V5_Testing Scripts

Joshua Berger, Jack Tillman, Gabe Sauvageau

Jul 05, 2019

CONTENTS:

CHAPTER

ONE

V5_TESTING

`temp_data_shifter.`**`convert_row`**`(`*row*`)`

Takes in a single row of temperature data from **TEMPS_DB** and converts it to two rows of temperature data to be inserted into *TEMPERATURES* in **RESULTS_DB**

Parameters:

`row` A dictionary representing the row retrieved from **TEMPS_DB**. Assumes the row was retrieved successfully

Returns:

`out_rows` A list containing the two newly converted rows

`temp_data_shifter.`**`insert_ignore_many_query`**`(`*table*, *rows*`)`

Generate an appropriate string for inserting potentially duplicate rows into an SQL table.

Parameters:

`table` The name of an SQL table in the current database. Assumes a table with that name exists in the database as defined by **RESULTS_DB**

`rows` a list of rows to insert into `table`. Assumes every row dictionary contains corresponding values for every data column in the table

Returns:

`query_str` A string representing an SQL statement that, if executed, performs an *INSERT IGNORE* of every row in `rows` into `table`

`temp_data_shifter.`**`round_time`**`(`*tm_str*`)`

Generates a string representation of the values in many rows, formatted for an SQL query.

Parameters:

`tm_str` A string containing a representation of a `datetime` object in standard format.

Standard format is defined as "YYYY-MM-DD hh:mm:ss(:[msecs])"

Returns:

`new_tm` A `datetime` object, containing the timestamp from `tm_str` rounded to the nearest ten second mark

`temp_data_shifter.`**`rowvals4SQLmany`**`(`*row_list*`)`

Generates a string representation of the values in many rows, formatted for an SQL query.

Parameters:

`row_list` A list containing the rows (as dictionaries) to be inserted. Assumes every row dictionary contains corresponding values for every data column in the table

Returns:

`out_str` A string containing strings generated by `rowwvals4SQLquery()` for each row in `row_list`, separated by commas

`temp_data_shifter.`**`rowvals4SQLquery`**`(`*row*`)`
Generates a string representation of the values in `row`, formatted for an SQL query.

Parameters:

`row` The dictionary representing the row. Assumes the dictionary contains corresponding values for every data column in the table

Returns:

`row_str` A string containing comma separated values from `row`, enclosed in parentheses

`temp_data_shifter.`**`temp_shifter`**`()`
Takes in temperature data from the database as defined by **TEMPS_DB**, rounds their timestamps to the nearest 10 seconds, and inserts it into the *TEMPERATURES* table in the database as defined by **RESULTS_DB**.

Assumes:

- Both databases exist as defined
- The row in *TESTS* with the time interval containing the data to be shifted exists
- the tables in both databases exist and are formatted as expected

`optimized_parsing.`**`make_datetime`**`(`*match_obj*`)`
Given a match object that contains groups with date/time info, return a corresponding python datetime object

`optimized_parsing.`**`return_or`**`(`*l*`)`

Parameters:

`l` A list of values `[v1,...,vn]`

Returns:

`or_str` The string `"((v1)|...|(vn))"`, which is formatted appropriately for use in an `or` statement in a regular expression

Example:

```
# Calling
return_or(['tea','milk','coffee'])
# Will return
"((tea)|(milk)|(coffee))"
```

`class` `optimized_parsing.`**`Log`**`(`*filename*, *units*`)`

`add_test``(`*test*`)`
Associate a new test with this log

`extract_data``()`
Read and interpret data from the log messages file given

`parse_line``(`*line*, *cur_test*, *recur_count*`)`
Given a line and a log file, and the test the current line is associated with, will convert the data in the line and in every consecutive line for the same test into a lists of rows to be inserted into the database (recursively)

`class` `optimized_parsing.`**`Test`**`(`*json_file_name=None*`)`

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX