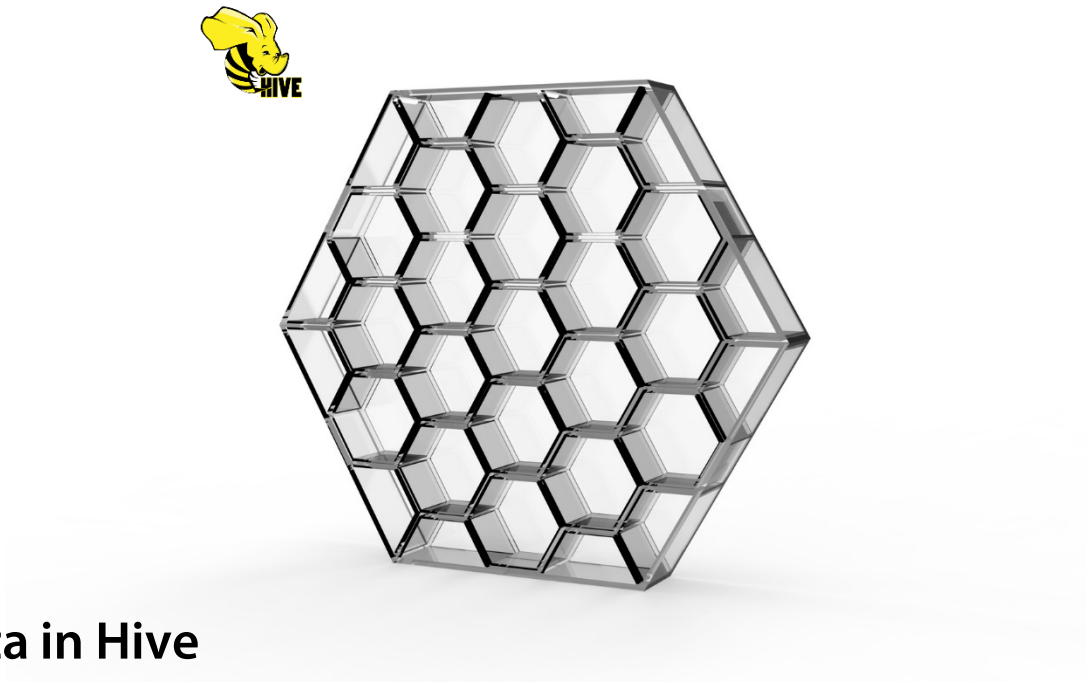
```
SELECT movie_name,
    participants[0],
    release_dates[“USA”],
    studio_addr.zip,
    complex_participants[“Leonardo
    DiCaprio”].attributes[“fav_color”],
    misc
FROM movies;
```

Type Conversions



Explicit Conversions

- CAST('13' AS INT)
- CAST('This results in NULL' AS INT)
- CAST('2.0' AS FLOAT)
- CAST(CAST(binary_data AS STRING) AS DOUBLE)



