Apache Pig

Declarative vs Functional Pgmming

Welcome to the Zoo!

What is Pig?

Pig in the Hadoop Ecosystem

- Pig is a high-level language and platform for analyzing large datasets.
- **Core Concept:** Pig Latin - a language designed for expressing data transformations.
- Pig translates **Pig Latin scripts** into **MapReduce jobs** for execution on Hadoop.

Pig provides a procedural way to wrangle and analyze Big Data, specifying step-by-step instructions on how to process it, in contrast to the declarative approach of SQL.

Pig provides a **procedural** way to wrangle and analyze Big Data, specifying step-by-step instructions on how to process it, **in contrast to the declarative** approach of SQL.

The approach we took for Hive

Programming Paradigms: Declarative vs. Procedural

What is Declarative Programming?

Focus: "What" you Need

- **Focus**: Specifying the desired result, not the exact steps.
- **Implementation**: The system determines the best way to achieve it.
- **Examples:**
  - SQL queries
  - Spreadsheets with formulas

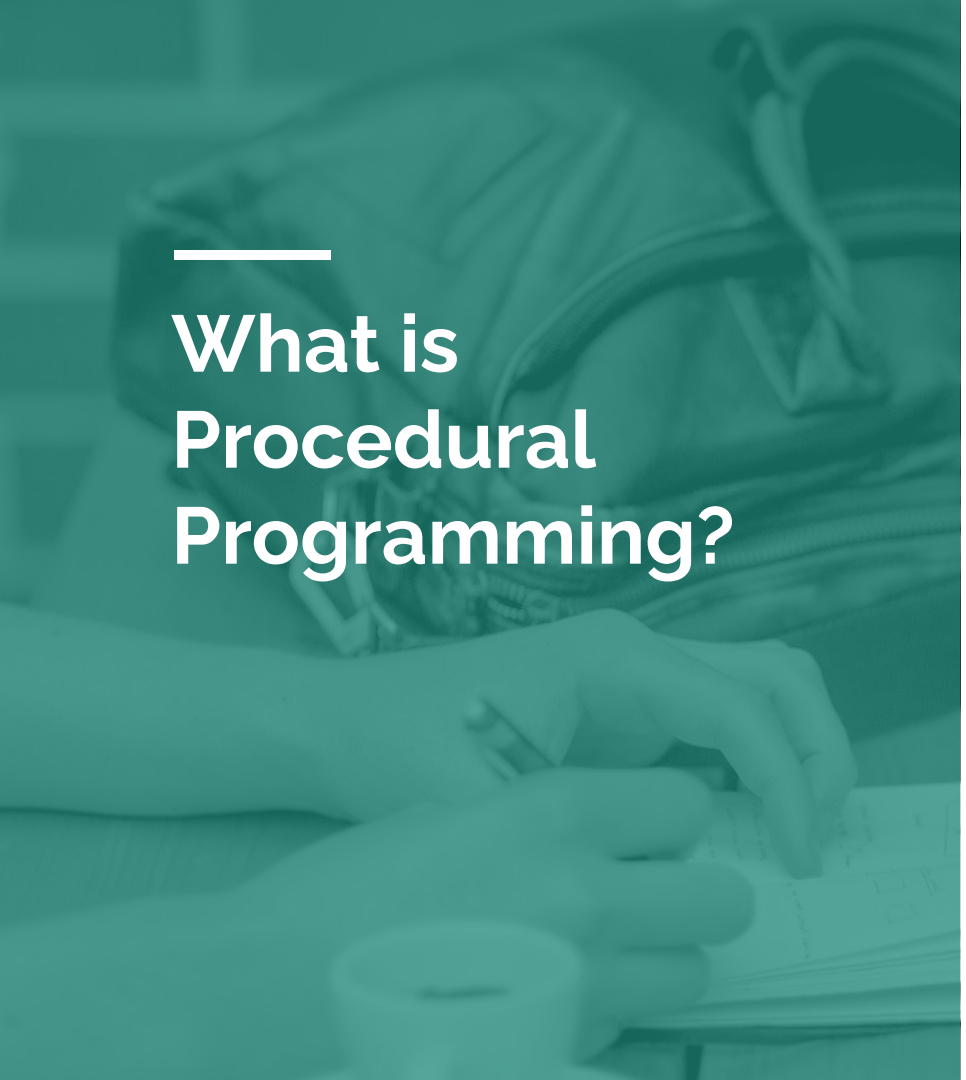
Focus: "What" you Need

SQL queries:

