Introduction to Scope

2017年8月4日 14:24

Scope is the query language for cosmos
Scope is not SQL, but it's syntax is very similar to SQL

run Scope script

1. install Scope studio (or scope for vs)
2. Write Scope script
3. Have any parameters?
4. Run on local or on Cosmos

Scope.script

	Input	process	output
Structured stream	SSTREAM "*.ss"		OUTPUT [Rowset name] TO "*.ss"
Unstructured stream	EXTRACT Name: string, Age: int FROM "*.hyp" USING DefaultTextExtractor	SELECT FROM [WHERE] [HAVING] [GROUP BY] [ORDER BY]	OUTPUT TO "*.txt" USING DefaultTextOutputter

DataType: 16

bool	Long
Byte[]	Ulong
Binary(an alias for byte[])	Double
byte	decimal
sbyte	String
Char	Short
Guid	ushort
Unit	User_defined Types

Run Scope

Run locally (don't include data on the server)	Run on cosmos
Stream names are not case sensitive	Case sensitive
Can use '\' and '/'	Only "/" works, "\" doesn't work
Copy the data on server to local machine File.copy(serverData, localData)	Data on cosmos, server and local are supported

Proposals:

1. Always care about case
2. Only use "/" and drop out "\"

Process

★ Selection
SELECT: pick the columns of interest

- Filtering

Where	HAVING
filter on input rows	filter on output rows

- https://stackoverflow.com/questions/9253244/sql-having-vs-where

Selection

1. Simple calculations : you must assign that column a name via the AS keyword

```
Rs1 =
    SELECT Market, DwellTime + 1.0 AS DwellTime2
    FROM searchlog;
```

2. Type Casting: Expression can be converted to a different type

```
Rs1 =
    SELECT Market, ((double) DwellTime) AS DwellTimeDouble
    FROM searchlog;
```

3. Call a C# method - a User Defined Function(#CS Blocks)

```
Rs1 =
    SELECT Market, MyHelper.SecondsToMinutes(DwellTime) AS
    DwellTimeMinutes
    FROM searchlog;
```

4. The Conditional Operator and Ternary IF
   - (cond ? a: b)

```
Rs1 =
    SELECT Market, (DwellTime > 300  ? "long" : "short" AS DwellType
    FROM seachlog;
```

   - IF (<cond>, <a>,<b>)

```
Rs1 =
    SELECT Market, IF (DwellTime>300, "long", "short") AS DwellType
    FROM searchlog;
```

5. Getting the TOP N Rows

```
Rs1 =
    SELECT TOP 5 Market, DwellTime
    FROM seachlog;
```

6. Finding Distinct Values with DISTINCT

```
RS1 =
    SELECT DISTINCT Market
    FROM searchlog;
```

7. Sorting: ORDER BY + ASC/DESC

```
Rs1 =
    SELECT Start, Markte, DwellTime
    FROM seachlog
    ORDER BY DwellTime ASC;
```

8. Numbering Rows

```
Rs1 =
    SELECT RANK AS RowNumber, Start, Market
    FROM seachlog
    ORDER BY Start;
```

Filtering

- WHERE operations on input rows and HAVING on output rows

```
Rs1 =
    SELECT Start, Market, DwellTime/60.0 AS DwellTimeMinutes
    FROM searchlog
    HAVING DwellTimeInMinutes >= 20;
```

Use a new rowset to achieve the same effect

```
Rs1 =
    SELECT Start, Market, DwellTime, DwellTime/60.0 AS DwellTimeMinutes
    FROM searchlog;
Rs2 =
    SELECT *
    FROM rs1
    WHERE DwellTimeMinutes >=20;
```

- ANY/OR versus ALL/ANY

ALL is conceptually equivalent to (e1 AND e2 AND … AND en)
ANY is conceptually equivalent to (e1 OR e2 OR … OR en)

```
Rs1 =
    SELECT Market, Query
    FROM seachlog
    WHERE (Query.Length  != 0) AND (Query.Substring(1) == "bing");
Rs2 =
    SELECT Market, Query
    FROM searchlog
    WHERE ALL (Query.Length != 0, Query.Substring(1) == "bing");
```

Proposals:
Only use ALL or ANY if you MUST gurantee that the expressions must be evaluated in a certain order and use short-circuiting.