Loading and organizing data in Hive

Hive Query Language

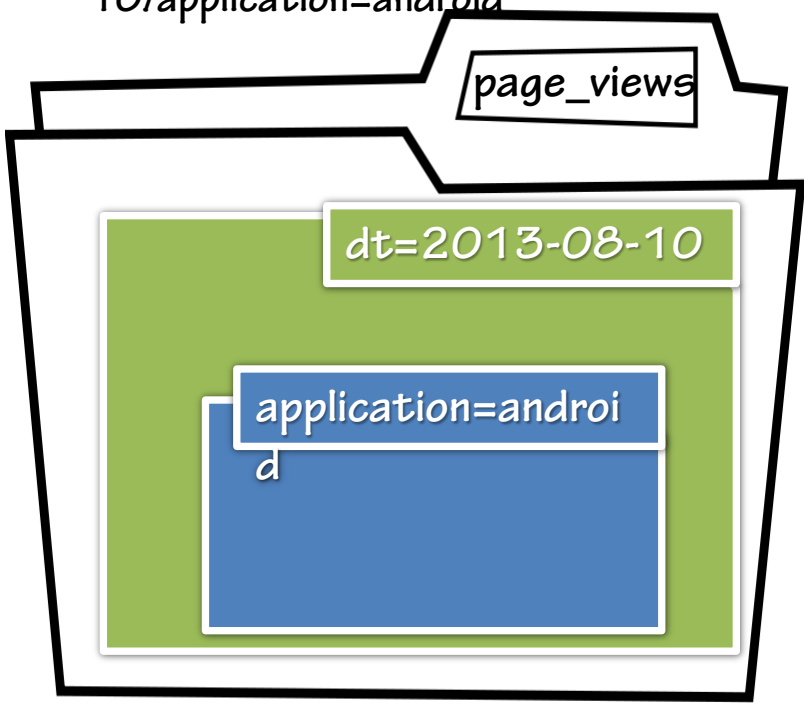
Table Partitions

- **Managed Partitioned Tables**

```
CREATE TABLE page_views ( eventTime STRING, userid STRING, page STRING)
PARTITIONED BY(dt STRING, applicationtype STRING)
STORED AS TEXTFILE;
```

/apps/hive/warehouse/page_views

/apps/hive/warehouse/page_views/dt=2013-08-10/application=android




```
LOAD DATA INPATH
      '/mydata/android/Aug_10_2013/pageviews/'
INTO TABLE page_views
PARTITION (dt = '2013-08-10', applicationtype = 'android');
```

```
LOAD DATA INPATH
      '/sample/android/Aug_10_2013/pageviews/'
OVERWRITE INTO TABLE page_views
PARTITION (dt = '2013-08-10', applicationtype = 'android');
```

Table Partitions

- **Virtual Partition Columns**

```
CREATE TABLE page_views ( eventTime STRING, userid STRING, page STRING)
PARTITIONED BY(dt STRING, applicationtype STRING)
STORED AS TEXTFILE;
```

eventTime	STRING
userid	STRING
page	STRING
dt	STRING
applicationtype	STRING

```
SELECT dt as eventDate, page, count(*) as pviewCount FROM page_views
WHERE applicationtype = 'iPhone';
```

Table Partitions

- **External Partitioned Tables**

```
CREATE EXTERNAL TABLE page_views ( eventTime STRING, userid STRING,
    page STRING)
PARTITIONED BY(dt STRING, applicationtype STRING)
STORED AS TEXTFILE;
```

eventTime	STRING
userid	STRING
page	STRING
dt	STRING
applicationtype	STRING

```
ALTER TABLE page_views ADD PARTITION (dt='2013-09-09', applicationtype='Windows Phone 8')
LOCATION '/somewhere/on/hdfs/data/2013-09-09/wp8';
```

```
ALTER TABLE page_views ADD PARTITION (dt='2013-09-09', applicationtype='iPhone')
LOCATION 'hdfs://NameNode/somewhere/on/hdfs/data/iphone/current';
```

```
ALTER TABLE page_views ADD IF NOT EXISTS
PARTITION (dt='2013-09-09', applicationtype='iPhone') LOCATION
    '/somewhere/on/hdfs/data/iphone/current'
PARTITION (dt='2013-09-08', applicationtype='iPhone') LOCATION
    '/somewhere/on/hdfs/data/prev1/iphone'
PARTITION (dt='2013-09-07', applicationtype='iPhone') LOCATION
    '/somewhere/on/hdfs/data/iphone/prev2';
```

Demo

Multiple Inserts

- Interchangeability of blocks

```
FROM movies
SELECT *;
```

- Syntax

```
FROM from_statement
INSERT OVERWRITE TABLE table1 [PARTITION (partcol1=val1, partcol2=val2)] select_statement1
INSERT INTO TABLE table2 [PARTITION (partcol1=val1, partcol2=val2) [IF NOT EXISTS]]
    select_statement2
INSERT OVERWRITE DIRECTORY 'path' select_statement3;
```

- Extract action and horror movies into tables for further processing

```
FROM movies
INSERT OVERWRITE TABLE horror_movies SELECT * WHERE horror = 1 AND release_date = '8/23/2013'
INSERT INTO action_movies SELECT * WHERE action = 1 AND release_date = '8/23/2013';
```

```
FROM (SELECT * FROM movies WHERE release_date = '8/23/2013') src
INSERT OVERWRITE TABLE horror_movies SELECT * WHERE horror = 1
INSERT INTO action_movies SELECT * WHERE action = 1;
```