SELECT * FROM customers WHERE country = 'USA';

(Focuses on desired data)

- **Focus**: Specifying the desired result, not the exact steps.
- **Implementation**: The system determines the best way to achieve it.
- **Examples:**
  - SQL queries
  - Spreadsheets with formulas



What is Procedural Programming?

Focus: "How" to Do It

- **Focus**: Providing step-by-step instructions on what to do.
- **Implementation**: A sequence of commands the computer follows.
- **Examples:**
  - Traditional languages (Python, Java, etc.)

Key Differences

Declarative	Procedural
Focus: "What"	Focus: “How
Flexibility: System optimizes	Flexibility: You dictate the flow
Readability: Easier for non-programmers	Readability: Requires coding understanding

Analogy

Planning a Trip

- **Declarative**: "I want a flight to Paris and a hotel near the Eiffel Tower."

- **Procedural**:
  - Search flight websites.
  - Compare prices.
  - Book the cheapest.
  - Do the same for hotels in the area, checking reviews..."

Analogy

Planning a Trip

- **Declarative**: "I want a flight to Paris and a hotel near the Eiffel Tower."

  Hive

- **Procedural**:
  - Search flight websites.
  - Compare prices.
  - Book the cheapest.
  - Do the same for hotels in the area, checking reviews..."

  Pig

Pig - Latin in Action

Analyzing Web Logs with Pig

Successful connection

```
123.45.67.89 - - [2023-12-21 15:37:23] "GET /product/123 HTTP/1.1" 200
201.85.12.55 - - [2023-12-21 16:51:02] "POST /user/login HTTP/1.1" 401
... more lines ...
```

Analyzing Web Logs with Pig

Find number of visits per country.

Successful connection

123.45.67.89 - - [2023-12-21 15:37:23] "GET /product/123 HTTP/1.1" 200
201.85.12.55 - - [2023-12-21 16:51:02] "POST /user/login HTTP/1.1" 401
... more lines ...

```
weblogs = LOAD 'webtraffic.log' AS (ip, timestamp, url, status_code);
filtered_logs = FILTER weblogs BY status_code == '200';
visits_per_country = GROUP filtered_logs BY country;
visit_counts = FOREACH visits_per_country GENERATE group, COUNT(filtered_logs);
STORE visit_counts INTO 'results';
```

Analyzing Web Logs with Pig

Find number of visits per country.

Successful connection

123.45.67.89 - - [2023-12-21 15:37:23] "GET /product/123 HTTP/1.1" 200
201.85.12.55 - - [2023-12-21 16:51:02] "POST /user/login HTTP/1.1" 401
... more lines ...

