Load and Transform Guide

Version 2.1.0

Table of Contents

License Statement

Disclaimer: *Apache Trafodion is an effort undergoing incubation at the Apache Software Foundation (ASF), sponsored by the Apache Incubator PMC. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.*

Acknowledgements

WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

Revision History

Version	Date
2.1.0	TBD
2.0.1	July 7, 2016
2.0.0	June 6, 2016
1.3.0	January, 2016

Chapter 1. About This Document

This guide describes how to load and transform data into a Trafodion database.

The information herein is complementary to the following Trafodion documentation:

- Trafodion SQL Reference Manual
- Trafodion odb User Guide

1.1. Intended Audience

This guide targets anyone wanting to load data into a Trafodion database.

You need to have skills in the following areas to make full use of the information in this guide:

- SQL DDL and DML.
- Installation and configuration of Linux software.
- Trafodion administration.
- Depending on your data source, Java and/or Hadoop ecosystem usage.

1.2. New and Changed Information

This is a new guide.

1.3. Notation Conventions

This list summarizes the notation conventions for syntax presentation in this manual.

- UPPERCASE LETTERS

  Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required.

  ```
  SELECT
  ```

- lowercase letters

  Lowercase letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required.

  ```
  file-name
  ```

- [ ] Brackets

  Brackets enclose optional syntax items.

  ```
  DATETIME [start-field TO] end-field
  ```

  A group of items enclosed in brackets is a list from which you can choose one item or none.

  The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines.

  For example:

  ```
  DROP SCHEMA schema [CASCADE]
  DROP SCHEMA schema [ CASCADE | RESTRICT ]
  ```

- { } Braces

  Braces enclose required syntax items.

  ```
  FROM { grantee [, grantee ] ... }
  ```

A group of items enclosed in braces is a list from which you are required to choose one item.

The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines.

For example:

```
INTERVAL { start-field TO end-field }
{ single-field }
INTERVAL { start-field TO end-field | single-field }
```

- | Vertical Line

A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces.

```
{expression | NULL}
```

- … Ellipsis

An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times.

```
ATTRIBUTE[S] attribute [, attribute] ...
{, sql-expression } ...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times.

For example:

```
expression-n ...
```

- Punctuation

Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown.

```
DAY (datetime-expression)
@script-file
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must

type as shown.

For example:

```
"{" module-name [, module-name] ... "}"
```

- Item Spacing

  Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma.

  ```
  DAY (datetime-expression) DAY(datetime-expression)
  ```

  If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

  ```
  myfile.sh
  ```

- Line Spacing

  If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line.

  This spacing distinguishes items in a continuation line from items in a vertical list of selections.

  ```
  match-value [NOT] LIKE _pattern
     [ESCAPE esc-char-expression]
  ```

1.4. Comments Encouraged

We encourage your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to user@trafodion.incubator.apache.org.

Include the document title and any comment, error found, or suggestion for improvement you have concerning this document.

Chapter 2. Introduction

2.1. Load Methods

There are two methods used to load data into a Trafodion table. Both methods can run while the database is concurrently queried:

Type	Description	Methods/Tools
Bulk Load	Large data volumes Stage data and load in the batches	Trafodion Bulk Loader
Trickle Load	Small data volumes Insert data as it arrives	ETL tool Custom ODBC/JDBC application User-Defined Functions odb Tool

These two methods use four types of SQL insert statements

- **Bulk Load**
  - LOAD
- **Trickle Load**
  - INSERT
  - UPSERT
  - UPSERT USING LOAD

The Trafodion SQL Reference Manual provides syntax descriptions for these statements.

The data source defines what type of load approach and method you use:

- **Bulk Load** (LOAD statement)
  - *Text Files*: Map an external Hive table.
  - *JDBC-Compliant Database*: Load into Hive on the Trafodion cluster using `sqoop`.
  - *Hive Tables*: Direct load.
  - *Native HBase Tables*: Direct load.
  - *Disparate Data Source*: Write Java/C++ UDF to read data from source and pass rows to LOAD.

- **Trickle Load** (odb utility)
  - *Text Files*: Direct access
  - *pipes*: Via `stdin`
  - *ODBC-Compliant Database*: odb COPY command, no intermediate storage

For more information, refer to:

- Bulk Load
- Trickle Load

2.1.1. Insert Types

The following insert types are supported in Trafodion:

- `INSERT INTO T …`
- `UPSERT INTO T …`
- `UPSERT USING LOAD INTO T …`
- `LOAD INTO T …`

The following table compares the different insert types:

Characteristic	INSERT	UPSERT	UPSERT USING LOAD	LOAD
Transaction	Yes	Yes	No, uses HBase WAL for recovery	No, uses snapshot for recovery
Method of Operation	Uses the standard HBase write path through its `CheckAndPut` call. Rows are held in transaction co-processor memory until the transaction is committed.	Uses the standard HBase write path through its `Put` call.	Uses the standard HBase write path through its `Put` call.	Uses the HBase bulk load write path and creates HFiles directly, bypassing HBase RegionServers for most of its operation.
Uniqueness Constraint	Enforced	Not enforced. New row with the same key value overwrites previous row.	Not enforced. New row with same key value overwrites the previous row.	Enforced only within the set of rows in a single statement. Not enforced with rows already in the table.
Index	Can be used on a table with an index.	Can be used on a table with an index.	When used on a table with an index, it reverts to UPSERT.	Can be used on a table with an index. Index is off-line during the LOAD.
Max Size/Invocation	10,000 * n[1] rows	10,000 * n[1] rows	5 million * n[1] rows	2 billion * n[1] rows
Min Size/Invocation	1 row	1 row	1 row	Suitable for greater than 1 million * n[1] rows
Speed	Slowest	Faster than INSERT	Faster than UPSERT	Fastest

[1] **n** is the number of nodes in each invocation.

Throughput, max/min sizes depends on multiple factors:

- Format of rows in Trafodion table (aligned format or not).
- Length of row.
- Number of columns in row.

- Data type of columns.
- Network between nodes in cluster.
- WAL setting.
- Number of clients.
- Use of rowsets.