Dynamic Partition Inserts

```
CREATE TABLE views_stg (eventTime STRING, userid STRING)
PARTITIONED BY(dt STRING, applicationtype STRING, page STRING);
```

```
FROM page_views src
INSERT OVERWRITE TABLE views_stg PARTITION (dt='2013-09-13', applicationtype='Web', page='Home')
    SELECT src.eventTime, src.userid WHERE dt='2013-09-13' AND applicationtype='Web', page='Home'
INSERT OVERWRITE TABLE views_stg PARTITION (dt='2013-09-14', applicationtype='Web', page='Cart')
    SELECT src.eventTime, src.userid WHERE dt='2013-09-14' AND applicationtype='Web', page='Cart'
INSERT OVERWRITE TABLE views_stg PARTITION (dt='2013-09-15', applicationtype='Web',
    page='Checkout')
    SELECT src.eventTime, src.userid WHERE dt='2013-09-15 AND applicationtype='Web', page='Checkout'
```

```
FROM page_views src
INSERT OVERWRITE TABLE views_stg PARTITION (applicationtype='Web', dt, page)
SELECT src.eventTime, src.userid, src.dt, src.page WHERE applicationtype='Web'
```

- Dynamically determine partitions to create and populate
- Use input data to determine partitions

Dynamic Partition Inserts

- **Default maximum dynamic partitions = 1000**
  - hive.exec.max.dynamic.partitions
  - hive.exec.max.dynamic.partitions.pernode
- **Enable/Disable dynamic partition inserts**
  - hive.exec.dynamic.partition=true
- **Use strict mode when in doubt**
  - hive.exec.dynamic.partition.mode=strict
- **Increase max number of files a data node can service in (hdfs-site.xml)**
  - dfs.datanode.max.xcievers=4096

Table Partitions

- Partitions for managed tables created by loading data into table
- LOCATION for EXTERNAL partitioned tables is optional
- Advantages to using same directory structure of managed tables
  - Apache Hive
    - MSCK REPAIR TABLE table_name;
  - Amazon's Elastic Map Reduce
    - ALTER TABLE table_name RECOVER PARTITIONS;
- Virtual columns and column name collision
- ALTER TABLE ADD PARTITION isn't restricted to managed tables
- ALTER TABLE table_name [PARTITION spec] SET LOCATION "new location"
- Not everything results in partition pruning
- Data is in lowest level, leaf, directory
- When filter doesn't show in explain plan that means partition pruning was used to service the predicate.

Data Retrieval

Hive Query Language

Group By

```
SELECT
   a, b, SUM(c)
FROM
   t1
GROUP BY
   a, b
```

```
SELECT
   a, SUM(c)
FROM
   t1
GROUP BY
   a
```

a	b	c
1	H	10
2	A	10
1	H	20
1	B	10
1	S	10

a	b	_c0
1	B	10
1	H	30
1	S	10
2	A	10

a	_c0
1	50
2	10

Grouping Sets, Cube, Rollup

```
SELECT a, b, SUM(c) FROM t1 GROUP BY a, b GROUPING SETS ((a,b),a)
```

```
SELECT a, b, SUM(c) FROM t1 GROUP BY a, b
UNION ALL
SELECT a, NULL, SUM(c) FROM t1 GROUP BY a
```

```
SELECT a, b, SUM(c) FROM t1 GROUP BY a, b GROUPING SETS (a,b,())
```

```
SELECT a, NULL, SUM(c) FROM t1 GROUP BY a
UNION ALL
SELECT NULL, b, SUM(c) FROM t1 GROUP BY b
UNION ALL
SELECT NULL, NULL, SUM(c) FROM t1
```

Grouping Sets, Cube, Rollup

Cube

```
SELECT a, b, c, SUM(d) FROM t1 GROUP BY a, b WITH CUBE
```

```
SELECT a, b, c, SUM(d) FROM t1 GROUP BY a, b, c GROUPING SETS
((a,b,c),(a,b),(b,c),(a,c),a,b,c,())
```

Rollup

```
SELECT a, b, c, SUM(d) FROM t1 GROUP BY a, b WITH ROLLUP
```

```
SELECT a, b, c, SUM(d) FROM t1 GROUP BY a, b, c GROUPING SETS
((a,b,c),(a,b),a,())
```