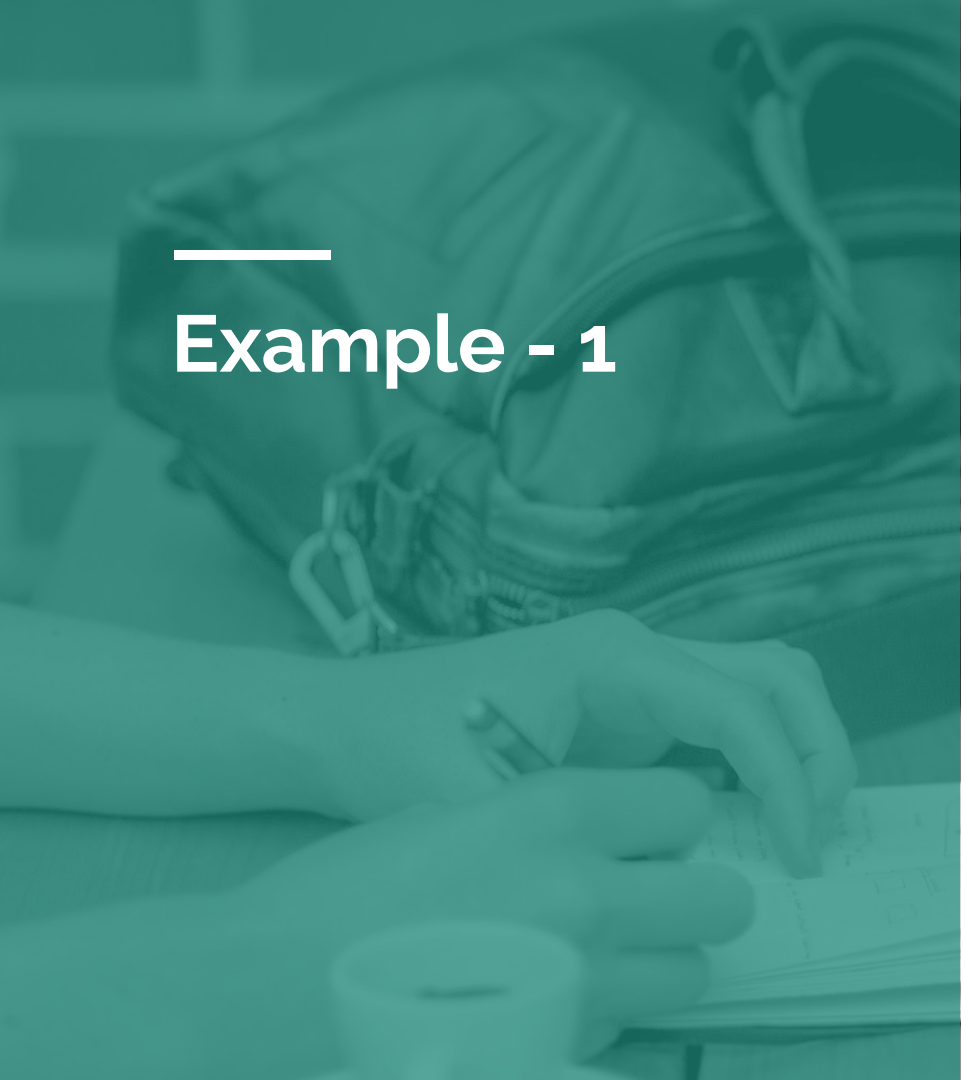
```
weblogs = LOAD 'webtraffic.log' AS (ip, timestamp, url, status_code);
filtered_logs = FILTER weblogs BY status_code == '200';
visits_per_country = GROUP filtered_logs BY country;
visit_counts = FOREACH visits_per_country GENERATE group, COUNT(filtered_logs);
STORE visit_counts INTO 'results';
```

Building Blocks of Pig Latin

- **LOAD**: Brings data from external sources into your Pig script.
  - Think of it as a temporary table.
- **FILTER**: Selectively removing rows that don't meet your criteria.
- **FOREACH**: Iterate through groups of data and perform transformations on each group
.
- **STORE**: Save it back to a storage system.

Building Blocks of Pig Latin

- **GROUP BY**: Groups rows based on a specified column or expression.
- **ORDER BY**: Sorts data in ascending or descending order based on a column or expression.
- **LIMIT**: Restricts the number of rows returned in a result set.



Example - 1

Find Top URLs

Task: Modify the script to find the top 10 most visited URLs, regardless of country.