2.2. Unload

The Trafodion UNLOAD statement exports data from Trafodion tables into an HDFS directory. Refer to Bulk Unload for more information.

Chapter 3. Tables and Indexes

The following guidance helps you set up your tables and indexes for better load performance.

3.1. Choose Primary Key

The primary key for a Trafodion table must be chosen based on the workload that accesses the table.

Keyed access to Trafodion tables is very efficient since HBase is a key-value store. You need to analyze the queries that are used to access the tables to understand their predicates and join conditions. Once identified, you can choose a primary key that ensures that the leading key columns have highly selective predicates applied to them.

This technique limits the number of rows that need to scanned in the HBase. Trafodion uses MDAM (Multi Dimensional Access Method) to limit the rows scanned when predicates are present to only trailing key columns and not the leading key column. MDAM works best when the unique entry count of leading key columns (on which predicates are absent) is low.

3.2. Salting

With range partitioned data in some workloads, certain key ranges of data may see more access than other key ranges. This can lead to an unbalanced usage pattern with some HBase RegionServers handling most of the load. This behavior is referred to as "hot-spotting."

With Native HBase tables, hot-spotting is often addressed by designing appropriate keys. In Trafodion, once you choose the key to a table, as discussed in Choose Primary Key, you can use **salting** to distribute the data evenly. Salting applies a hash function to the salt keys and distributes data to partitions based on this hash value. The hash value is physically stored in the table as the leading key value. Each split of the table will have only one salt key value.

The salting key can be any subset (including the whole set) of the primary key. It is a good practice to keep the salting key as small as possible. The key should provide an even distribution of data, which can be achieved when the key values have a large unique entry count and no significant skew.

The number of partitions must also be specified during table creation. You choose the number of partition depending on the size of the cluster and the expected size of the table. A salted table can split if more data is added to it than initially estimated. If this happens, then more than one partition having rows with the same salt value, which may result in suboptimal execution plans for the table.

You can also choose not to salt Trafodion tables. This is similar to range partitioning in a traditional database. The number of partitions grows with the size of the table, and range boundaries are determined by HBase based on the specified split policy.

3.3. Compression and Encoding

Large Trafodion tables must be encoded and compressed. Trafodion tables that have a large key or several columns grow in size to 10X or more when compared to a Hive table with equivalent data since HBase stores the key separately for every column in a row.

HBase provides several types of encoding to avoid storing the same key value to disk for every column in the row. HBase also supports various types of compression of the entire data block, regardless whether it is encoded or not. See Appendix E: Compression and Data Block Encoding In HBase in the Apache HBase Reference Guide for a comparison of various compression and encoding algorithms. Use the information in the Which Compressor or Data Block Encoder To Use section to determine the best compression technique for your tables. <<<

3.4. Create Tables and Indexes

Create Trafodion tables using the CREATE TABLE statements with the `SALT USING <num> PARTITIONS` clause for salting and the `HBASE_OPTIONS` clause for compression and encoding.

Example

```
CREATE TABLE trafodion.sch.demo
( demo_sk INT NOT NULL
, name VARCHAR(100)
, PRIMARY KEY (demo_sk)
)
HBASE_OPTIONS
( DATA_BLOCK_ENCODING = 'FAST_DIFF'
, COMPRESSION = 'SNAPPY'
, MEMSTORE_FLUSH_SIZE = '1073741824'
)
SALT USING 8 PARTITIONS ON (demo_sk);
```

ANY indexes on the table may be salted or not. However, if they are salted, their salting key and number of partitions must be the same as the table.

Example

```
CREATE INDEX demo_ix ON sch.demo(name)
HBASE_OPTIONS
( DATA_BLOCK_ENCODING = 'FAST_DIFF'
, COMPRESSION = 'GZ'
)
SALT LIKE TABLE;
```

3.5. Update Statistics

To generate good plans that allow queries to execute quickly and use resources wisely, the Trafodion Optimizer must have a good idea about how the values of columns are distributed, the number of distinct values, and so on. Trafodion supplies this information to the optimizer in the form of histograms generated by executing the UPDATE STATISTICS statement. See the Trafodion SQL Reference Manual for a full description of this statement.

3.5.1. Default Sampling

While accurate statistics are important, the time required to generate them by reading every row in the table may be prohibitive and is usually unnecessary. Random sampling of the rows of the table can give adequate results in a fraction of the time required to read all the values in the table. For most situations, the best option is to simply specify SAMPLE at the end of the UPDATE STATISTICS statement, which will use the default sampling protocol. For example, to use default sampling in the construction of histograms for each column of table T1, you would execute the following statement:

```
UPDATE STATISTICS FOR TABLE t1 ON EVERY COLUMN SAMPLE;
```

This default sampling protocol uses a high sampling rate for small tables, reducing the rate with a steep gradient until hitting 1% and capping the sample size at one million rows. The specific details of default sampling are as follows:

- Use the full table for tables up to 10,000 rows.
- For table sizes from 10,000 up to a million rows, 10,000 rows are randomly sampled. In effect, this causes the sampling rate to decline from 100% to 1% as a function of increasing table size.
- For tables with one million to 100 million rows, use a 1% random sample.
- For tables exceeding 100 million rows, the sampling rate is calculated as 1 million divided by the number of rows in the table. This limits the overall sample size to 1 million rows while ensuring uniform random sampling across the entire table.

3.6. Generate Single-Column and Multi-Column Histograms From One Statement

If you use the ON EVERY COLUMN syntax in an UPDATE STATISTICS statement, then it is important to realize that multi-column histograms can be requested in the same statement. For example, if you wanted to generate a histogram for each single column of table T1, as well as multi-column histograms for column sets (c1, c2) and (c5, c6, c7), then you could use the following statement:

```
UPDATE STATISTICS FOR TABLE t1 ON EVERY COLUMN, (c1,c2), (c5,c6,c7) SAMPLE;
```

In terms of the end result, this is equivalent to the following pair of statements:

```
UPDATE STATISTICS FOR TABLE t1 ON EVERY COLUMN SAMPLE;
UPDATE STATISTICS FOR TABLE t1 ON (c1, c2), (c5, c6, c7) SAMPLE;
```

However, the performance is superior when they are combined into a single statement because a multi-column histogram depends on the single-column histograms of its component columns. Therefore, separating the generation of single-column and multi-column histograms for a table into two statements leads to redundantly calculating some of the single-column histograms. Even though the relevant single-column histograms already exist, they are recomputed at the time the multi-column histograms are generated.