Functions in Hive

- **Built -in Functions**
  - Mathematical
  - Collection
  - Type conversion
  - Date
  - Conditional
  - String
  - Misc.
  - xPath
- **UDAFs**
- **UDTFs**

Built-in Functions

- **Mathematical**

```
SELECT rand(), a FROM t1; SELECT rand(3), rand(a) FROM t1;
SELECT pow(a, b) FROM t2; SELECT tan(a) FROM t3;

abs(double a)
round(double a, int d)
floor(double a)
```

- **Collection**

```
size(Map<K.V>)
map_keys(Map<K.V>)
map_values(Map<K.V>)

SELECT array_contains(a, ‘test’) FROM t1;
```

Built-in Functions

- **Date**

```
unix_timestamp()
year(string d), month(string d), day(string d), hour, second
datediff(string enddate, string startdate)
date_add(string startdate, int days)
date_sub(string startdate, int days)
to_date(string timestamp)
```

- **Conditional**

```
SELECT IF(a = b, ‘true result’, ‘false result’) FROM t1;
SELECT COALESCE(a, b, c) FROM t1;
SELECT CASE a WHEN 123 THEN ‘first’ WHEN 456 THEN ‘second’
   ELSE ‘none’ END FROM t1;
SELECT CASE WHEN a = 13 THEN c ELSE d END FROM t1;
```

Built-in Functions

- **String**

```
SELECT concat(a, b) FROM t1; SELECT concat_ws(sep, a, b) FROM t1;
SELECT regex_replace("Hive Rocks", "ive", "adoop") FROM dummy;
```

```
substr(string|binary A, int start)
substring(string|binary A, int start, int length)
```

```
sentences(string str, string lang, string locale)

SELECT sentences("Loving this course! Hive is awesome.") FROM dummy;
```

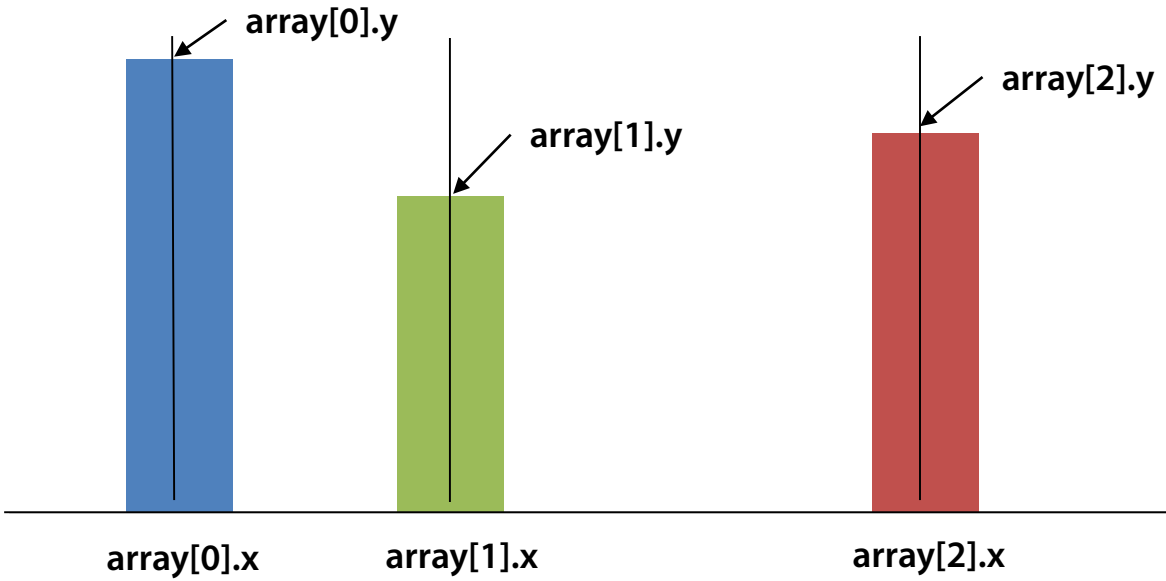
(("Loving", "this", "course"), ("Hive", "is", "awesome"))

Built-in Aggregate Functions (UDAFs)

```
COUNT(*), COUNT(expr), COUNT(DISTINCT expr)
SUM(col), SUM(DISTINCT col)

AVG, MIN, MAX, VARIANCE, STDDEV_POP

HISTOGRAM_NUMERIC(col, b)
   returns array<struct {‘x’, ‘y’}>
```