Find number of visits per country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_country = GROUP
filtered_logs BY country;

visit_counts = FOREACH
visits_per_country GENERATE group,
COUNT(filtered_logs);

STORE visit_counts INTO 'results';
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);
```

Find number of visits per country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_country = GROUP
filtered_logs BY country;

visit_counts = FOREACH
visits_per_country GENERATE group,
COUNT(filtered_logs);

STORE visit_counts INTO 'results';
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';
```

Find number of visits per country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_country = GROUP
filtered_logs BY country;

visit_counts = FOREACH
visits_per_country GENERATE group,
COUNT(filtered_logs);

STORE visit_counts INTO 'results';
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_url = GROUP filtered_logs
BY url;
```

Find number of visits per country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_country = GROUP
filtered_logs BY country;

visit_counts = FOREACH
visits_per_country GENERATE group,
COUNT(filtered_logs);

STORE visit_counts INTO 'results';
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_url = GROUP filtered_logs
BY url;

visit_counts = FOREACH visits_per_url
GENERATE group, COUNT(filtered_logs);
```

Find number of visits per country.

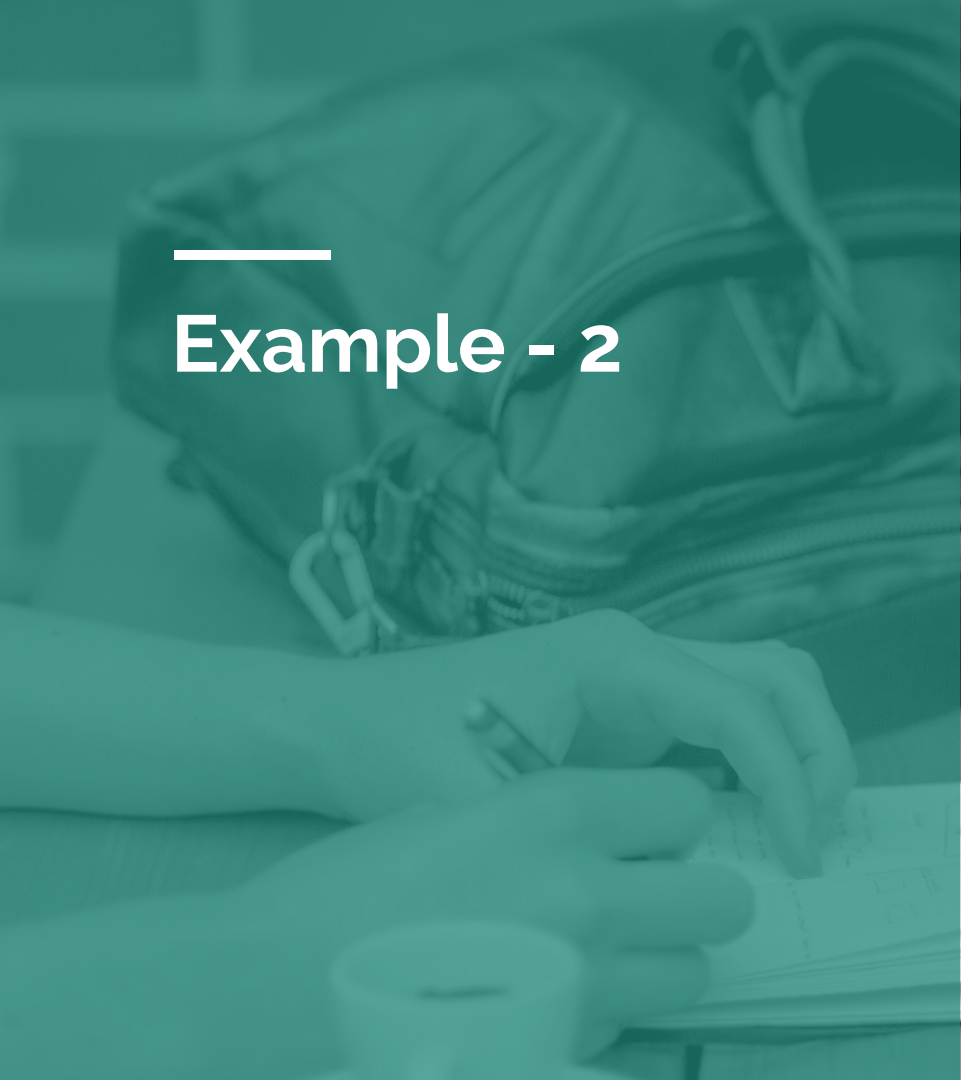
```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_country = GROUP
filtered_logs BY country;

visit_counts = FOREACH
visits_per_country GENERATE group,
COUNT(filtered_logs);

STORE visit_counts INTO 'results';
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_url = GROUP filtered_logs
BY url;

visit_counts = FOREACH visits_per_url
GENERATE group, COUNT(filtered_logs);

ordered_visits = ORDER visits_per_url
BY visit_counts DESC;
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_url = GROUP filtered_logs
BY url;

visit_counts = FOREACH visits_per_url
GENERATE group, COUNT(filtered_logs);

ordered_visits = ORDER visits_per_url
BY visit_counts DESC;

top_10_urls = LIMIT ordered_visits 10;
STORE top_10_urls INTO
'top_urls_results';
```