3.6.1. Enable Update Statistics Automation

If a standard set of queries is run on a regular basis, then one way to generate only those histograms that are needed for efficient execution of those queries is to enable update statistics automation, and then PREPARE each of the queries:

```
CONTROL QUERY DEFAULT USTAT_AUTOMATION_INTERVAL '1440';
PREPARE s FROM SELECT...;
```

The value of the CQD USTAT_AUTOMATION_INTERVAL is intended to determine the automation interval (in minutes) for update statistics automation. The PREPARE statement causes the Trafodion Compiler to compile and optimize a query without executing it. In the process of doing so with automation enabled, any histograms needed by the optimizer that are missing causes those columns to be marked as needing histograms. Then, the following UPDATE STATISTICS statement can be run against each table to generate the needed histograms:

```
UPDATE STATISTICS FOR TABLE <table-name> ON NECESSARY COLUMNS SAMPLE;
```

3.6.2. Regenerate Histograms

Histograms can become "stale" as the underlying data changes and possibly reflects a different distribution of values, although it is possible that data turnover or accumulation can be high while maintaining the same distribution. To ensure that statistics remain accurate, you should regenerate histograms for a table once significant changes have been made to that table since its histograms were last generated. To refresh existing histograms without adding new ones, use the following statement:

```
UPDATE STATISTICS FOR TABLE <table-name> ON EXISTING COLUMNS SAMPLE;
```

The critical set of histograms that were previously generated with the ON NECESSARY COLUMNS syntax can be periodically regenerated using ON EXISTING COLUMNS. Note that using ON NECESSARY COLUMNS will only identify those columns that have been previously requested by the optimizer but do not exist. The current implementation of automation does not know which existing histograms might be stale.

Chapter 4. Bulk Load

The LOAD statement enables batch loading large volumes of data efficiently in a scalable manner.

See the Trafodion SQL Reference Manual for a full description of this SQL statement.

You can bulk-load data using one of the following methods:

- Load Data From Trafodion Tables
- Load Data From HDFS Files
- Load Data From Hive Tables
- Load Data From External Databases

4.1. Load Data From Trafodion Tables

You copy data between two Trafodion tables by using the appropriate SELECT statement in the LOAD command.

4.1.1. Example

```
LOAD INTO target_table SELECT * FROM source_table WHERE custkey >= 1000 ;
```

4.2. Load Data From HDFS Files

You copy your data (local or remote) into an HDFS folder. Then, you create an external Hive table (with correct fields) that points to the HDFS folder containing the data. You may also specify a WHERE clause on the source data as a filter, if needed. See the External Tables page on the Hive Wiki for more information.

Trafodion can access columns in Hive tables having integer, string and char types. See the LanguageManual Types page on the Hive Wiki for the data types available in Hive.

Overall, you do the following:

1. Export the data on the local or remote cluster.
2. If applicable, transport files to Trafodion cluster via FTP, scp, or some other method.
3. Use LOAD referencing HIVE external tables.

4.2.1. Example

You have a customer-demographics in a text file, which you need to load into Trafodion. The columns are separated by `|`.

Do the following:

1. Using trafci, define the Trafodion table where you want to load the data.

```
CREATE TABLE customer_demographics_salt
(
    cd_demo_sk            INT NOT NULL
  , cd_gender             CHAR(1)
  , cd_marital_status     CHAR(1)
  , cd_education_status   CHAR(20)
  , cd_purchase_estimate  INT
  , cd_credit_rating      CHAR(10)
  , cd_dep_count          INT
  , cd_dep_employed_count INT
  , cd_dep_college_count  INT
  , PRIMARY KEY (cd_demo_sk)
)
SALT USING 4 PARTITIONS ON (cd_demo_sk)
;
```

2. Copy the data into HDFS

```
hadoop fs -copyFromLocal $HOME/data/customer_demographics
/hive/tpcds/customer_demographics
```

3. Using the Hive shell, create an external Hive table:

```
CREATE EXTERNAL TABLE customer_demographics
(
    cd_demo_sk            INT
  , cd_gender             STRING
  , cd_marital_status     STRING
  , cd_education_status   STRING
  , cd_purchase_estimate  INT
  , cd_credit_rating      STRING
  , cd_dep_count          INT
  , cd_dep_employed_count INT
  , cd_dep_college_count  INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LOCATION '/hive/tpcds/customer_demographics'
;
```

4. Using trafci, load the Trafodion `customer_demographics_salt` table from the Hive table named `hive.hive.customer_demographics`:

```
>>LOAD INTO customer_demographics_salt
+>SELECT * FROM hive.hive.customer_demographics WHERE cd_demo_sk <= 5000;
Task: LOAD Status: Started Object: TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: DISABLE INDEX Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: DISABLE INDEX Status: Ended Object: TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: PREPARATION Status: Started Object: TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
       Rows Processed: 5000
Task: PREPARATION Status: Ended ET: 00:00:03.199
Task: COMPLETION Status: Started Object: TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: COMPLETION Status: Ended ET: 00:00:00.331
Task: POPULATE INDEX Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: POPULATE INDEX Status: Ended ET: 00:00:05.262
```

4.3. Load Data From Hive Tables

You can import data from Hive using the trafci or sqlci command interface. Do the following:

1. Set these required Control Query Defaults (CQDs) to improve load performance:

```
CQD HIVE_MAX_STRING_LENGTH '1000'; -- if the widest column is 1KB
```

   This setting is required if there are time-related column types in the target Trafodion table.

```
CQD ALLOW_INCOMPATIBLE_ASSIGNMENT 'on';
```

2. Issue the LOAD statement to load data into Trafodion tables from Hive. For example:

```
LOAD WITH NO POPULATE INDEXES INTO trafodion.sch.demo SELECT * FROM hive.hive.demo;
```

See the Trafodion SQL Reference Manual. for the complete syntax of the LOAD statement.

