Apply SQL Transformation

Updated: August 7, 2015

Runs a SQLite query on input datasets to transform the data

Category: Data Transformation / Manipulation (https://msdn.microsoft.com/en-us/library/azure/dn905863)

Module Overview

You can use the **Apply SQL Transformation** module to specify a SQL query, using the SQLite syntax, to work with values in the input datasets.

SQL statements are handy when you need to modify your data or persist the data for use in other environments. For example, using **Apply SQL Transformation**, you can:

- Create tables for results and save the datasets in a portable database.
- Perform custom transformations on data types, or create aggregates.
- Execute SQL query statements to filter or alter data and return the query results as a data table.

What is SQLite?

SQLite is a public domain relational database management system that is contained in a C programming library. SQLite is a popular choice as an embedded database for local storage in web browsers.

SQLite was originally designed in 2000 for the U.S. Navy, to support serverless transactions. It is a self-contained database engine that has no management system and hence requires no configuration or administration.

How to Configure Apply SQL Transformation

The module can take up to three datasets as inputs. When you reference the datasets connected to each input port, use the names t1, t2, and t3. The table number indicates the index of the input port.

The remaining parameter is a SQL query, which uses the SQLite syntax. This module supports all standard statements of the SQLite syntax.

For a list of unsupported statements, see the Technical Notes section.

General Syntax and Usage

- When typing multiple lines in the SQL Script text box, use a semi-colon to terminate the statement. Otherwise, line breaks are converted to spaces.

  For example, the following statements are equivalent:

```
SELECT
*
from
t1;
```

```
SELECT * from t1;
```

- If a column name duplicates the name of a reserved keyword, syntax highlighting will be applied.

  For example, in the following query on the Blood Donation dataset, Time is a valid column name and is also a reserved keyword. In some cases, the query will return the correct results, but to avoid confusion, you should enclose column names with square brackets or double quotation marks.

```
SELECT Recency, Frequency, Monetary, Time, Class
FROM t1
WHERE Time between 3 and 20;
```

  Note that syntax highlighting remains on the keyword even after it is enclosed in quotes or brackets.

- SQLite is case insensitive, except for a few commands that have case-sensitive variants with different meanings (GLOB vs. glob).

- You can add comments by using either `--` at the beginning of each line, or by enclosing text using `/* */`.

  For example, this statement is valid:

```
SELECT * from t1
/*WHERE ItemID BETWEEN 1 AND 100*/;
```

SELECT Statement

In the SELECT statement, column names that include spaces or other characters prohibited in identifiers must be enclosed in double quotation marks or in square brackets. This query uses the Two-Class Iris dataset on `t1`.

```
SELECT class, "sepal-length" FROM t1;
```

This example demonstrates how to filter values in the dataset using a WHERE clause.

```
SELECT class, "sepal-length" FROM t1 WHERE "sepal-length" >5.0;
```

Identifiers that include spaces, hyphens, etc. must be enclosed in brackets or quotation marks.

```
SELECT class, "sepal-length" FROM t1 WHERE 'sepal-length' >5.0;
```

```
SELECT class, "sepal-length" FROM t1 WHERE [sepal-length] >5.0;
```

Joins

The following examples use the Restaurant Ratings dataset on the input port corresponding to `t1`, and the Restaurant Features dataset on the input port corresponding to `t2`.

The following statement joins the two tables to create a dataset that combines the specified restaurant features with average ratings for each restaurant.

```
SELECT DISTINCT(t2.placeid),
t2.name, t2.city, t2.state, t2.price, t2.alcohol,
AVG(rating)  AS 'AvgRating'
FROM t1
JOIN t2
ON t1.placeID = t2.placeID
GROUP BY t2.placeid;
```

Aggregate Functions

The following query returns a dataset containing the restaurant ID, along with the average rating for the restaurant.

```
SELECT DISTINCT placeid,
AVG(rating) AS ‘AvgRating’,
FROM t1
GROUP BY placeid
```

Aggregate functions currently supported are: **AVG**, **COUNT**, **MAX**, **MIN**, **SUM**, **TOTAL**.

Working with Strings

SQLite supports the double pipe operator for concatenating strings.

The following statement creates a new column by concatenating two text columns.

```
SELECT placeID, name,
(city || '-' || state) AS 'Target Region',
FROM t1
```

Warning

The Transact-SQL string concatenation operator is not supported: + (String Concatenation) (https://msdn.microsoft.com/en-us/library/ms177561.aspx),

For example, the expression `('city + '-' + state) AS 'Target Region'` in the example query would return 0 for all values.

However, even though the operator is not supported for this data type, no error is raised in Azure Machine Learning. Be sure to verify the results of **Apply SQL Transformation** before using the resulting dataset in an experiment.