Example - 2

Dealing with Multiple Files

Imagine that you have -

customer data in one file (customers.txt) &

order data in another file (orders.txt).

You want to find out which customers placed orders and how many orders each placed.

Imagine that you have -

customer data in one file **(customers.txt)** &

order data in another file **(orders.txt)**.

You want to find out which customers placed orders and how many orders each placed.

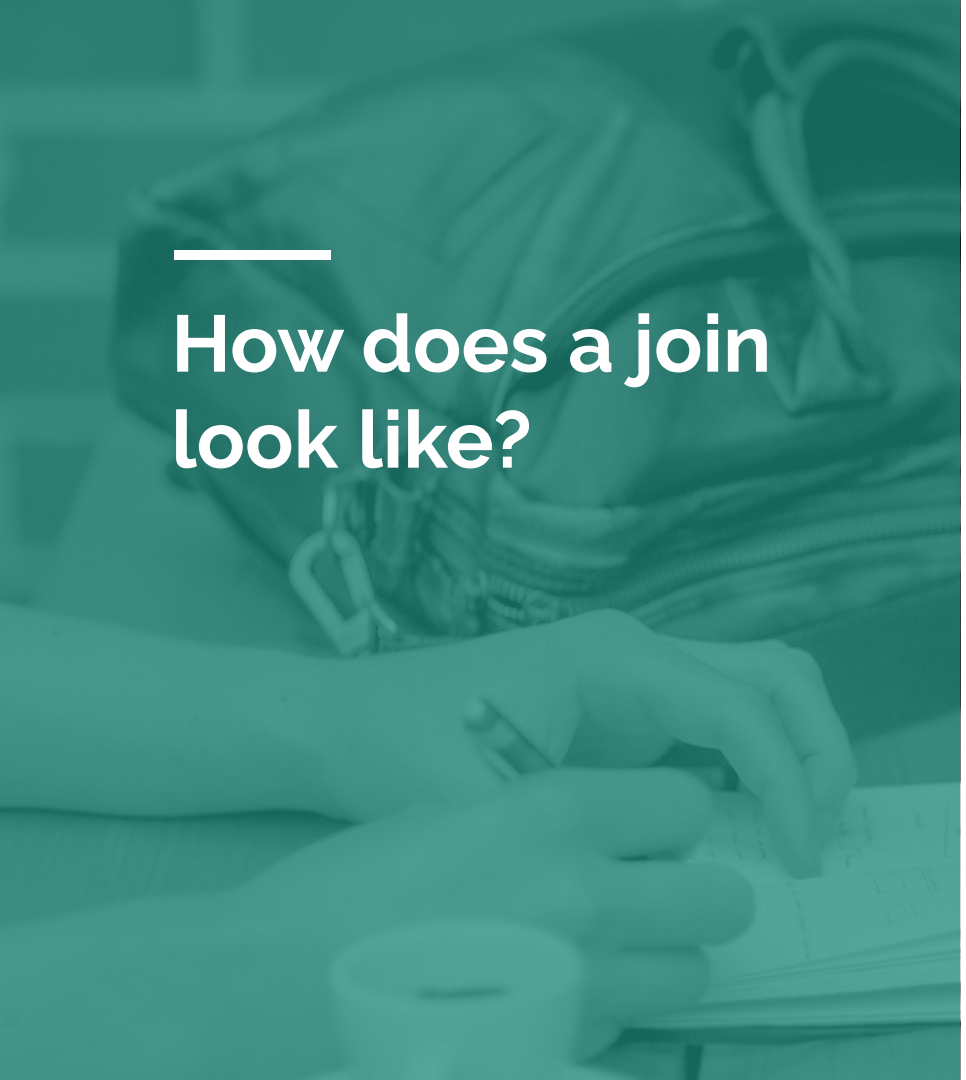
```
Data Files:

customers.txt: (customer_id, name, email)

orders.txt: (order_id, customer_id, product_id, quantity)
```