If you use multiple LOAD statements to incrementally load sets of data into a single target table, then several HFiles are created for each partition of the target table. This causes inefficient access during SELECT queries and may also cause a compaction to be triggered based on the policies configured in the HBase settings.

To avoid this issue, it is good practice to perform a major compaction on a table that has been the target of more than two

LOAD statements in a short period of time. To perform compaction, use this `hbase shell` command:

```
major_compact 'TRAFODION.SCH.DEMO'
```

The `major_compact` command returns immediately since it's not waited. Typically, compaction of a large table takes a long time (several minutes to hours) to complete. You can monitor the progress of compaction from the HBase Master Web user interface.

4.3.1. Example

```
>> CQD HIVE_MAX_STRING_LENGTH '1000' ;
>> CQD ALLOW_INCOMPATIBLE_ASSIGNMENT 'on' ;
>> LOAD WITH NO POPULATE INDEXES INTO trafodion.sch.demo SELECT * FROM hive.hive.demo ;
```

4.4. Load Data From External Databases

You need to import data into Hive when loading data from external databases. Use Apache Sqoop, an open-source tools to move the data from the external database into Hive tables on the Trafodion cluster.

Source data can be in the following formats:

Format	Examples
Structured	Relational databases such as Oracle or MySQL.
Semi-Structured	Cassandra or HBase
Unstructured	HDFS

You use the Sqoop command-line shell for interactive commands and basic scripting.

Sqoop basics:

- Generic JDBC Connector: supports JDBC T-4 Driver.
- Configuration Language for FROM/TO jobs that specify in SQL terms.
- Partitioner: Divide/parallelize the data streams; uses primary key by default.
- Extractor: Uses FROM configuration for SQL statements, plus partitioner information to query data subsets.
- Loader: Uses TO job configuration; INSERT INTO could be generated from col list or explicitly specified.
- Destroyer: Copies staging table to final table and deletes staging table.

See the Sqoop 5 Minutes Demo for a quick introduction to Sqoop.

4.4.1. Install Required Software

By default, Sqoop is not installed on Trafodion clusters. Do the following:

- Install and start Sqoop on the Trafodion cluster using either the Ambari or Cloudera Manager GUI. See the Sqoop installation instructions.
- Install JDK 1.8
- Install the Oracle JDBC driver
- Set the following environment variables:

```
export JAVA_HOME=/opt/java/jdk1.8.0_11
export JAVA_OPTIONS=-Dmapred.child.java.opts=\-Djava.security.egd=file:/dev/urandom+
```

4.4.2. Sample Sqoop Commands

List All Oracle Tables

```
sqoop list-tables --driver oracle.jdbc.OracleDriver
--connect jdbc:oracle:thin:@<Oracle host name>:<port>/<database>
--username <user-name> --password <password>
```

Import Data to Hive

Syntax

```
sqoop import --connect jdbc:oracle:thin:@<Oracle host name:port>/<database>
--username <user-name> --password <password> --table <tablename>
--split-by <column-name> --hive-import --create-hive-table
--hive-table <hive-table-name> --hive-overwrite --null-string ''
--null-non-string '' --hive-drop-import-delims--verbose
```

Parameter	Guidelines
`--split-by <column-name>`	By default, if not specified, sqoop uses the primary key column as a splitting column, which is not optimal most of the time. If the table does not contain a primary key, then you must manually specify the splitting column.
`--null-string <null-string>`	This is the string to be written for a null value in a string column.
`--null-non-string <null-string>`	This is the string to be written for a null value in a non-string column.

Parameter	Guidelines
`--hive-drop-import-delims`	This drops `\n`, `\r`, and `\01` string fields when importing to Hive. **NOTE**: If the data contains \n or \r and if you do not use the hive-drop-import-delims option, then data is truncated. You need to use additional Sqoop options during migration by specifying the delimiter that you would like to use, which does not exist in the data itself.

4.4.3. Example

```
sqoop import --connect jdbc:oracle:thin:@localhost:1521/orcl
--username trafdemo --password traf123 --table CUSTOMER
--split-by CUSTNUM --hive-import --create-hive-table
--hive-table customers --hive-overwrite --null-string ''
--null-non-string '' --hive-drop-import-delims--verbose
```

Chapter 5. Trickle Load

Trafodion Trickle Load allows data to be committed in batches, with sizes ranging from 1 row to a several thousand rows in each commit. Trickle Load uses the following SQL statements (defined in the Trafodion SQL Reference Manual:

- INSERT
- UPSERT
- UPSERT USING LOAD

Contrary to Bulk Load, committed rows are immediately visible from other transactions thereby leading to minimal latency in making newly ingested rows visible to applications and end users.

You use Trickle Load in the following situations:

- Inserting and/or updating data on an ongoing basis. Typically, you create a custom JDBC or ODBC application for this approach.
- You want to migrate a smaller amount of data (a few millions rows). Typically, you use JDBC- or ODBC-based ETL tools for this approach; for example:
  - Trafodion odb[1]
  - SQuirrel-SQL
  - Pentaho
  - Informatica.

[1] Trafodion obd typically achieves better load throughput than third-party ETL tools.

5.1. Improving Throughput

Trickle Load uses the HBase write path, with every row being written to the WAL (Write-Ahead Log) and HBase MemStore. When memstore is full data is flushed to HStorefile in background.

Throughput can be improved by use of:

- Rowsets or Batch Updates.
- UPSERT instead of INSERT statements, if applicable.
- Multiple simultaneous connections.

In addition, when using INSERT or UPSERT USING LOAD with the objective of maximizing data ingestion throughput, increasing the HBase table attribute MEMSTORE_FLUSHSIZE from its default value helps.

The actual value you use depends on the heap size allocated to each Region Server, the concurrent query workload, and the number of tables for which simultaneous fast data ingestion is needed. With a heap size of 31 GB for each Region Server in an environment with heavy concurrent query workload, setting this attribute 1 GB gives good performance.

