CARLSON SCHOOL
OF MANAGEMENT
UNIVERSITY OF MINNESOTA

# Text Analytics with Hive

MSBA 6330 Prof Liu

# Text Analytics With Hive

- In this chapter, you will learn
  - How to use Hive's string functions
  - How to use regular expressions in Hive
  - How to load text files that do not use consistent delimiters
  - How to obtain n-grams as part of sentiment analysis
  - How to estimate how often words or phrases occur in text

# Hive Text Analytics Overview

- What types of (text) data are we producing today?
  - Unstructured text data
  - Semi-structured data in formats like JSON
  - Log files
- Examples of unstructured and semi-structured data:
  - Free-form notes in electronic medical records
  - Electronic messages
  - Product reviews
- We discuss two kinds of text processing in Hive
  - Loading text file formats correctly
  - Dealing with text after it is loaded

Text Analytics with Hive

# STRING FUNCTIONS IN HIVE

# Basic String Functions

- Hive supports many string functions often found in RDBMSs

| Name | Description | Example | Input | Output |
|---|---|---|---|---|
| `UPPER` | Return input converted to upper case | `UPPER(name)` | `Bob` | `BOB` |
| `LOWER` | Return input converted to lower case | `LOWER(name)` | `Bob` | `bob` |
| `TRIM` | Return input without leading or trailing spaces | `TRIM(name)` | `_Bob_` | `Bob` |
| `REPLACE` | Replaces the occurrences of OLD string with NEW | `REPLACE(name, "B","P")` | `Bob` | `Pob` |
| `RTRIM (LTRIM)` | Return input without trailing spaces | `RTRIM(name)` | `_Bob_` | `_Bob` |
| `SUBSTR*` | Return a portion of input | `SUBSTR(name,0,3)` | `Samuel` | `Sam` |
| `INSTR` | Return position of substring | `INSTR(name,'mu')` | `Samuel` | `3` |

* Unlike SQL, starting position is zero-based in Hive

# Numeric Format Functions

- Hive offers two functions for formatting a number
  - Simple: `FORMAT_NUMBER` (0.10.0 and later)
  - Versatile: `PRINTF` (0.9.0 and later)

| Function | Example | Input | Output |
|---|---|---|---|
| FORMAT_NUMBER | FORMAT_NUMBER(commission, 2) | 2345.519728 | 2,345.52 |
| PRINTF | PRINTF("$%1.2f", total_price) | 356.9752 | 356.98 |
| PRINTF | PRINTF("%s owes $%1.2f", name, amt) | Bob, 3.9 | Bob owes $3.90 |
| PRINTF | PRINTF("%.2f%%", taxrate * 100) | 0.47314 | 47.3% |

# Splitting and Combining Strings

- `CONCAT` combines one or more strings
  - The `CONCAT_WS` variation joins them WITH SEPARATOR
- `SPLIT` does nearly the opposite
  - Difference: Return value is `ARRAY<STRING>`

| Example | Output |
|---|---|
| `CONCAT('alice', '@example.com')` | `alice@example.com` |
| `CONCAT_WS(' ', 'Bob', 'Smith')` | `Bob Smith` |
| `CONCAT_WS('/', 'Amy', 'Sam', 'Ted')` | `Amy/Sam/Ted` |
| `SPLIT('Amy/Sam/Ted', '/')` | `["Amy" , "Sam" , "Ted" ]` * |
| `LEFT('Hello',2),RIGHT('Hello',3)` | `'He', 'llo'` |

* The result is a complex field of ARRAY<STRING>

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-StringFunctions

# Converting An Array To Records With EXPLODE

- The `EXPLODE` function creates a record for each element in an array
  - An example of a table generating function
  - The alias is required when invoking table generating functions (like SELECT)