HAVING & GROUP BY

- **Having Syntax**

```
SELECT
   a, b, SUM(c)
FROM
   t1
GROUP BY
   a, b
HAVING
   SUM(c) > 2
```

- **Group By on Function**

```
SELECT
   CONCAT(a,b) as r
 , SUM(c)
FROM
   t1
GROUP BY
   CONCAT(a,b)
HAVING
   SUM(c) > 2
```

Sorting in Hive

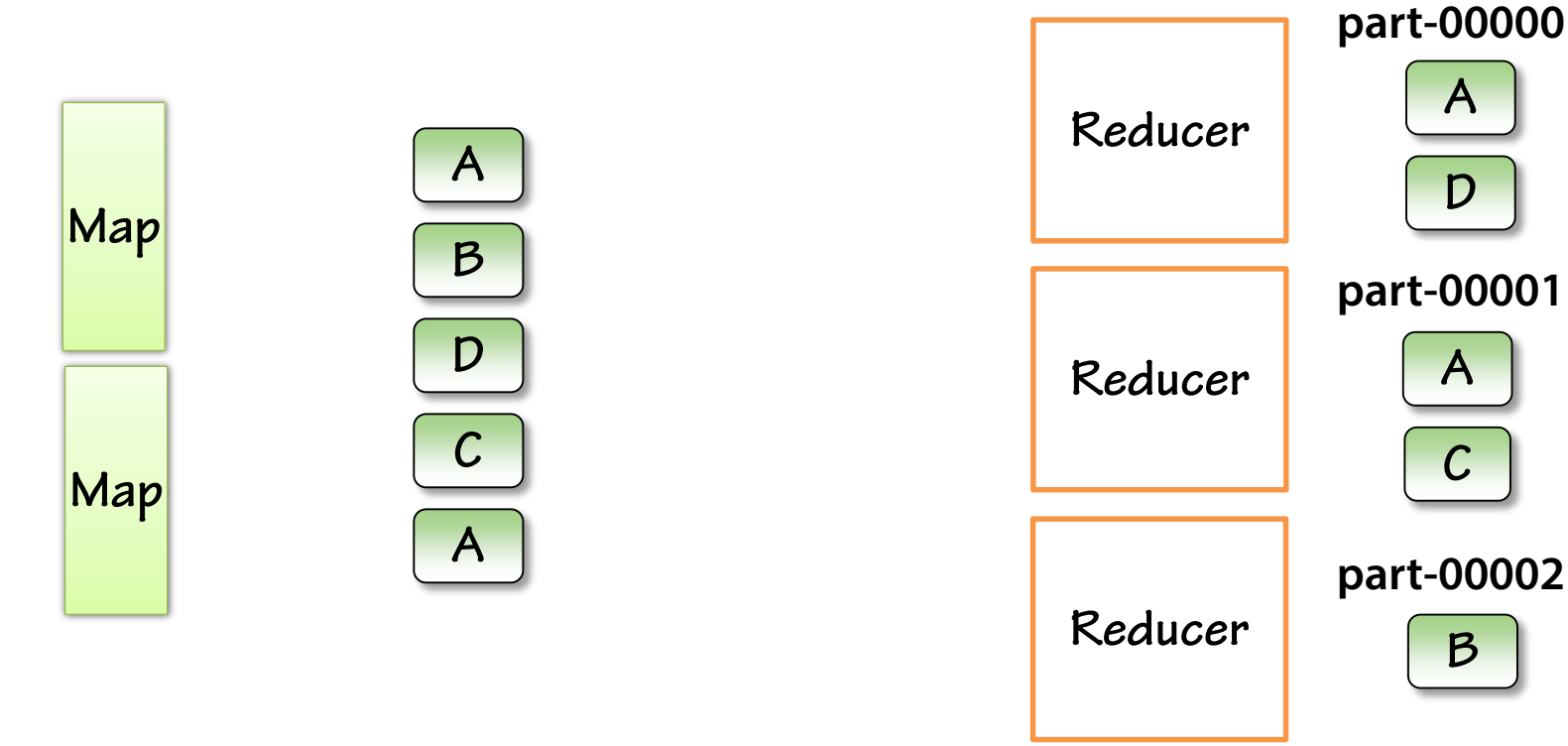
ORDER BY

```
SELECT x, y, z FROM t1 ORDER BY x ASC
```

