Datadiff Documentation

Erez Shinan

Dec 01, 2022

API REFERENCE

data_diff.**connect**(*db_conf: Union[str, dict]*, *thread_count: Optional[int] = 1*, *shared: bool = True*) → Database

Provides methods for connecting to a supported database using a URL or connection dict.

Ensures all sessions use UTC Timezone, if possible.

data_diff.**connect_to_table**(*db_info: Union[str, dict]*, *table_name: Union[Tuple[str, ...], str]*, *key_columns: str = ('id',)*, *thread_count: Optional[int] = 1*, ***kwargs*) → *TableSegment*

Connects to the given database, and creates a TableSegment instance

Parameters

- **db_info** – Either a URI string, or a dict of connection options.
- **table_name** – Name of the table as a string, or a tuple that signifies the path.
- **key_columns** – Names of the key columns
- **thread_count** – Number of threads for this connection (only if using a threadpooled db implementation)

See also:

connect()

data_diff.**diff_tables**(*table1: TableSegment*, *table2: TableSegment*, ***, *key_columns: Optional[Sequence[str]] = None*, *update_column: Optional[str] = None*, *extra_columns: Optional[Tuple[str, ...]] = None*, *min_key: Optional[Union[int, str, bytes, ArithUUID, ArithAlphanumeric]] = None*, *max_key: Optional[Union[int, str, bytes, ArithUUID, ArithAlphanumeric]] = None*, *min_update: Optional[datetime] = None*, *max_update: Optional[datetime] = None*, *algorithm: Algorithm = Algorithm.HASHDIFF*, *bisection_factor: int = 32*, *bisection_threshold: int = 16384*, *threaded: bool = True*, *max_threadpool_size: Optional[int] = 1*) → Iterator

Finds the diff between table1 and table2.

Parameters

- **key_columns** (*Tuple[str, ...]*) – Name of the key column, which uniquely identifies each row (usually id)
- **update_column** (*str, optional*) – Name of updated column, which signals that rows changed. Usually updated_at or last_update. Used by *min_update* and *max_update*.
- **extra_columns** (*Tuple[str, ...], optional*) – Extra columns to compare
- **min_key** (*DbKey*, optional) – Lowest key value, used to restrict the segment
- **max_key** (*DbKey*, optional) – Highest key value, used to restrict the segment
- **min_update** (*DbTime*, optional) – Lowest update_column value, used to restrict the segment
- **max_update** (*DbTime*, optional) – Highest update_column value, used to restrict the segment
- **algorithm** (*Algorithm*) – Which diffing algorithm to use (*HASHDIFF* or *JOINDIFF*)
- **bisection_factor** (*int*) – Into how many segments to bisect per iteration. (Used when algorithm is *HASHDIFF*)
- **bisection_threshold** (*Number*) – Minimal row count of segment to bisect, otherwise download and compare locally. (Used when algorithm is *HASHDIFF*).
- **threaded** (*bool*) – Enable/disable threaded diffing. Needed to take advantage of database threads.

- **max_threadpool_size** (*int*) – Maximum size of each threadpool. None means auto. Only relevant when *threaded* is True. There may be many pools, so number of actual threads can be a lot higher.

Note: The following parameters are used to override the corresponding attributes of the given *TableSegment* instances: *key_columns*, *update_column*, *extra_columns*, *min_key*, *max_key*. If different values are needed per table, it's possible to omit them here, and instead set them directly when creating each *TableSegment*.

Example

```
>>> table1 = connect_to_table('postgresql:///', 'Rating', 'id')
>>> list(diff_tables(table1, table1))
[]
```

See also:

TableSegment *HashDiffer* *JoinDiffer*

class data_diff.**HashDiffer**(*threaded: bool = True*, *max_threadpool_size: (int+NoneType) = 1*, *bisection_factor: int = 32*, *bisection_threshold: Number = 16384*, *stats: dict[(Any*Any)] = <factory>*)

Finds the diff between two SQL tables

The algorithm uses hashing to quickly check if the tables are different, and then applies a bisection search recursively to find the differences efficiently.

Works best for comparing tables that are mostly the same, with minor discrepencies.