COALESCE and CASE

COALESCE evaluates multiple arguments in order and returns the value of the first expression that does not evaluate to NULL. For example, this query on the Steel Annealing Multi-Class dataset returns the first non-null flag from a list of columns assumed to have mutually exclusive values. If no flag is found, the string "none" is returned.

```
SELECT classes, family, [product-type],
COALESCE(bt,bc,bf,[bw/me],bl, "none") AS TemperType
FROM t1;
```

The CASE statement is useful for testing values and returning a new value based on the evaluated results. SQLite supports the following syntax for CASE statements:

- CASE WHEN [condition] THEN [expression] ELSE [expression] END
- CASE [expression] WHEN [value] THEN [expression] ELSE [expression] END

For example, suppose you had previously used the Indicator Values (https://msdn.microsoft.com/en-us/library/azure/dn905831) module to create a set feature columns containing true-false values. The following query collapses the values in multiple feature columns into a single multivalued column.

```
SELECT userID, [smoker-0], [smoker-1],
CASE
WHEN [smoker-0]= '1' THEN 'smoker'
WHEN [smoker-1]= '1' THEN 'nonsmoker'
ELSE 'unknown'
END AS newLabel
FROM t1;
```

Examples

For an example of how this module might be used in machine learning experiments, see this sample in the Model Gallery (http://gallery.azureml.net/):

- The Apply SQL Transformation (http://go.microsoft.com/fwlink/?LinkId=525947) sample uses the Restaurant Ratings, Restaurant Features, and Restaurant Customers dataset to illustrate simple joins, select statements, and aggregate functions.

Technical Notes

- An input is required on port 1.
- If the input dataset has column names, the columns in the output dataset will use the column names from the input dataset.
- If the input dataset does not have column names, the column names in the table are automatically created by using the following naming convention: T1COL1, T1COL2, T1COL3, and so on, where the numbers indicate the index of each column in the input dataset.
- For column names that contain a space or other special characters, use square brackets or double quotation marks when referring to the columns in the **SELECT** or **WHERE** clauses.

Unsupported Statements

Although SQLite supports much of the ANSI SQL standard, it does not include many features supported by commercial relational database systems.

Also, be aware of the following restrictions when creating SQL statements:

- SQLite uses dynamic typing for values, rather than assigning a type to a column as in most relational database systems. It is weakly typed, and allows implicit type conversion.
- LEFT OUTER JOIN is implemented, but not RIGHT OUTER JOIN or FULL OUTER JOIN.

For more information, see SQL as Understood by SQLite (http://www.sqlite.org/lang.html).

- You can use **RENAME TABLE** and **ADD COLUMN** statements with the **ALTER TABLE** command, but other clauses are not supported, including **DROP COLUMN**, **ALTER COLUMN**, and **ADD CONSTRAINT**.
- You can create a VIEW within SQLite, but thereafter views are read-only. You cannot execute a **DELETE**, **INSERT**, or **UPDATE** statement on a view. However, you can create a trigger that fires on an attempt to **DELETE**, **INSERT**, or **UPDATE** on a view and perform other operations in the body of the trigger.

In addition to the list of non-supported functions provided on the official SQLite site, the following wiki provides a list of other unsupported features: SQLite - Unsupported SQL (http://www2.sqlite.org/cvstrac/wiki?p=UnsupportedSql)

Expected Inputs

Name	Type	Description
Table1	Data Table (https://msdn.microsoft.com/en-us/library/azure/dn905851)	Input dataset1
Table2	Data Table (https://msdn.microsoft.com/en-us/library/azure/dn905851)	Input dataset2
Table3	Data Table (https://msdn.microsoft.com/en-us/library/azure/dn905851)	Input dataset3

Module Parameter

Name	Range	Type	Default	Description
SQL Query Script	any	StreamReader		SQL query statement

Output

Name	Type	Description
Results dataset	Data Table (https://msdn.microsoft.com/en-us/library/azure/dn905851)	Output dataset

Exceptions

For a list of all exceptions, see Machine Learning Module Error Codes (https://msdn.microsoft.com/en-us/library/azure/dn905910).

Exception	Description
Error 0001 (https://msdn.microsoft.com/en-us/library/azure/dn905993)	An exception occurs if one or more specified columns of the dataset couldn't be found.
	An exception occurs if one or more of the input datasets is null or empty.

Error 0003 (https://msdn.microsoft.com/en-us/library/azure/dn906003)	

See Also

Data Transformation / Manipulation (https://msdn.microsoft.com/en-us/library/azure/dn905863)
Data Transformation (https://msdn.microsoft.com/en-us/library/azure/dn905834)
A-Z List of Machine Learning Studio Modules (https://msdn.microsoft.com/en-us/library/azure/dn906033)