Sorting in Hive

SORT BY

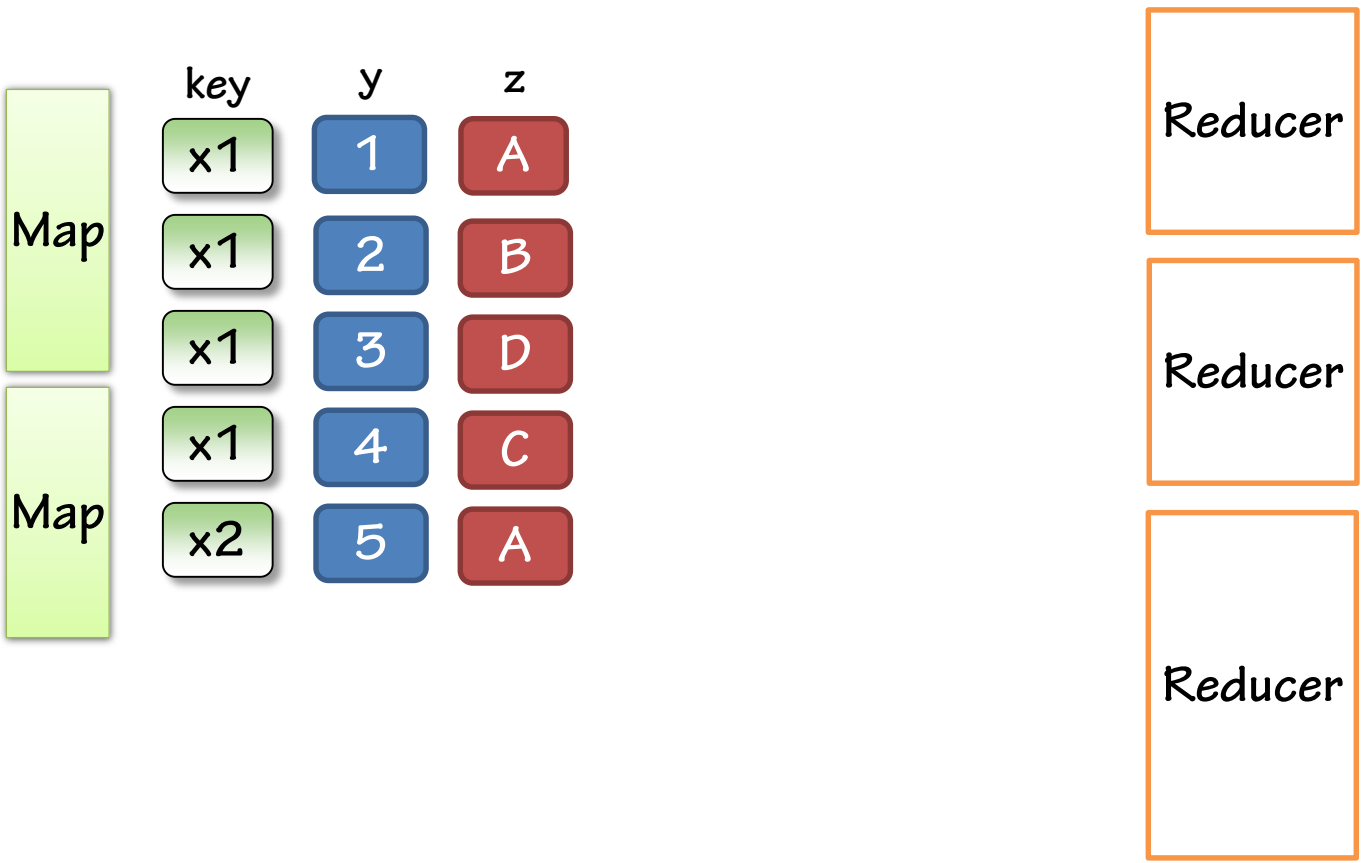
```
SELECT x, y, z FROM t1 SORT BY x
```