You can specify this attribute in the HBASE_OPTIONS clause when creating the table. Alternatively, you can also set it from the `hbase shell` through an `ALTER 'TRAFODION.<schema-name>.<table-name>', MEMSTORE_FLUSHSIZE >= '1073741824'` command.

5.2. odb

odb is a Linux and Windows Trafodion client that is:

- ODBC based
- Database agnostic query driver
- Query interpreter
- Loader and extractor

odb may be installed on:

- The Trafodion cluster.
- The machine that contains source data
- An intermediate machine that is being used for data loading.

Source data can be in any database (local or remote) that supports ODBC or in flat files local to the machine hosting the odb tool.

odd uses threads to achieve parallelism, rowsets to improve throughput. You can specify INSERT, UPSERT or UPSERT USING LOAD insert types.

odb does not use the bulk load command LOAD, and, therefore, throughput when using odb may be lower than what can be achieved with the bulk loader. However, when using the odb tool, source data need not be moved to the Trafodion cluster in a separate step.

odb allows you to access Hadoop data using one of the following methods:

1. **Use Hive and its ODBC Driver**: odb can access HIVE like any other relational database. For example, you can copy to from HIVE and other databases using odb's copy option.
2. **Add the hdfs.`** prefix to the input or output file during loads/extracts*: The file is read/written from/to Hadoop. odb interacts directly with the HDFS file system using **libhdfs**.

   This option is currently available only under Linux.

The following odb commands/features are discussed in this guide:

- odb Load
- odb Copy
- odb Extract
- odb Transform

See the Trafodion odb User Guide for installation instructions and usage syntax for the odb tool.

The following subsections assume that you've installed odb.

5.2.1. odb Throughput

You achieve the best throughput with odb if using the UPSERT USING LOAD option.

The default insert type used by odb is INSERT; to use UPSERT USING LOAD, please specify `:loadcmd=UL` in odb's `load` or `copy` command.

Example

Copy the table `mytable` from `<source_catalog>.<source_schema>` on the source database to `trafodion.myschema.mytable` on Trafodion.

```
odb64luo -u <src_username>:<tgt_username> -p <src_pswd>:<tgt_pswd>
-d <src_dsn>:<tgt_dsn>
-cp src:<source_catalog>.<source_schema>.mytable tgt:trafodion.myschema.mytable
:splitby=<col-name>:parallel=4:loadcmd=UL
```

Option	Defines
`src_username`	User name for the source database.
`src_pswd`	Password for the source database.
`src_dsn`	ODBC DSN for the source database.
`tgt_username`	User name for the Trafodion database.

Option	Defines
`tgt_pswd`	Password for the Trafodion database.
`tgt_dsn`	ODBC DSN for the Trafodion database.
`splitby`	Defines the column used to evenly distributed values for parallelism. Consider using a leading key column.
`parallel=4`	Use four connections to extract data from the source database and another four connections to write data to the target Trafodion database.
`loadcmd=UL`	Use UPSERT USING LOAD syntax to write data.

5.2.2. odb Load

Refer to the Load Files section in the Trafodion odb User Guide for complete documentation of this option.

You use the `-l` option to load into a table from:

- File or standard input (pipe)
- gzip compressed files with no external dependencies
- HDFS
- Load XML files
- Delimited and fixed format files
- "Binary" files (example images)
- Generic SQL scripts before/after loads

The `-l` option provides:

- Data generation (constant, sequences, random values, from external datasets)
- Configurable rowsets

You can load single tables or list of tables in the same session using single/parallel threads. Limited "ETL like" functionalities are provided; for example: SUBSTR, TRANSLITERATION, TRUNCATE target, DATE/TIME format conversion, and TOUPPER.

Important Options

Option	Defines
`src`	Source file. If empty, then odb generates sample data.
`fs`	Field separator.
`tgt`	Target table, required.
`map`	Map file. A text file describing which input column is mapped to which target table column. See odb Transform below.

Option	Defines
`rows`	Rowset size to be used.
`parallel`	Number of connections/threads to be used.`
`loadcmd`	`IN`, `UP` or `UL`. INSERT, UPSERT or UPSERT USING LOAD. Use UL for best throughput.

Example

```
$ odb64luo -u user -p xx -d dsn -l src=customer.tbl:tgt=TRAFODION.MAURIZIO.CUSTOMER \
:fs=\|:rows=1000:loadcmd=UL:truncate:parallel=4
```

This command:

- Loads the file named `customer.tbl` (`src=customer.tbl`)
- in the table `TRAFODION.MAURIZIO.CUSTOMER` (`tgt=TRAFODION.MAURIZIO.CUSTOMER`)
- using `|` (vertical bar) as a field separator (`fs=\|`)
- using `1000 rows` as row-set buffer (`rows=1000`)
- using UPSERT USING LOAD syntax to achieve better throughput
- truncating the target table before loading (`truncate`)
- using `4 parallel threads` to load the target table (`parallel=4`)

```
./odb64luo -u xx -p yy -d traf_sqws125 -l
src=myfile:fs=|:tgt=TRAFODION.SEABASE.REGION:map=region.map:max=10000:rows=500:parallel
=2:loadcmd=UL
```

You can load multiple files using different `-l` options. By default odb creates as many threads (and ODBC connections) as the sum of parallel load threads.

Example

Truncates and load the CUSTOMER, ORDERS and LINEITEM tables in parallel.

```
odb64luo -u user -p xx -d dsn -T 5 \
-l src=./data/%t.tbl.gz:tgt=TRAFODION.MAURO.CUSTOMER:fs=\
|:rows=m2:truncate:norb:parallel=4 \
-l src=./data/%t.tbl.gz:tgt=TRAFODION.MAURO.ORDERS:fs=\
|:rows=1000:truncate:norb:parallel=4 \
-l src=./data/%t.tbl.gz:tgt=TRAFODION.MAURO.LINEITEM:fs=\
|:rows=m10:truncate:norb:parallel=4
```

5.2.3. odb Copy

Refer to the Copy Tables From One Database to Another section in the Trafodion odb User Guide for complete documentation of this option.