Parameters

- **bisection_factor** (*int*) – Into how many segments to bisect per iteration.
- **bisection_threshold** (*Number*) – When should we stop bisecting and compare locally (in row count).
- **threaded** (*bool*) – Enable/disable threaded diffing. Needed to take advantage of database threads.
- **max_threadpool_size** (*int*) – Maximum size of each threadpool. None means auto. Only relevant when *threaded* is True. There may be many pools, so number of actual threads can be a lot higher.

__init__(*threaded: bool = True*, *max_threadpool_size: (int+NoneType) = 1*, *bisection_factor: int = 32*, *bisection_threshold: Number = 16384*, *stats: dict[(Any*Any)] = <factory>*) → None

diff_tables(*table1:* TableSegment, *table2:* TableSegment, *info_tree: Optional[InfoTree] = None*) → DiffResultWrapper

Diff the given tables.

Parameters

- **table1** (TableSegment) – The "before" table to compare. Or: source table
- **table2** (TableSegment) – The "after" table to compare. Or: target table

Returns

An iterator that yield pair-tuples, representing the diff. Items can be either - ('-', row) for

items in table1 but not in table2. ('+', row) for items in table2 but not in table1. Where *row* is a tuple of values, corresponding to the diffed columns.

class data_diff.**JoinDiffer**(*threaded: bool = True*, *max_threadpool_size: (int+NoneType) = 1*, *validate_unique_key: bool = True*, *sample_exclusive_rows: bool = True*, *materialize_to_table: (tuple[str]+NoneType) = None*, *materialize_all_rows: bool = False*, *table_write_limit: int = 1000*, *stats: dict[(Any*Any)] = <factory>*)

Finds the diff between two SQL tables in the same database, using JOINs.

The algorithm uses an OUTER JOIN (or equivalent) with extra checks and statistics. The two tables must reside in the same database, and their primary keys must be unique and not null.

All parameters are optional.

Parameters

- **threaded** (*bool*) – Enable/disable threaded diffing. Needed to take advantage of database threads.
- **max_threadpool_size** (*int*) – Maximum size of each threadpool. `None` means auto. Only relevant when *threaded* is `True`. There may be many pools, so number of actual threads can be a lot higher.
- **validate_unique_key** (*bool*) – Enable/disable validating that the key columns are unique. Single query, and can't be threaded, so it's very slow on non-cloud dbs. Future versions will detect UNIQUE constraints in the schema.
- **sample_exclusive_rows** (*bool*) – Enable/disable sampling of exclusive rows. Creates a temporary table.
- **materialize_to_table** (*DbPath, optional*) – Path of new table to write diff results to. Disabled if not provided.
- **table_write_limit** (*int*) – Maximum number of rows to write when materializing, per thread.

__init__(*threaded: bool = True*, *max_threadpool_size: (int+NoneType) = 1*, *validate_unique_key: bool = True*, *sample_exclusive_rows: bool = True*, *materialize_to_table: (tuple[str]+NoneType) = None*, *materialize_all_rows: bool = False*, *table_write_limit: int = 1000*, *stats: dict[(Any*Any)] = <factory>*) → None

diff_tables(*table1:* TableSegment, *table2:* TableSegment, *info_tree: Optional[InfoTree] = None*) → DiffResultWrapper

Diff the given tables.

Parameters

- **table1** (TableSegment) – The "before" table to compare. Or: source table
- **table2** (TableSegment) – The "after" table to compare. Or: target table

Returns

An iterator that yield pair-tuples, representing the diff. Items can be either - ('-', row) for items in table1 but not in table2. ('+', row) for items in table2 but not in table1. Where *row* is a tuple of values, corresponding to the diffed columns.

class data_diff.**TableSegment**(*database: Database = <object object>*, *table_path: tuple[str] = <object object>*, *key_columns: tuple[str] = <object object>*, *update_column: (str+NoneType) = None*, *extra_columns: tuple[str] = ()*, *min_key: (NoneType+(int+ArithUUID+ArithAlphanumeric+bytes+str)) = None*, *max_key: (NoneType+(int+ArithUUID+ArithAlphanumeric+bytes+str)) = None*, *min_update: (datetime+NoneType) = None*, *max_update: (datetime+NoneType) = None*, *where: (str+NoneType) = None*, *case_sensitive: bool = True*, *_schema: (CaseAwareMapping+NoneType) = None*)