Controlling Data Flow

DISTIRBUTE BY

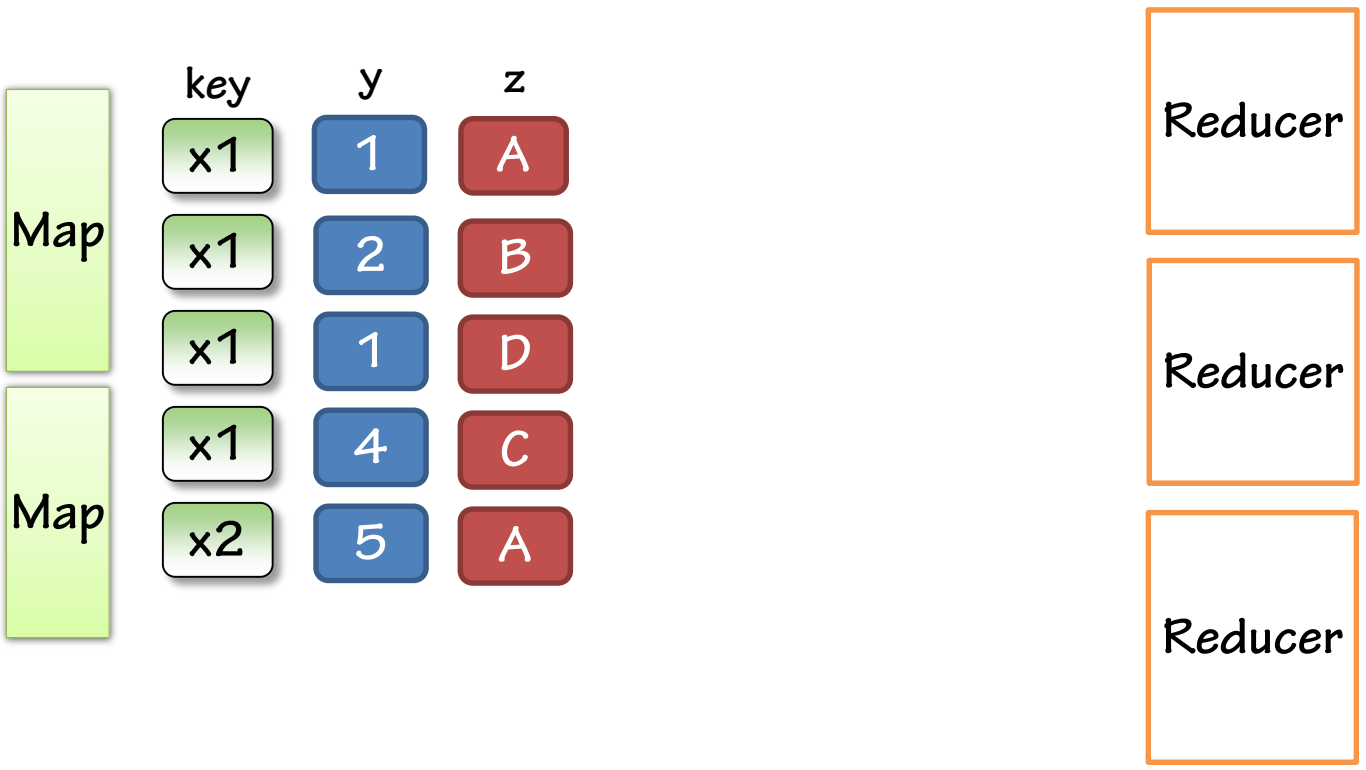
```
SELECT x, y, z FROM t1 DISTRIBUTE BY y
```