Grouping and Aggregation

- Aggregation command

ARGMAX	AVG	COUNT	COUNTIF
FIRST	LAST	LIST	STDE
MAX	MIN	SUM	VAR

```
Rs2 =
    SELECT Department, ARGMAX(Tenure, LastName) AS MostTentured
    GROUP BY Department
    FROM rs0;
```

ARGMAX(a,b) = Find the row with the maximum value for column a, from that row return the value for b

- Use HAVING to select our interested aggregated value.

- Merging Rows
  - Merging Rowsets with Set options (UNION/UNION ALL)
  - Finding Common Rows (INTEREST/INTERSET ALL)
  - Finding Rows That Are NOT in The Other Table (EXCEPT)
  - IN operator

```
Rs =
    SELECT FirstName, LastName, JobTitle
    FROM People
```

WHERE JobTitle IN ("Design Engineer", "Tool Designer", "Marketing Assistant");

A

ID	Name
1	Smith
1	Smith
2	Brown
3	Case

B

ID	Name
1	Smith
1	Smith
1	Smith
2	Brown
4	Dey
4	Dey

UNION DISTINCT

ID	Name
1	Smith
2	Brown
3	Case
4	Dey

UNION ALL

ID	Name
1	Smith
2	Smith
2	Brown
3	Case
1	Smith
1	Smith
2	Smith
4	Dey
4	Dey

INTERSECT ALL

ID	Name
1	Smith
1	Smith
2	Brown

INTERSECT DISTINCT

ID	Name
1	Smith
2	Brown

EXCEPT ALL(A,B)

ID	Name
3	Case

EXCEPT DISTINCT(A,B)

ID	Name
3	Case

EXCEPT ALL (B, A)

ID	Name
1	Smith
4	Dey
4	Dey

EXCEPT DISTINCT(B, A)

ID	Name
4	Dey

- Looking up data from other rowsets
  - INNER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
  - CROSS JOIN (an INNER JOIN without a join condition)
  - LEFT SEMIJOIN and RIGHT SEMIJOIN

A LEET SEMIJOIN is more like a filter than a join. It is the syntactical way of expressing

```
Select <columns from left>
FROM left
WHERE left.JoinKey IN <right.JoinKeys>;
```

JOIN

EMPLOYEES

DepID	EmpName
D1	Robinson
D3	John

DEPARTMENTS

DepID	DepName
D1	Clerical
D2	Engineering

CORSS JOIN

D1	Robinson	D1	Clerical
D1	Robinson	D2	Engineering
D3	John	D1	Clerical
D3	John	D2	Engineering

EMPLOYEES

DepID	EmpName
D1	Robinson
D1	Smith
D1	John

DEPARTMENTS

DepID	DepName
D1	Clerical
D1	Admin
D2	Engineering

INNER JOIN

D1	Robinson	D1	Clerical
D1	Robinson	D1	Admin
D1	Smith	D1	Clerical
D1	Smith	D1	Admin

LEFT OUTER JOIN

D1	Robinson	D1	Clerical
D1	Robinson	D1	Admin
D1	Smith	D1	Clerical
D1	Smith	D1	Admin
D3	John	NULL	NULL

INNER OUT JOIN

D1	Robinson	D1	Clerical
D1	Robinson	D1	Admin
D1	Smith	D1	Clerical
D1	Smith	D1	Admin
NULL	NULL	D2	Engineering

FULL OUTER JOIN

D1	Robinson	D1	Clerical
D1	Robinson	D1	Admin
D1	Smith	D1	Clerical
D1	Smith	D1	Admin
D3	John	NULL	NULL
NULL	NULL	D2	Engineering

Parameters

- Preprocessor parameters (New-Style Parameters)
  - #DECLARE
  - @foo
  - #DECLARE str1 string = "Hello World";
  - #DECLARE str2 string = "BEGIN" + @str1 + "END";

- #DECLARE str3 string = string.format("BEGIN{0}END", @str1) ;
- #DECLARE date0 DateTime = DateTime.Parse("2010/03/31");

- Script parameters (Old-Style Parameters)
  - @@foo@@
  - Provided by the user when the Script is run.

- Scalar Parameters in Views/Functions/Procedures
  - @foo