Signifies a segment of rows (and selected columns) within a table

Parameters

- **database** (*Database*) – Database instance. See *connect()*
- **table_path** (*DbPath*) – Path to table in form of a tuple. e.g. *('my_dataset', 'table_name')*
- **key_columns** (*Tuple[str]*) – Name of the key column, which uniquely identifies each row (usually id)
- **update_column** (*str, optional*) – Name of updated column, which signals that rows changed. Usually updated_at or last_update. Used by *min_update* and *max_update*.
- **extra_columns** (*Tuple[str, ...], optional*) – Extra columns to compare
- **min_key** (*DbKey*, optional) – Lowest key value, used to restrict the segment
- **max_key** (*DbKey*, optional) – Highest key value, used to restrict the segment
- **min_update** (*DbTime*, optional) – Lowest update_column value, used to restrict the segment
- **max_update** (*DbTime*, optional) – Highest update_column value, used to restrict the segment
- **where** (*str, optional*) – An additional 'where' expression to restrict the search space.
- **case_sensitive** (*bool*) – If false, the case of column names will adjust according to the schema. Default is true.

with_schema() → *TableSegment*

Queries the table schema from the database, and returns a new instance of TableSegment, with a schema.

get_values() → list

Download all the relevant values of the segment from the database

choose_checkpoints(*count: int*) → List[Union[int, str, bytes, ArithUUID, ArithAlphanumeric]]

Suggests a bunch of evenly-spaced checkpoints to split by (not including start, end)

segment_by_checkpoints(*checkpoints: List[Union[int, str, bytes, ArithUUID, ArithAlphanumeric]]*) → List[*TableSegment*]

Split the current TableSegment to a bunch of smaller ones, separated by the given checkpoints

new(***kwargs*) → *TableSegment*

Using new() creates a copy of the instance using 'replace()'

count() → int

Count how many rows are in the segment, in one pass.

count_and_checksum() → Tuple[int, int]

Count and checksum the rows in the segment, in one pass.

__init__(*database: Database = <object object>*, *table_path: tuple[str] = <object object>*, *key_columns: tuple[str] = <object object>*, *update_column: (str+NoneType) = None*, *extra_columns: tuple[str] = ()*, *min_key: (NoneType+(int+ArithUUID+ArithAlphanumeric+bytes+str)) = None*, *max_key: (NoneType+(int+ArithUUID+ArithAlphanumeric+bytes+str)) = None*, *min_update: (datetime+NoneType) = None*, *max_update: (datetime+NoneType) = None*, *where: (str+NoneType) = None*, *case_sensitive: bool = True*, *_schema: (CaseAwareMapping+NoneType) = None*) → None

data_diff.**DbKey**

The central part of internal API.

This represents a generic version of type 'origin' with type arguments 'params'. There are two kind of these aliases: user defined and special. The special ones are wrappers around builtin collections and ABCs in collections.abc. These must have 'name' always set. If 'inst' is False, then the alias can't be instantiated, this is used by e.g. typing.List and typing.Dict.

alias of Union[int, str, bytes, ArithUUID, ArithAlphanumeric]

data_diff.**DbTime = <class 'datetime.datetime'>**

datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass. The remaining arguments may be ints.

data_diff.**DbPath**

The central part of internal API.

This represents a generic version of type 'origin' with type arguments 'params'. There are two kind of these aliases: user defined and special. The special ones are wrappers around builtin collections and ABCs in collections.abc. These must have 'name' always set. If 'inst' is False, then the alias can't be instantiated, this is used by e.g. typing.List and typing.Dict.

alias of Tuple[str, ...]

enum data_diff.**Algorithm**(*value*)

An enumeration.

Valid values are as follows:

AUTO = <Algorithm.AUTO: 'auto'>

JOINDIFF = <Algorithm.JOINDIFF: 'joindiff'>

HASHDIFF = <Algorithm.HASHDIFF: 'hashdiff'>

CHAPTER

ONE

DATA-DIFF

Data-diff is a command-line tool and Python library to efficiently diff rows across two different databases.

- Verifies across many different databases (e.g. *PostgreSQL -> Snowflake*) !
- Outputs diff of rows in detail
- Simple CLI/API to create monitoring and alerts
- Verify 25M+ rows in <10s, and 1B+ rows in ~5min.
- Works for tables with 10s of billions of rows

For more information, See our README

CHAPTER

TWO

RESOURCES

- Source code (git): https://github.com/datafold/data-diff
- *Python API Reference*
- The rest of the documentation

PYTHON MODULE INDEX

d

INDEX