Use the `-cp` option to copy tables **directly** from one data-source to another using ODBC (for example, from Trafodion to Teradata or vice-versa):

- Single/Multiple table(s) copy from one database to another
- Data never lands to disk (ODBC buffers moved from source to target)
- Multi-threaded copy: single/multiple tables in parallel using single/multiple "data streams"/table
- Each "data stream" consists of one "extractor" and one or more "loaders"
- Table subsets copy (columns and/or rows)
- No data conversion required
- Other functionalities: sequence creation, limit text col length, max rows to copy, . . .
- Each data stream is "multiple buffered" with loaders and extractors working in parallel (no need to extract before loading).

The target table has to be be created in advance and should have a compatible structure.

Important Options

Option	Defines
`src`	Source file. If empty, then odb generates sample data.
`fs`	Field separator.
`tgt`	Target table, required.
`parallel`	Number of connections/threads to be used.`
`splitby`	Source column to parallelize copy operation on.
`pwhere`	`where` condition on source
`loadcmd`	`IN`, `UP` or `UL`. INSERT, UPSERT or UPSERT USING LOAD. Use UL for best throughput.

When copying data from one data source to another, odb needs user/password/dsn for both source and target system. User credentials and DSN for the target system are specified this way:

```
$ odb64luo -u src_user:tgt_user -p src_pwd:tgt:pwd -d src_dsn:tgt_dsn ... -cp
src=...:tgt=...
```

You can use odb to copy a list of tables from one database to another.

Example

```
$ cat tlist.txt
# List of tables to extract
src=TRAFODION.MAURIZIO.ORDERS
src=TRAFODION.MAURIZIO.CUSTOMER
src=TRAFODION.MAURIZIO.PART
src=TRAFODION.MAURIZIO.LINEITEM
```

You can extract all these tables by running:

```
$ odb64luo -u user1:user2 -p xx:yy -d dsn1:dsn2 \
-cp src=-tlist.txt:tgt=tpch.stg_%t:rows=m2:truncate:parallel=4
```

Please note the `src=-tlist.txt`. This command copies:

Source	Target
`TRAFODION.MAURIZIO.ORDERS`	`tpch.stg_orders`
`TRAFODION.MAURIZIO.CUSTOMER`	`tpch.stg_customer`
`TRAFODION.MAURIZIO.PART`	`tpch.stg_part`
`TRAFODION.MAURIZIO.LINEITEM`	`tpch.stg_lineitem`

Optionally, you can define any other command-line options in the input file.

Example

Using different *splitby columns*.

```
$ cat tlist2.txt
# List of tables to extract and their "splitby columns"
src=TRAFODION.MAURIZIO.ORDERS:splitby=O_ORDERKEY
src=TRAFODION.MAURIZIO.CUSTOMER:splitby=C_CUSTOMERKEY
src=TRAFODION.MAURIZIO.PART:splitby=P_PARTKEY
src=TRAFODION.MAURIZIO.LINEITEM:splitby=L_PARTKEY
```

5.2.4. odb Extract

Refer to the Extract Tables section in the Trafodion odb User Guide for complete documentation of this option.

Use then -e option to extract from data a table and write it to standard files or named pipes.

You can:

- Export single tables, list of tables or generic SQL output.
- Export table subsets (columns and/or rows).
- Exports one or multiple tables in parallel using one or multiple data streams for each table
- Invoke other functionalities (trim, remote trim, cast, limit text col length, max rows to export,. . .)

You can write the extracted data to:

- Single/multiple files or standard output (pipe).
- gzip compressed files (no external libraries required).
- XML formatted files (no external libraries required).
- Hadoop File System (requires libhdfs).

Other useful features:

- Configurable NULL/EMPTY strings, field/record separators
- Configurable rowset
- Possibility to run generic SQL scripts before/after extracts
- Multi-threaded export

Important Options

Option	Defines
`src`	Source file. If empty, then odb generates sample data.
`fs`	Field separator.
`tgt`	Target table, required.
`parallel`	Number of connections/threads to be used.`
`splitby`	Source column to parallelize extract operation on.
`pwhere`	`where` condition on source

Example

```
$ odb64luo -u user -p xx -d dsn -T 3 \
-e src=TRAFODION.MAURIZIO.LIN%:tgt=$\{DATA}/ext_%t.csv.gz:rows=m10:fs=\|:trim:gzip: \
-e src=TRAFODION.MAURIZIO.REGION:tgt=$\{DATA}/ext_%t.csv.gz:rows=m10:fs=\|:trim:gzip \
-e src=TRAFODION.MAURIZIO.NATION:tgt=$\{DATA}/ext_%t.csv.gz:rows=m10:fs=\|:trim:gzip
```

The example above:

- Extracts tables `REGION`, `NATION`, and all tables starting with `LIN` from the `TRAFODION.MAURIZIO` schema.
- Saves data into files `ext_%t.csv.gz` (`%t` is expanded to the real table name).
- Compresses the output file (gzip) on the fly (uncompressed data never lands to disk).
- Trims text fields.
- Uses a 10 MB IO buffer.
- Uses three threads (ODBC connection) for the extraction process.

Example

Use odb to extract all tables listed in a file.

```
$ cat tlist.txt

# List of tables to extract src=TRAFODION.MAURIZIO.ORDERS
src=TRAFODION.MAURIZIO.CUSTOMER src=TRAFODION.MAURIZIO.PART
src=TRAFODION.MAURIZIO.LINEITEM
```

Extract all these tables by running:

```
$ odb64luo -u user -p xx -d dsn -e src=-tlist.txt:tgt=%t_%d%m:rows=m20:sq=\"
```

The example above:

- Reads the list of source tables from `tlist.txt`.
- Extracts the data into file using the table name in lowercase (`%t`). appending extraction data and time (`_%d%m`) for the target file name.
- Uses a 20MB I/O buffer for each extraction thread.
- Encloses strings with double-quote characters (`sq=\"`).

