Transforms, UDFs and variables

User Defined Functions

Transforms

written in Python, Perl, Ruby

Using TRANSFORM to Process Data

- **You are not limited to manipulating data exclusively in HiveQL**
  - Hive allows you to transform data through external scripts or programs
  - These can be written in nearly any language
- **This is done with HiveQL's `TRANSFORM . . . USING` construct**
  - One or more fields are supplied as arguments to `TRANSFORM()`
  - The external script is identified by `USING`
    - It receives each record, processes it, and returns the result
- **Use `ADD FILE` to distribute the script to nodes in the cluster**

```
hive> ADD FILE myscript.pl;
hive> SELECT TRANSFORM(*) USING 'myscript.pl'
      FROM employees;
```

Data Input and Output with TRANSFORM

- **Your external program will receive one record per line on standard input**
  - Each field in the supplied record will be a tab-separated string
  - `NULL` values are converted to the literal string `\N`
- **You may need to convert values to appropriate types within your program**
  - For example, converting to numeric types for calculations
- **Your program must return tab-delimited fields on standard output**
  - Output fields can optionally be named and cast using the syntax below

```
SELECT TRANSFORM(product_name, price)
       USING 'tax_calculator.py'
       AS (item_name STRING, tax INT)
       FROM products;
```

Hive TRANSFORM Example (1)

- **Here is a complete example of using `TRANSFORM` in Hive**
  - Our Perl script parses an e-mail address, determines to which country it corresponds, and then returns an appropriate greeting
  - Here's a sample of the input data

```
hive> SELECT name, email_address FROM employees;
Antoine   antoine@example.fr
Kai       kai@example.de
Pedro     pedro@example.mx
```

  - Here's the corresponding HiveQL code

```
hive> ADD FILE greeting.pl;
hive> SELECT TRANSFORM(name, email_address)
          USING 'greeting.pl' AS greeting
          FROM employees;
```

Hive TRANSFORM Example (2)

```
#!/usr/bin/env perl

%greetings = ('de' => 'Hallo',
              'fr' => 'Bonjour',
              'mx' => 'Hola');

while (<STDIN>) {
    ($name, $email) = split /\t/;
    ($suffix) = $email =~ /\.([a-z]+)/;
    $greeting = $greetings{$suffix};
    $greeting = 'Hello' unless defined($greeting);
    print "$greeting $name\n";
}
```

Create greeting array: key is country code

Parse to find country code

Construct the greeting

Hive TRANSFORM Example(3)