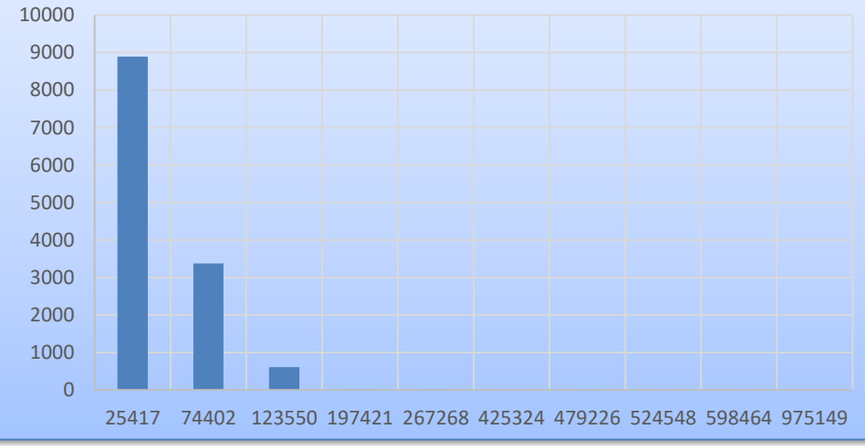
```
hive> SELECT people FROM example;
Amy,Sam,Ted        -- String
hive> SELECT SPLIT(people, ',') FROM example;
["Amy", "Sam", "Ted"]   -- Array <STRING>
hive> SELECT EXPLODE(SPLIT(people, ',')) AS names FROM example;
Amy   -- 3 Strings
Sam
Ted
```

  - In this example, there is only one row of data; and the input for `EXPLODE` is an array of strings.
  - The input for `EXPLODE` can also be a map, in which case each row of `names` would consist 2 fields, a key and its value

# Calculating Data for Histograms

- `histogram_numeric` creates data needed for histograms
  - Input: column name and number of “bins” in the histogram
  - Output: coordinates representing bin centers and heights

```
> SELECT explode(histogram_numeric(total_price, 10)) FROM cart_orders;
{"x":25417.336745023003,"y":8891.0}
{"x":74401.5041469194,"y":3376.0}
{"x":123550.04418985262,"y":611.0}
{"x":197421.12500000006,"y":24.0}
{"x":267267.53846153844,"y":26.0}
{"x":425324.0,"y":4.0}
{"x":479226.38461538474,"y":13.0}
{"x":524548.0,"y":6.0}
{"x":598463.5,"y":2.0}
{"x":975149.0,"y":2.0}
```




- Import this data into charting software to produce a histogram
  - Excel, Gnuplot, matlab, Mathematica etc.

Text Analytics with Hive

# REGULAR EXPRESSIONS IN HIVE

# Regular Expressions

- A regular expression (regex) matches a pattern in text

| Regular Expression | Matches | Matching Pattern (In Blue) |
|---|---|---|
| `Dualcore` | literal text “Dualcore” | I wish Dualcore had 2 stores in 90210. |
| `\\d` | a single digit | I wish Dualcore had 2 stores in 90210. |
| `\\d{5}` | exactly five digits | I wish Dualcore had 2 stores in 90210. |
| `\\d\\s\\w+` | single digit followed by a whitespace and 1+ word characters | I wish Dualcore had 2 stores in 90210. |
| `\\w{5,9}` | 5 to 9 word characters | I wish Dualcore had 2 stores in 90210. |
| `.?\\.` | Any character followed by a dot (“.”) | I wish Dualcore had 2 stores in 90210. |
| `.*\\.` | Any length of any character followed by a dot (“.”) | I wish Dualcore had 2 stores in 90210. |

**Character class:**
`\d`: digit
`\s`: whitespace including space, `\t`, `\r`, `\n`
`\w`: word characters including `a-z, A-Z, 0-9, _`
`.`: matches everything.