You want to find out which customers placed orders and how many orders each placed.

customers.txt: (customer_id, name, email)

orders.txt: (order_id, customer_id, product_id, quantity)

```
-- Step 1: Load the data files

customers = LOAD 'customers.txt' AS
(customer_id: int, name: chararray, email:
chararray);
orders = LOAD 'orders.txt' AS (order_id:
int, customer_id: int, product_id: int,
quantity: int);

 -- Step 2: Join customer and order data on
 customer_id
 joined_data = JOIN customers BY
 customer_id, orders BY customer_id;

 -- Step 3: Group by customer and count
 orders
 customer_orders = FOREACH (GROUP
 joined_data BY customers.customer_id)
 GENERATE group, COUNT(orders.order_id) AS
 order_count;

 -- Step 4: Store the results
```



How does a join look like?

Dealing with Multiple Files

Sample `customers.txt: (customer_id, name, email)`

```
1, Alice Smith, alice@email.com
2, Bob Johnson, bjohnson@email.com
3, Charlie Parker, charlie@email.com
4, Sarah Davis, sarah.d@email.com
```

Sample `orders.txt: (order_id, customer_id, product_id, quantity)`

```
101, 1, P123, 2
102, 3, P456, 1
103, 2, P123, 3
104, 1, P789, 1
105, 3, P123, 1
```

Sample `customers.txt: (customer_id, name, email)`

```
1, Alice Smith, alice@email.com
2, Bob Johnson, bjohnson@email.com
3, Charlie Parker, charlie@email.com
4, Sarah Davis, sarah.d@email.com
```

Sample `orders.txt: (order_id, customer_id, product_id, quantity)`

```
101, 1, P123, 2
102, 3, P456, 1
103, 2, P123, 3
104, 1, P789, 1
105, 3, P123, 1
```

Join Output

```
(1, Alice Smith, alice@email.com, 101, 1, P123, 2)
(1, Alice Smith, alice@email.com, 104, 1, P789, 1)
(2, Bob Johnson, bjohnson@email.com, 103, 2, P123, 3)
(3, Charlie Parker, charlie@email.com, 102, 3, P456, 1)
(3, Charlie Parker, charlie@email.com, 105, 3, P123, 1)
```

Top 10 most visited URLs, regardless of country.

```
weblogs = LOAD 'webtraffic.log' AS
(ip, timestamp, url, status_code);

filtered_logs = FILTER weblogs BY
status_code == '200';

visits_per_url = GROUP filtered_logs
BY url;

visit_counts = FOREACH visits_per_url
GENERATE group, COUNT(filtered_logs);

ordered_visits = ORDER visits_per_url
BY visit_counts DESC;

top_10_urls = LIMIT ordered_visits 10;
STORE top_10_urls INTO
'top_urls_results';
```

User Defined Functions in Pig

Extending Pig Latin with UDFs

Why UDFs:

- When Pig Latin's built-in operations aren't enough, write functions in Java, Python, etc.
- Perform custom calculations, string manipulations, or date formatting.