Controlling Data Flow

DISTIRBUTE BY

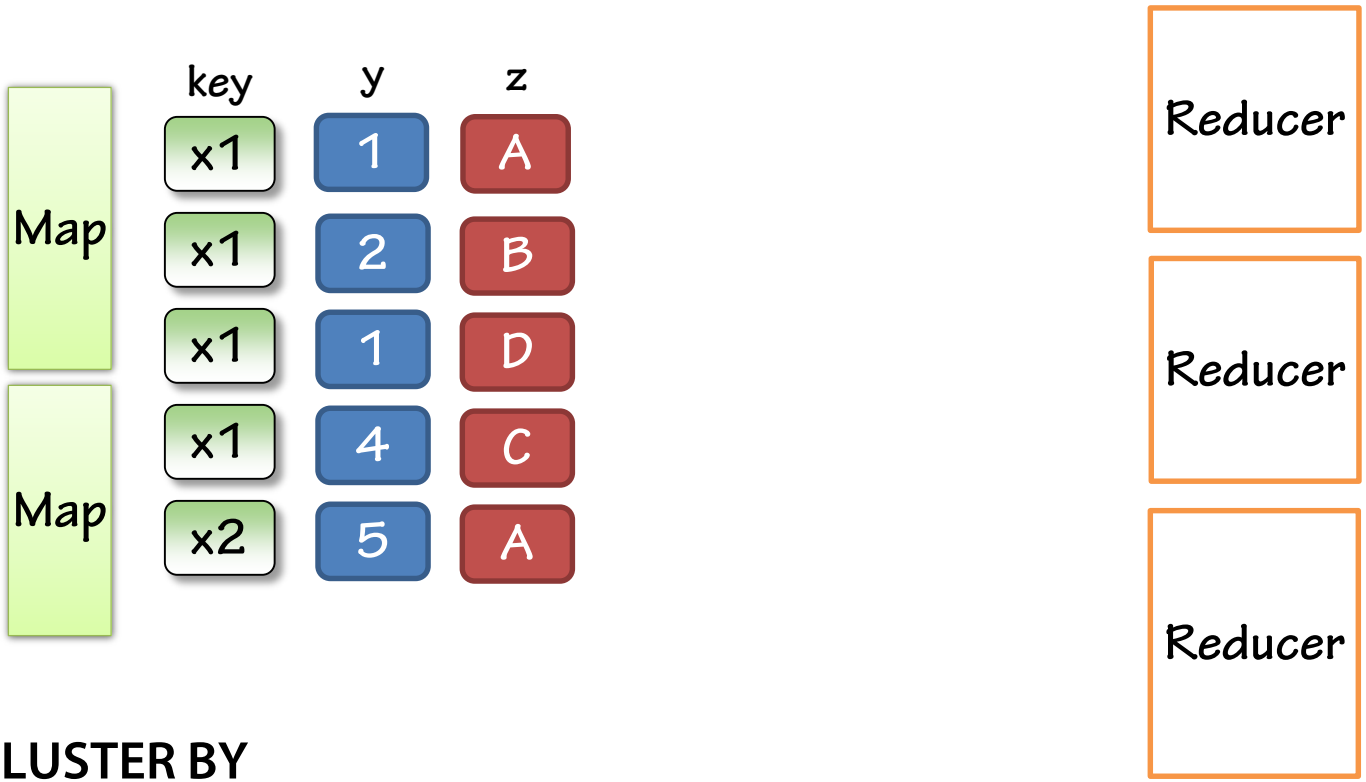
```
SELECT x, y, z FROM t1 DISTRIBUTE BY y
```



Controlling Data Flow

DISTIRBUTE BY with SORT BY

```
SELECT x, y, z FROM t1 DISTRIBUTE BY y SORT BY z
```



CLUSTER BY

```
SELECT x, y, z FROM t1 CLUSTER BY y
```

Command line options and variable substitution

Hive CLI

The CLI

- **hive**
  - hive -e 'select a, b, from t1 where c = 15'
  - hive -S -e 'select a, b from t1' > results.txt
  - hive -f /my/local/file/system/get-data.sql

-e and -f run hive in batch mode

- **Variable Substitution**

  4 namespaces
  - hivevar
    - -d, --define , --hivevar
    - set hivevar:name=value
  - hiveconf
    - --hiveconf
    - set hiveconf:property=value
  - system
    - set system:property=value
  - env
    - set env:property=value

```
$ hive -d srctable=movies
hive> set hivevar:cond=123;
hive> select a,b,c from pluralsight.${hivevar:srctable}
      where a = ${hivevar:cond};

$ hive -v -d src=movies -d db=pluralsight -e 'select * from
      ${hivevar:db}.${hivevar:src} LIMIT 100;'
```

Summary

- **Data Types**
  - Primitive and Complex
- **Table Partitioning**
  - Managed tables by loading data
  - Alter Table for External tables
  - Dynamic partition inserts
- **Multi Inserts**
- **Functions**
- **Order By, Sort By, Distribute By, Cluster By**
- **The Hive CLI**