**Repetition**:
`?`: 0-1
`+`: 1-n,
`*`: 0-n,
`{3}`: exactly three times
`{3,5}`: 3-5 times.

# Regular Expression

- Alternatives:

  `(cat|dog)`: matches “cat” or “dog”

  `[ab]`: “`a`” or “`b`”

  `[a-z]`: “`a`” to “`z`”,

  `[,\t]`: “`,`” or `tab`.

  `[^,]`: any character but `","` (`^` is “negate” when used within `[]`)

- Capture – if you hope to return part of matched text.

  `"([^"]*)"`: captures what is the parenthesis, in this case, the content between two quotation marks.

- In Hive, double escaping `\\` is needed because Hive interpreter and regular expression interpreter both handle escaping.
  - E.g. “`\\d`” becomes “`\d`” after Hive interpretation, which is then interpreted by a regex interpreter.

Read More: http://www.regular-expressions.info/charclassintersect.html

# Hive's Regular Expression Functions

- Hive has three important functions that use regular expressions
  - **`REGEXP`**: `"text REGEXP pattern"`: whether `text` matches with the `pattern`
  - **`REGEXP_EXTRACT(text,pattern,n)`**: returns the string extracted using the `pattern` (`n` for returning n-th captured group)
  - **`REGEXP_REPLACE`**`(text,pattern,replacement)`: returns the string resulting from replacing all substrings in `text` that matches the `pattern` with the `replacement`
- Examples (`txt`): `It's on Staple St or Maple St. in 90210,12345`
  - Find the 1st instance of 5 digits in `txt`

```
SELECT REGEXP_EXTRACT(txt, '(\\d{5}).(\\d{5})', n);
```

n:
1 - First group: `90210`
2 - Second group: `12345`
0 – the entire pattern: `90210,12345`

  - Replace all occurrences of street abbrevations with "`Street `" in `txt`

```
SELECT REGEXP_REPLACE(txt, 'St\\.?\\s+', 'Street ');
It's on Staple Street or Maple Street in 90210,12345
```

St (+ . or not) (+ space or not)

Matches:
St.[space]
St[space]
St[\tab]
But not: Staple

Text Analytics with Hive

# PROCESSING TEXT DATA WITH SERDES IN HIVE

# Hive SerDes

- A SerDe is an interface Hive uses to read and write data
  - SerDe stands for serializer/deserializer
- You specify the SerDe when creating a table in Hive
  - Sometimes it is specified implicitly
  - SerDes enable Hive to access data that is not in structured tabular format
- Hive includes several built-in SerDes for record formats in text files

| | |
|---|---|
| LazySimpleSerDe | Using specified field delimiters (default) |
| RegexSerDe | Based on supplied regular expression patterns |
| OpenCSVSerde | In CSV format |
| JsonSerDe | In JSON format |

https://cwiki.apache.org/confluence/display/Hive/SerDe#SerDe-Built-inSerDes


# Specifying a Hive SerDe

- Previously, we specified the row format using
  - ROW FORMAT DELIMITED and FIELDS TERMINATED BY
  - `LazySimpleSerDe` is specified implicitly

```
CREATE TABLE people(fname STRING, lname STRING)
  ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t';
```

- You can also specify the SerDe explicitly
  - Using ROW FORMAT SERDE

```
CREATE TABLE people(fname STRING, lname STRING)
  ROW FORMAT SERDE
      'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
    WITH SERDEPROPERTIES ('field.delim'='\t');
```

# Regex SerDe

- Sometimes you need to analyze data that lacks consistent delimiters
  - Log files are a common example of this
    - You would need a different delimiter to extract the month of the year than to extract the hour of the day

```
05/23/2013 19:45:19 312 555 7834 CALL_RECEIVED ""
05/23/2013 19:45:23 312 555 7834 OPTION_SELECTED      "Shipping"
05/23/2013 19:46:23 312 555 7834 ON_HOLD  ""
05/23/2013 19:47:51 312 555 7834 AGENT_ANSWER "Agent ID N7501"
05/23/2013 19:48:37 312 555 7834 COMPLAINT "Item not received"
05/23/2013 19:48:41 312 555 7834 CALL_END "Duration: 3:22"
```

- RegexSerDe will read records based on a supplied regular expression that parses the line according to our business requirements
  - Allows us to create a table from this log file

# Creating A Table With Regex SerDe (1 of 3)

- Each pair of parentheses denotes a field

```
CREATE TABLE calls (
              event_date    STRING,
              event_time    STRING,
              phone_num     STRING,
              event_type    STRING,
              details       STRING)
          ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
          WITH SERDEPROPERTIES ("input.regex" =
              "([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\"");
```

  - Field value is text matched by the pattern within the corresponding parentheses
  - `input.regex` is a standard property of the `RegexSerde` that represents the regular expression being applied
- The `RegexSerDe` wasn't formally part of Hive prior to 0.10.0

# Creating A Table With Regex SerDe (2 of 3)

- Applying the Regular Expression
  - Only the parts of the expression captured in () are saved to the resulting table

| Regular Expression | Log |
|---|---|
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013_19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37_312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834_COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT_"Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |
| ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\" | 05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received" |

# Creating A Table With Regex SerDe (3 of 3)

- Input Log

```
Log
05/23/2013 19:45:19 312-555-7834 CALL_RECEIVED ""
05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received"
```

- Resulting Table

| event_date | event_time | phone_num | event_type | details |
|---|---|---|---|---|
| 05/23/2013 | 19:45:19 | 312-555-7834 | CALL_RECEIVED | |
| 05/23/2013 | 19:48:37 | 312-555-7834 | COMPLAINT | Item not received |

  - Note that the details field for the first record is empty, not NULL, because the "matching" String is empty

# CSV format

- Simple comma-delimited data can be processed using the default SerDe
- But the actual CSV format is more complex, and handles cases including
  - Embedded commas: `"Doe, John"`
  - Quoted fields: `"Doe, John"`
  - Missing values: `25,\N,…`
- Hive provides a SerDe for processing CSV data
  - OpenCSVSerde available in CDH 5.4 and later
  - Also supports other delimiters such as tab (`\t`) and pipe (`|`)

# CSV SerDe Example

- Input data

```
1,Gigabux,gigabux@example.com
2,"ACME Distribution Co.",acme@example.com
3,"Bitmonkey, Inc.",bmi@example.com
```

- DDL

```
CREATE TABLE vendors
(id INT,name STRING,email STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   "separatorChar" = ",",
   "quoteChar"     = "\"",
   "escapeChar"    = "\\"
)
;
```

These are default specifications

- Data as loaded:

| id | Name | email |
|---|---|---|
| 1 | Gigabux | gigabux@example.com |
| 2 | ACME Distribution Co. | acme@example.com |
| 3 | Bitmonkey, Inc. | bmi@example.com |

Text Analytics with Hive

# SENTIMENT ANALYSIS AND N-GRAMS

# Parsing Sentences Into Words

- Before we do sentiment analysis, we need to tokenize the text.
- Hive's `SENTENCES(input)` function parses supplied text into words
  - `Input` is a string containing one or more sentences
- Output is a two dimensional array of strings
  - Outer array contains one **element per sentence**
  - Inner array contains one **element per word** in that sentence
  - Sentence terminators (`.!?`) are removed automatically

```
hive> SELECT txt FROM phrases WHERE id = 12345;
I bought this computer and really love it! It's very fast and  does
not crash.
hive> SELECT SENTENCES(txt) FROM phrases WHERE id = 12345;
[["I","bought","this","computer","and","really","love","it"],
 ["It's","very","fast","and","does","not","crash"]]
```