5.2.5. odb Transform

Refer to the Map Source File Fields to Target Table Columns section in the Trafodion odb User Guide for complete documentation of odb's mapping/transformation capabilities.

odb provides mapping/transformation capabilities though mapfiles. By specifying `map=<mapfile>` load option you can:

- Associate any input file field to any table column

- Skip input file fields
- Generate sequences
- Insert constants
- Transform dates/timestamps formats
- Extract substrings
- Replace input file strings. For example: insert Maurizio Felici when you read MF
- Generate random values
- And much more

A generic mapfile contains:

- **Comments** (line starting with #)
- **Mappings** to link input file fields to the corresponding target table columns.

Mappings use the following syntax:

```
<colname>:<field>[:transformation operator]
```

Example

Suppose you have a target table like this:

```
+------+---------------+----+-------+------------+
|COLUMN|TYPE           |NULL|DEFAULT|INDEX       |
+------+---------------+----+-------+------------+
|ID    |INTEGER SIGNED |NO  |       |mf_pkey 1 U |
|NAME  |CHAR(10)       |YES |       |            |
|AGE   |SMALLINT SIGNED|YES |       |            |
|BDATE |DATE           |YES |       |            |
+------+---------------+----+-------+------------+
```

And an input file like this:

---

uno,00,**51**,due,*Maurizio*,tre,07 Mar 1959, ignore,remaining, fields

uno,00,**46**,due,*Lucia*,tre,13 Oct 1964, ignore, this

uno,00,**34**,due,*Giovanni*,tre,30 Mar 1976

uno,00,**48**,due,*Antonella*,tre,24 Apr 1962 *****

- **Bold text** represents age.
- *Italics text* represents name.
- Underline text represents birth date.

You want to load the marked fields into the appropriate column, generate a unique key for ID and ignore the remaining fields, In addition, you need to convert the date format and replace all occurrences of `Lucia` with `Lucy`.

The following map file accomplishes these goals:

```
$ cat test/load_map/ml1.map +
# Map file to load TRAFODION.MFTEST.FRIENDS from friends.dat
ID:seq:1                  # Inserts into ID column a sequence starting from 1
NAME:4:REPLACE:Lucia:Lucy # Loads field #4 into NAME and replace all occurrences of
Lucia with Lucy
AGE:2                     # Loads field #2 (they start from zero) into AGE
BDATE:6:DCONV:d.b.y       # Loads field #6 into BDATE converting date format from dd
mmm yyyy
```

Load as follows:

```
$ odb64luo -u user -p xx -d dsn \
  -l src=friends.dat:tgt=TRAFODION.MFTEST.FRIENDS:map=ml1.map:fs=,
```

The above example:

- Reads data from `friends.dat` (`src`).
- Writes data to the `TRAFODION.MFTEST.FRIENDS` Trafodion table (`tgt`).
- Uses `ml1.map` to define transformation specifications (`map`).
- Uses comma as a field separator (`fs`).

Chapter 6. Bulk Unload

The UNLOAD statement is a Trafodion extension that exports data from Trafodion tables into an HDFS location that you specify. Refer to the Trafodion SQL Reference Manual for complete documentation.

The extracted data can be either compressed or uncompressed based on what you choose. UNLOAD performs multiple steps based on the options you give; for example:

- If using snapshot scan:
  - Get list of Trafodion tables from the query plan.
  - Create/verify snapshots.
- Purge target location, if specified
- Extract:
  - Copy table data to data files.
  - Non-compressed is straight copy.
  - Compressed means compression takes place while writing data (no extra step)
- Merge Data Files if specified

Example

This example shows how the UNLOAD statement extracts data from a Trafodion table, `TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS`, into an HDFS folder, `/bulkload/customer_demographics`:

```
>>UNLOAD
+>WITH PURGEDATA FROM TARGET
+>MERGE FILE 'merged_customer_demogs.gz' OVERWRITE
+>COMPRESSION GZIP
+>INTO '/bulkload/customer_demographics'
+>SELECT * FROM trafodion.hbase.customer_demographics
+><<+ cardinality 10e10 ,+ cardinality 10e10 >>;
Task: UNLOAD Status: Started
Task: EMPTY TARGET Status: Started
Task: EMPTY TARGET Status: Ended ET: 00:00:00.014
Task: EXTRACT Status: Started
       Rows Processed: 200000
Task: EXTRACT Status: Ended ET: 00:00:04.743 Task: MERGE FILES Status: Started
Task: MERGE FILES Status: Ended ET: 00:00:00.063

--- 200000 row(s) unloaded.
```

Chapter 7. Monitor Progress

7.1. INSERT and UPSERT

For an INSERT statement, rows are written to the HBase table that represents the Trafodion table when the transaction commits. It is more difficult to see query progress here.

7.2. UPSERT USING LOAD

For an UPSERT USING LOAD statement, rows added are visible in the Trafodion table after each `ListOfPut` call succeeds. You can use a `SELECT COUNT(*)` statement to monitor progress. That way, you know how many rows are already in the table when the statement starts executing.

```
SELECT COUNT(*) FROM trafodion.sch.demo ;
```

7.3. LOAD

For LOAD, query progress goes through a few phases, which sometimes overlap:

1. Hive scan.
2. Sort.
3. Create prep HFiles in HDFS bulkload staging directory (`/bulkload` by default).
4. Move HFiles into HBase.

You can monitor progress in step 2, sort, with this shell command:

```
lsof +L1 | grep SCR | wc -l
```

This command returns a count of the number of overflow files for sort. Each file is 2GB in size. You need to have an approximate idea of the volume of data being loaded to know how much more data needs to be sorted. On a cluster, sort is done on all nodes with a pdsh-like utility. Trafodion data volume can also be larger than Hive data volume by a factor of 2 or 3.

In step 3, create prep HFiles, use the following command to monitor the volume of data written out to the staging directory:

```
hadoop fs -dus /bulkload
```

The `hadoop fs` command must be run from one node and does not have to be repeated across the cluster.

If compression and encoding are used, then the size should be similar to the Hive source data volume. There may be some remnant data in the staging directory from previous commands, so we have to take that into account. This step will start only when sort has completed.

