# Teradata Conversion Tool Documentation

Date: May 29, 2025

Version: 1.0

Author: Rishabh Dinkar

**Purpose:** *Convert Teradata stored procedures to IBM DB2-compatible SQL syntax.*

## **1. Purpose of the Tool**

The SQL Conversion Tool automates the migration of Teradata stored procedures to IBM DB2-compatible SQL syntax.   
It processes Teradata SQL files, applies transformations to align with DB2 syntax, and generates converted SQL files along with a conversion report.

***Key Objectives:-***

**Syntax Conversion:** Transform Teradata-specific constructs to DB2 equivalents.

**Error Code Mapping:** Convert Teradata error codes to DB2 SQLSTATE values.

**Cursor Handling:** Standardize cursor loops using a done variable.

**Preserve String Literals:** Protect string literals during transformations.

**Pattern-Based Transformations:** Use sp\_patterns.cfg for extensible replacements.

**Reporting:** Highlight warnings and lines requiring manual review.

***Use Case***

**Users:** Database developers and migration specialists.

**Scenario:** Migrating Teradata databases to DB2.

**Example:** Convert TD\_input.sql to 1.out.sql

## **2. Requirements**

**Functional Requirements:**

**Input:** Teradata SQL file (e.g., TD\_input.sql).

**Output:** DB2-compatible SQL file and conversion report.

**Transformations:**

* Replace keywords (e.g., SQL SECURITY INVOKER).
* Map error codes (e.g., SQLCODE -5522 to SQLSTATE 42704).
* Handle cursors with done variable.
* Normalize INTERVAL expressions.
* Remove comments and normalize whitespace.

**Extensibility:** Use sp\_patterns.cfg for patterns.

**Error Handling:** Validate paths and report errors.

**Reporting:** Identify unconverted lines and warnings.

**Non-Functional Requirements:**

**Platform:** Python 3.x (macOS compatible).

**Dependencies:** Standard libraries (re, os, sys, datetime, uuid).

**Encoding:** UTF-8.

**Performance:** Process <10 MB files in seconds.

**Maintainability:** Modular and commented code.

**Usability:** Simple CLI (python main.py input.sql output.sql).

**Constraints:**

* Limited to stored procedures.
* Requires sp\_patterns.cfg in patterns/.
* Manual review for complex constructs.

## **3. Tool Structure**

**Files:**

* **main.py:** Entry point, handles I/O and preprocessing.
* **sp\_converter.py:** Core conversion logic.
* **sp\_patterns.cfg:** Regex patterns for transformations.

**Directory:**

/V6/

├── main.py

├── sp\_converter.py

├── patterns/

│ └── sp\_patterns.cfg

├── Input\_TD.sql

├── Output\_Db2.out\_report.sql

**Workflow:**

1. Run python main.py TD\_input.sql 1.out.sql.

2. Validate paths.

3. Preprocess: Protect literals, remove comments, normalize whitespace.

4. Convert: Apply patterns, handle cursors/errors.

5. Output: Write converted SQL and report.

## **4. File Documentation**

**4.1 main.py**

**Purpose:** Orchestrates conversion, handles I/O, preprocessing, and reporting.

**Imports:**

sys, os, re, datetime

sp\_converter: SPConverter, \_protect\_string\_literals, \_restore\_protected\_strings

**Variables:**

NEWLINE\_TOKEN = "\_\_NEWLINE\_\_": Preserves newlines.

**Functions:**

***strip\_comments(sql: str) ->*** str: Removes comments, preserves literals.

***replace\_newlines(sql: str) ->*** str: Replaces newlines with token.

***restore\_newlines(sql: str) ->*** str: Restores newlines.

***clean\_whitespace(sql: str) ->*** str: Normalizes whitespace.

***print\_usage():*** Displays CLI usage.

***validate\_file\_paths(input\_path, output\_path, pattern\_path):*** Validates paths.

***convert\_sp(input\_path, output\_path, pattern\_path):*** Processes SQL.

***generate\_conversion\_report(converter: SPConverter, output\_path: str) ->*** str: Creates report.

***main():*** Entry point, drives execution.

***Usage:*** *python main.py input.sql output.sql*

**4.2 sp\_converter.py**

**Purpose:** Implements conversion logic.

**Imports:**

*re, uuid*

**Functions:**

\_protect\_string\_literals(code: str) -> tuple[str, dict]: Protects literals.

\_restore\_protected\_strings(code: str, protected: dict) -> str: Restores literals.

remove\_sql\_security\_invoker(sql: str) -> str: Removes SQL SECURITY INVOKER.

replace\_first\_begin(sql\_text): Adds LANGUAGE SQL\nBEGIN ATOMIC.

**Class:** SPConverter:

**Attributes:** SQL, patterns, warnings, error mappings.

**Methods:**

***load\_patterns:*** Loads sp\_patterns.cfg.

***mask\_string\_literals:*** Protects literals for patterns.

***unmask\_string\_literals:*** Restores literals.

***apply\_patterns:*** Applies regex patterns.

***add\_done\_handling:*** Adds cursor handling.

***convert\_teradata\_error\_codes:*** Maps error codes.

***convert\_error\_codes:*** Converts SQLCODE to SQLSTATE.

***inject\_continue\_handler\_once:*** Adds cursor handler.

***convert\_exception\_handlers:*** Converts exception blocks.

***inject\_default\_exception\_handler:*** Adds default handler.

***inject\_cursor\_handlers:*** Ensures cursor handling.

***convert\_cursor\_checks:*** Standardizes cursor checks.

***convert:*** Orchestrates transformations.

***get\_unconverted\_lines:*** Returns unconverted lines.

***get\_conversion\_warnings:*** Returns warnings.

**4.3 sp\_patterns.cfg**

**Purpose:** Defines regex patterns for transformations.

**Format:** SECTION|PATTERN|REPLACEMENT

**Key Patterns:**

**SP\_HEAD:** Procedure headers (e.g., CREATE PROCEDURE).

**SP\_BODY:** Error codes, cursors, functions, intervals, etc.

**Usage:** Extend by adding patterns.

## **5. How to Use**

1. Prepare TD\_input.sql.

2. **Run:** python main.py TD\_input.sql 1.out.sql.

3. **Check** 1.out.sql and 1.out\_report.sql.

4. Test in db2 enviroment.

5. Extend patterns or functions as needed.

## **6. Maintenance**

**Add Patterns:** Edit sp\_patterns.cfg.

**New Functions:** Add to sp\_converter.py.

**Test:** Use unittest in test\_converter.py.

## **7. Conclusion**

The tool efficiently migrates Teradata stored procedures to DB2, with a modular and extensible design. For issues, share input/output files.