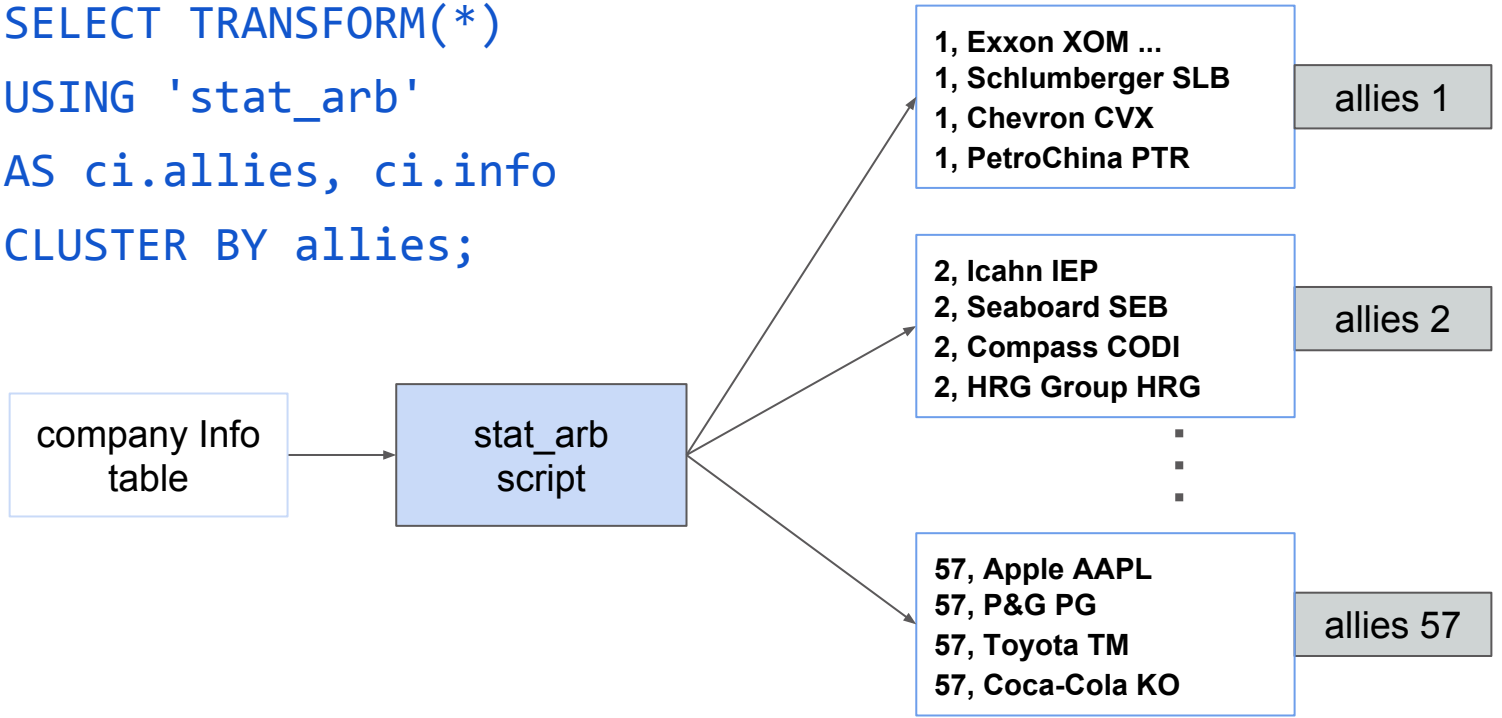
- Finally, here's the result of our transformation

```
hive> ADD FILE greeting.pl;
hive> SELECT TRANSFORM(name, email_address)
          USING 'greeting.pl' AS greeting
          FROM employees;

Bonjour Antoine
Hallo Kai
Hola Pedro
```

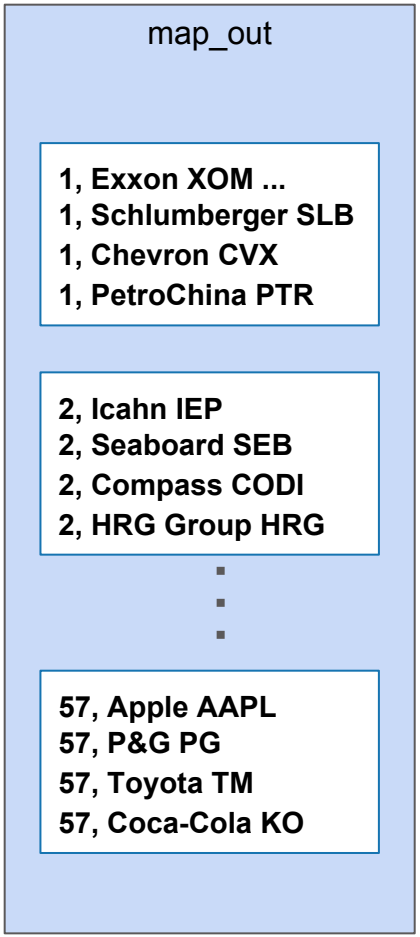
Advanced transforms (1)

```
FROM companyInfo ci
  SELECT TRANSFORM(*)
  USING 'stat_arb'
  AS ci.allies, ci.info
  CLUSTER BY allies;
```



Advanced transforms (2)