Step 4 is usually the shortest and typically does not exceed a few minutes.

Chapter 8. Troubleshoot

8.1. Improving Throughput

8.1.1. Tuplelists or Rowsets

When Tuplelists or Rowsets are used as the data source, performance typically increases with the number of rows in the Tuplelist or Rowset. Performance peaks at some value for the number of rows and remain more or less steady after that. This peak value depends on row size. Typically a value in the range of 100 to few thousand is reasonable.

8.1.2. Native HBase Tables

When native HBase tables are used as the data source, it is important to override the default value for the attribute HBASE_MAX_COLUMN_VALUE_LENGTH (columnwise mode) or HBASE_MAX_COLUMN_INFO_LENGTH (rowwise mode) and set the value to the maximum for the table being used as the source. The default values may be too large.

8.1.3. Hive Tables

When Hive tables are used as the data source, it is important to override the default value for the attribute HIVE_MAX_STRING_LENGTH when the Hive source table has columns of type string. Please set the value to the length of the longest string in the Hive table.

To determine that length, run this query from a Hive shell:

```
SELECT MAX(LENGTH(<col-name>)) FROM <hive-tab-name>;
```

If the query returns a value less than the current HIVE_MAX_STRING_LENGTH, then you need to increase that value and retry. If the query returns a value that is far less than the current HIVE_MAX_STRING_LENGTH, then you can achieve better performance by reducing the value. An approximate value can be used, too. The Trafodion default of 32000 may be too generous in some cases.

8.2. Checking Plan Quality

It is good practice to check the quality of the plan generated by the SQL compiler before executing a data loading statement that may take a long time to complete.

- For INSERT and UPSERT USING LOAD statements, use the EXPLAIN statement, which is described in the Trafodion SQL Reference Manual.
- For the LOAD statement, which is implemented as a utility operator (that is, a collection of secondary SQL statements), use the following SQL statements to see the plan that it uses to add data to the target table:

```
CONTROL QUERY DEFAULT COMP_BOOL_226 'ON' ;
PREPARE s1 FROM LOAD TRANSFORM INTO <target-table> <select-query-used-as-source> ;
EXPLAIN OPTIONS 'f' s1 ;
```

A typical problem with the plan is that the scan is not parallel enough. For Trafodion tables, you can address this issue with the default attribute, PARALLEL_NUM_ESPS. Using this attribute, a Trafodion scan can be parallelized to as many number of SALT partitions that are defined for the table. For Hive source tables, the default attributes, HIVE_NUM_ESPS_PER_DATANODE and HIVE_MIN_BYTES_PER_ESP_PARTITION, can be used to adjust the degree of parallelism.

8.3. UPDATE STATISTICS Times Out During Sampling

Sampling in update statistics is implemented using the HBase Random RowFilter. For very large tables with several billion rows, the sampling ratio required to get a sample of one million rows is very small. This can result in HBase client connection timeout errors since there may be no row returned by a RegionServer for an extended period of time.

You can avoid this problem by:

- Choosing a sampling percentage higher than the default setting of 1 million rows for large tables.

  For example, suppose table T has one billion rows. Use the following UPDATE STATISTICS statement to sample a million rows, or approximately one-tenth of one percent of the total rows:

  ```
  UPDATE STATISTICS FOR TABLE t ON EVERY COLUMN SAMPLE;
  ```

  To sample one percent of the rows, regardless of the table size, you must explicitly state the sampling rate as follows:

  ```
  UPDATE STATISTICS FOR TABLE t ON EVERY COLUMN SAMPLE RANDOM 1 PERCENT;
  ```

- Setting `hbase.rpc.timeout` to a higher value than currently specified in the HBase settings.

8.4. Index Creation Takes Too Long

When creating an index, all rows of the Trafodion table must be scanned and a subset of columns is returned to the client. This can take a while to complete. If there is a Hive table with the same data as the Trafodion table being scanned, then you can specify the default attribute, USE_HIVE_SOURCE. This causes the Hive table to be used as the source creating the index.

The name of the Hive table must use the Trafodion table name as its prefix. For example, if the Trafodion table is TRAFODION.SCH.DEMO, then the Hive table name can be DEMO_SRC. In this case, set the attribute as follows:

```
CONTROL QUERY DEFAULT USE_HIVE_SOURCE '_SRC' ;
CREATE INDEX demo_ix ON sch.demo(name) ;
```

8.5. Large Deletes Take Too Long or Error Out

If a large number of rows is either updated or deleted in a single SQL statement, then it is likely that the statement does not complete successfully.

Deleting or updating more than 10,000 rows with a single statement is not recommended. Instead, a large delete or update should be broken up into multiple statements each affecting less than 10,000*n rows, if possible. n is number of nodes in the cluster.

8.6. Large UPSERT USING LOAD On a Table With Index Errors Out

UPSERT USING LOAD automatically reverts to a transactional UPSERT when used on a table with an index. This causes Trafodion to run into the limitation discusses in Large Deletes Take Too Long or Error Out above: no more than 10,000*n rows (n = number of nodes) can be affected in a single statement.

Workaround: The UPSERT USING LOAD operation can be placed in a LOAD statement as shown below. The LOAD statement disables indexes on the table before the UPSERT USING LOAD starts. Once the UPSERT USING LOAD completes indexes are populated by the LOAD statement.

```
LOAD WITH UPSERT USING LOAD INTO trafodion.sch.demo SELECT * FROM hive.hive.demo;

Task: LOAD             Status: Started    Object: TRAFODION.SCH.DEMO
Task:  DISABLE INDEXE  Status: Started    Object: TRAFODION.SCH.DEMO
Task:  DISABLE INDEXE  Status: Ended      Object: TRAFODION.SCH.DEMO
Task:  UPSERT USING L  Status: Started    Object: TRAFODION.SCH.DEMO
       Rows Processed: 200000
Task:  UPSERT USING L  Status: Ended      ET: 00:01:03.715
Task:  POPULATE INDEX  Status: Started    Object: TRAFODION.SCH.DEMO
Task:  POPULATE INDEX  Status: Ended      ET: 00:03:11.323
```