User Defined Functions in Pig

How to use UDFs

The 'REGISTER' Command:

Tells Pig the location of your compiled UDF code (usually a JAR file).

Example: REGISTER '/path/to/my_udfs.jar';

Using in Scripts: UDFs are called just like built-in Pig operators.

Example: capitalized_data = FOREACH data GENERATE ToUpper(name);

UDF!

UDF Example in Pig

User Defined Functions in Pig

How to use UDFs

Problem
Your input data has timestamps like "2023_12_20-15:34:51" (not directly usable by Pig's date functions).

```
123.45.67.89 - - [2023-12-21-15:37:23] "GET /product/123 HTTP/1.1" 200
201.85.12.55 - - [2023-12-21-16:51:02] "POST /user/login HTTP/1.1" 401
... more lines ...
```

Goal
Convert these to standard ISO format like "2023-12-20T15:34:51".

Using UDFs

Python script for format conversion -

```
from datetime import datetime

def parse_timestamp(raw_timestamp):
  # Assuming the input format is always  "YYYY_MM_DD-HH:MM:SS"
  dt_obj = datetime.strptime(raw_timestamp, '%Y_%m_%d-%H:%M:%S')
  return dt_obj.isoformat()  # Returns the standard ISO format
```

Saved as `timestamp_parser.py`

Using UDFs

Pig script for format conversion -

```
-- Assuming your UDF code is compiled into 'udf.jar' in your working directory
REGISTER 'udf.jar';
DEFINE parse_timestamp timestamp_parser.parse_timestamp();

-- Your raw data might be loaded from HDFS
raw_data = LOAD 'input.csv' USING PigStorage(',') AS (field1, weird_timestamp,
field3);

-- Data transformation using the UDF
clean_data = FOREACH raw_data GENERATE
             field1,
             parse_timestamp(weird_timestamp) AS clean_timestamp,
             field3;

-- Store the result with standardized timestamps
STORE clean_data INTO 'output' USING PigStorage(',');
```

Using UDFs

Pig script for format conversion -

```
-- Assuming your UDF co                                                    ctory
REGISTER 'udf.jar';
DEFINE parse_timest

-- Your raw data mi
raw_data = LOAD 'in                                                        o,
field3);

-- Data transformat
clean_data = FOREAC
             field1
             parse_
             field3

-- Store the result wi
STORE clean_data INTO 'output' USING PigStorage(',');
```

Using functions in Pig Latin is possible.
How about the other way around?

Using Pig script in Python code

Using Pig script in Python code

Prerequisites:

- Pig Installed: You'll need Pig set up in your environment.
- Python Library: Install the `pyglib` library (pip install pyglib)

Using Pig script in Python code -

Calculating word frequencies from a text file.

```
from pyglib import pigscript

pigscript.register_code("""
  data = LOAD 'input.txt' AS (line:chararray);
  words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS word;
  grouped = GROUP words BY word;
  wordcount = FOREACH grouped GENERATE group, COUNT(words);
  STORE wordcount INTO 'word_frequency_out';
""")

pigscript.run_query()
```

Using Pig script in Python code -

Calculating word frequencies from a text file.

```
from pyglib import pigscript

pigscript.register_code("""
  data = LOAD 'input.txt' AS (line:chararray);
  words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS word;
  grouped = GROUP words BY word;
  wordcount = FOREACH grouped GENERATE group, COUNT(words);
  STORE wordcount INTO 'word_frequency_out';
""")

pigscript.run_query()
```

Output

Why is it “Pig” - the name?

Metaphor for its ability to "eat" any kind of data.