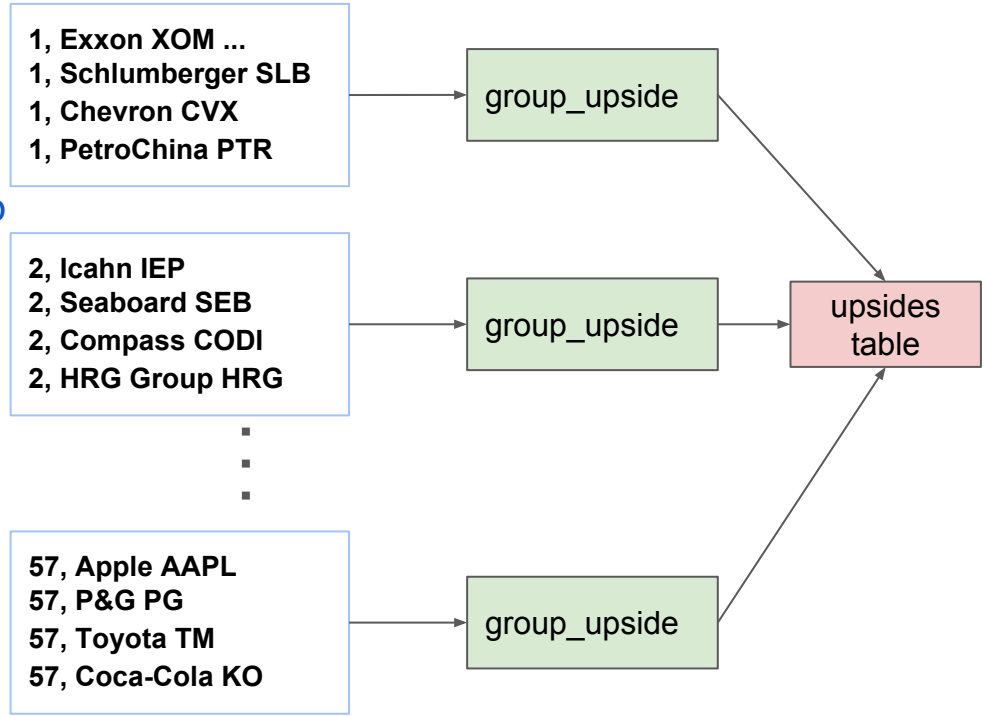
```
FROM (
  FROM companyInfo ci
  SELECT TRANSFORM(*)
  USING 'stat_arb' AS allies, info
  CLUSTER BY allies
) map_out
```



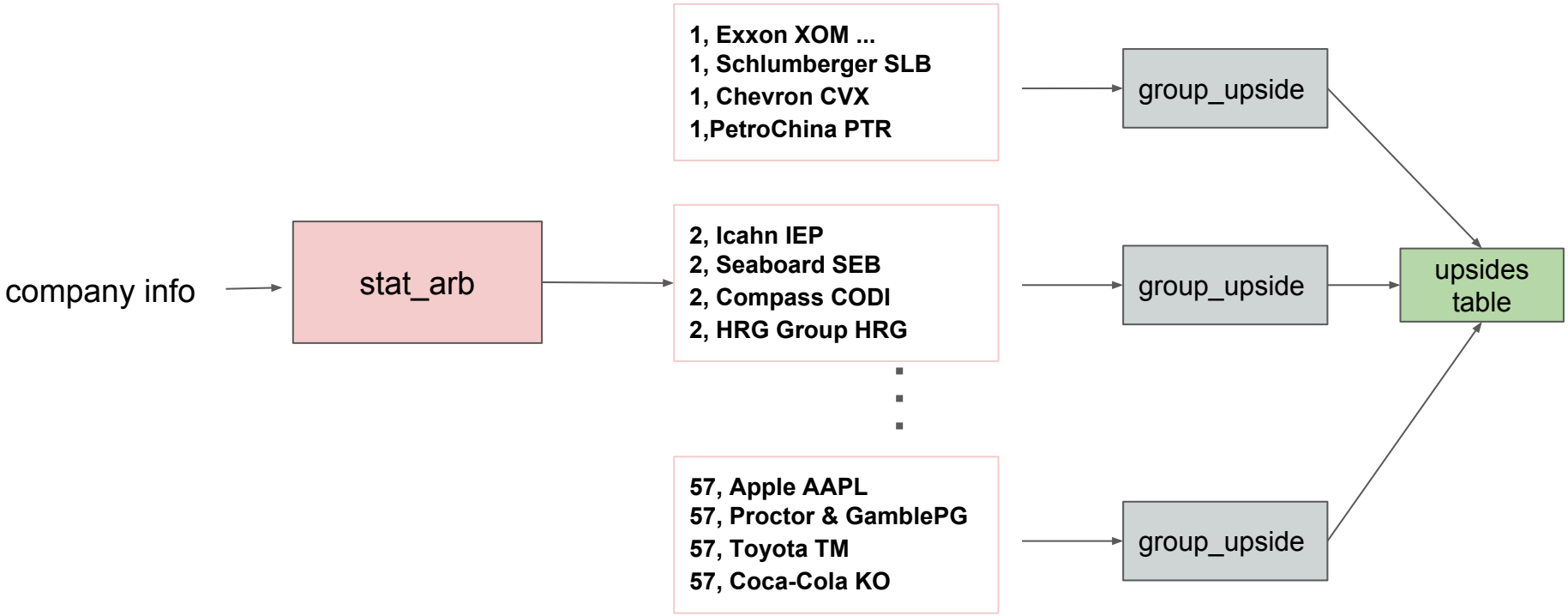
Advanced transforms (3)

```
FROM (
  FROM companyInfo ci
  SELECT TRANSFORM(*)
  USING 'stat_arb' AS allies, info
  CLUSTER BY allies) map_out

INSERT OVERWRITE TABLE upsides
SELECT TRANSFORM(
   map_out.allies, map_out.info)
USING 'group_upside'
AS a_group, upside;
```