# N-grams

- An n-gram is a word combination (n=number of words)
  - Bi-gram is a sequence of two words (n=2)
- N-gram frequency analysis is an important step in many applications
  - Finding the most important topics in a body of text to build a keyword index
  - Identifying trending topics in social media messages
  - Extract marketing intelligence around certain words (e.g., “Twitter is __”)
  - Find frequently accessed URL sequences

# Calculating N-grams In Hive (1 of 2)

- Hive offers an `NGRAMS` function for calculating n-grams
- `NGRAMS(input,ngram,topn)`
  - `input`: Array of strings, each containing an array of words
    - Exactly the output generated by the `SENTENCES` function
  - `ngram`: The number of words in each n-gram
  - `topn`: The desired number of results (top N, based on frequency)
- Output is an array of `STRUCT` with two attributes
  - `ngram`: The n-gram itself (an array of words)
  - `estfrequency`: The estimated frequency with which this n-gram appears

```
[{"ngram":["of","the"],"estfrequency":23.0},
{"ngram":["on","the"],"estfrequency":20.0},
{"ngram":["in","the"],"estfrequency":18.0},
{"ngram":["he","was"],"estfrequency":17.0},
{"ngram":["at","the"],"estfrequency":17.0}]
```

# Calculating N-grams In Hive

- The `NGRAMS` function is often used with the `SENTENCES` function
  - Use `UPPER` or `LOWER` to normalize case
  - Use `EXPLODE` to convert the resulting array of elements to a set of rows

```
hive> SELECT txt FROM phrases limit 2;
This tablet is great. The size is great. The screen is great. The audio
is great. I love this tablet! I love everything about this tablet!!!
hive> SELECT EXPLODE(NGRAMS(SENTENCES(LOWER(txt)), 2, 5))
                    AS bigrams FROM phrases;
{"ngram":["is","great"],"estfrequency":14.0}
{"ngram":["great","the"],"estfrequency":13.0}
{"ngram":["this","tablet"],"estfrequency":13.0}
{"ngram":["i","love"],"estfrequency":12.0}
{"ngram":["tablet","i"],"estfrequency":11.0}
```

- Further improvements:
  - Note how the bi-grams may split across sentence boundaries, unless you take preventative measures
    - e.g. `EXPLODE` each sentence before submitting it for analysis
  - You may also choose to filter out stopwords before running the NGRAMS function to eliminate them from consideration and improve performance in the process
    - E.g. create a stop words table, and use LEFT JOIN to retain only unmatched rows (non stop words).

# Finding Specific N-grams In Text

- The `CONTEXT_NGRAMS` function is similar, but considers only specific combinations
  - Additional input parameter: Array of words used for filtering
  - Any NULL values in the array are treated as placeholders

```
hive> SELECT txt FROM phrases WHERE txt LIKE '%new computer%';
My new computer is fast! I wish I'd upgraded sooner. This new computer
is expensive, but I need it now. I can't believe her new computer failed
already.
hive> SELECT EXPLODE(CONTEXT_NGRAMS(SENTENCES(LOWER(phrase)),
                    ARRAY("new", "computer", NULL, NULL), 4, 3)) AS ngrams
                    FROM phrases;
{"ngram":["is","expensive"],"estfrequency":1.0}
{"ngram":["failed","already"],"estfrequency":1.0}
{"ngram":["is","fast"],"estfrequency":1.0}
```

  - Finds the top 3, 4 word phrases that begin with “new computer”

# Where to go from here

- This exercise can be further extended
  - Remove stop words
  - Handle sentence boundaries
  - The words extracted can be combined with AFINN dictionary to calculate sentiment score for each tweet. https://goo.gl/u1n5QE
  - Visualize N-grams

# Essential Points

- The `SPLIT` function creates an array from a string
  - `EXPLODE` creates individual records from an array
- Hive has extensive support for regular expressions
  - You can extract or substitute values based on patterns
  - You can even create a table based on regular expressions
- An n-gram is a sequence of words
  - Use `NGRAMS` and `CONTEXT_NGRAMS` to find their frequency