Processing transforms stat_arb and group_upside



http://www.kellogg.northwestern.edu/faculty/papanikolaou/htm/finc460/ln/lecture6.pdf

Transforms

Used to execute scripts

- can take as input multiple columns (or all columns) in a table
- scripts can be python, ruby, perl, bash -- anything that reads stdin and writes to stdout.

Familiar?

UDFs

written in Java, JRuby, Clojure, Groovy…
(anything that runs on a JVM)

UDFs - overview

code create and compile DataFormatUDF.java
create date-format-udf.jar

add jar

```
hive> ADD JAR date-format-udf.jar;
```

register

```
hive> CREATE TEMPORARY FUNCTION DATE_FORMAT
      AS 'com.nexr.platform.hive.udf.UDFDateFormat';
```

use

```
hive> SELECT order_date FROM orders LIMIT 1;
2011-12-06 10:03:35

hive> SELECT DATE_FORMAT(order_date, 'dd-MMM-yyyy')
      FROM orders LIMIT 1;
06-Dec-2011

hive> SELECT DATE_FORMAT(order_date, 'dd/mm/yy')
      FROM orders LIMIT 1;
06/12/11

hive> SELECT DATE_FORMAT(order_date, 'EEEE, MMM d, yyyy')
      FROM orders LIMIT 1;
Tuesday, Dec 6, 2011
```

Types of functions

UDFs

- one or more values input, single value output

```
hive> SELECT CALC_SHIPPING_COST(order_id, 'OVERNIGHT')
      FROM orders WHERE order_id = 5742354;
```

UDAFs (aggregate functions) - multiple row input, single value output

- **Example: SUM, MAX**

UDTFs (table functions) - single row input, multiple row output

- **Example: EXPLODE**

How to (1): Write the UDF

Extend UDF class Implement "evaluate" method

```
package edu.ucsc.hadoop.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class DomainUdf extends UDF {
        public Text evaluate (final Text s) {
                if (s == null) { return null; }
                String str = s.toString();
                int lastDot = str.lastIndexOf(".");
                int penultimateDot = str.lastIndexOf(".", lastDot);
                int startIndex = (penultimateDot == -1) ? 0: penultimateDot;
                return new Text(str.substring(startIndex, penultimateDot));
        }
}
```

How to(2): Compile code and create jar

```
hive > !java
javac -classpath .:/usr/lib/hadoop/hadoop-common.jar:\
/usr/lib/hadoop/hadoop-common-2.0.0-cdh4.2.1.jar:\
/usr/lib/hive-common-0.10-cdh4.2.1.jar:\
/usr/lib/hive/lib/hive-exec-0.10.0-cdh4.2.1 \
DomainUdf.java

hive > !jar
jar -cf email_domain.jar /edu/ucsc/hadoop/udf/DomainUdf.class
```

jars to include in the classpath

How to(2): Add jar and register the UDF

- Add and register the UDF for the current session only
  - places the jar file in the distributed cache

```
hive> add email_domain.jar;
        Added email_domain.jar to classpath
        Added resource:  email_domain.jar
hive> CREATE TEMPORARY FUNCTION email_domain
        > AS 'edu.ucsc.hadoop.udf.DomainUdf';
        OK
```

- To persist and make available for future sessions
  - Add the UDF to the .hiverc file or use a script to invoke it

How to(3): Create table schema and add data

```
hive > CREATE TABLE IF NOT EXISTS subscriber (
       > username STRING,
       > dept STRING,
       > email STRING,
       > trained STRING)
       > ROW FORMAT DELIMITED FIELDS TERMINATED BY '.';
OK
hive > LOAD DATA LOCAL INPATH 'subscribers.txt'
       > INTO TABLE subscriber;
```

how-to (4): Invoke the function

```
hive> SELECT email_domain(email) from subscribers;
Total MapReduce jobs = 1 …
```

Using Maven to create UDFs

- Create a Maven project in Eclipse
  - **Use this pom.xml file**
  - **Change it to match your project and package names**
- Write the UDF in eclipse
- Build the jar with Maven:
  - **right-click on the project > Run As > Maven install**
  - **rename the jar to a meaningful name (my_amazing_udf)**
- Startup hive (you can't do this part in eclipse)

  hive> add my_amazing_udf.jar;
  hive> CREATE TEMPORARY FUNCTION my_amazing_udf
  > AS 'edu.ucsc.hadoop.udf.WhateverUdf';
- Now you can invoke my_amazing_udf in Hive

Summary

Create a Java class that extends UDF and implements evaluate(...)

Compile and jar the file using the necessary HBASE classes

start hive

```
hive> add jar myInfo.jar

hive> CREATE TEMPORARY FUNCTION my_info
    > AS 'com.example.hive.udf.MyInfoUDF';

hive> SELECT my_info(name) from importantTable;
```

-- make sure importantTable exists.

UDAFs: Unholy Difficult Aggravating Functions

- How writing the code for a moving average calculation can blow your mind: https://blogs.oracle.com/datawarehousing/entry/three_little_hive_udfs_part2
- To find averaging completely incomprehensible: https://ragrawal.wordpress.com/2013/10/26/writing-hive-custom-aggregate-functions-udaf-part-ii/
- To witness code that is mysterious and without explanation: http://www.ericlin.me/hive-user-defined-aggregation-function-udaf

The valuable tutorial on UDFs, UDAFs and UDTFs:

http://blog.matthewrathbone.com/2013/08/10/guide-to-writing-hive-udfs.html

- It is nontrivial.
- Great coverage of UDFs and UDTFs.
- And when you are done with the UDAF section, you will say:
  - “All that for letter count?”

Setting variables in Hive

Set variables within Hive

- Within the Hive shell, set a named variable equal to some value:

```
hive> SET state=CA;
```

- To use the variable's value in a HiveQL query:

```
hive> SELECT * FROM employees
      WHERE STATE = '${hiveconf:state}';
```

Set variables from the command line

- **Create a simple query called state.hql that contains:**

```
SELECT COUNT(DISTINCT emp_id) FROM employees
        WHERE state = '${hiveconf:state}';
```

- **Run it from the commandline with different states**

```
$ hive -hiveconf state=CA -f state.hql > ca_count.txt
$ hive -hiveconf state=NY -f state.hql > ny_count.txt
$ hive -hiveconf state=TX -f state.hql > tx_count.txt
```

Essential points

- **SerDes govern how Hive reads and writes a table's records**
  - Specified (or defaulted) when creating a table
- **`TRANSFORM` processes records using an external program**
  - This can be written in nearly any language
- **UDFs are User-Defined Functions**
  - Custom logic that can be invoked just like built-in functions
- **Hive substitutes variable placeholders with literal values you assign**
  - This is done when you execute the query
  - Especially helpful with repetitive queries