GraphScript: Implementing Complex Graph Algorithms in SAP HANA

Marcus Paradies, Cornelia Kinder, Jan Bross, Thomas Fischer, Romans Kasperovics and Hinnerk Gildhoff, SAP SE

September 1, 2017
SAP HANA Overview

<table>
<thead>
<tr>
<th>APPLICATION SERVICES</th>
<th>PROCESSING SERVICES</th>
<th>INTEGRATION &amp; QUALITY SERVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Server</td>
<td>Spatial</td>
<td>Data Virtualization</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Text Analytics</td>
<td>ELT &amp; Replication</td>
</tr>
<tr>
<td>Fiori UX</td>
<td>Streaming Analytics</td>
<td></td>
</tr>
<tr>
<td>Graphic Modeler</td>
<td>Predictive</td>
<td></td>
</tr>
<tr>
<td>Application Lifecycle Management</td>
<td>Search</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DATABASE SERVICES

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Columnar OLTP+OLAP</td>
<td>Multi-core &amp; Parallelization</td>
<td>Advanced Compression</td>
</tr>
<tr>
<td>Multi-tenant</td>
<td>Multi-tier Storage</td>
<td>Data Modeling</td>
</tr>
<tr>
<td>Advanced Compression</td>
<td>Openness</td>
<td>Admin &amp; Security</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High Availability &amp; Disaster Recovery</td>
</tr>
</tbody>
</table>

- Offers advanced analytics features for graph, text, geospatial, and machine learning directly on business data
Graph Querying Paradigms in SAP HANA

Graph Pattern Matching

"Retrieve all suppliers of Company D"

openCypher*

Language Interface

Graph Analysis

"Compute all communities in the graph"

GraphScript

* Limited subset of language spec
## Design Principles

<table>
<thead>
<tr>
<th>Expressiveness &amp; Simplicity</th>
<th>Minimality &amp; Orthogonality</th>
<th>Native Graph Abstraction</th>
<th>Tight Integration</th>
<th>High Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy-to-use for graph algorithm implementers</td>
<td>Limited but effective set of types and operations thereon</td>
<td>Native exposure of graph-specific types</td>
<td>Pushdown of operations to relational store</td>
<td>Desired performance close to hand-written code</td>
</tr>
<tr>
<td>Support for a large variety of graph algorithm classes and workflows</td>
<td>Extensibility of built-in graph operators</td>
<td>Full exposure of graph data model</td>
<td>Reuse of dependency management</td>
<td>Explicit parallelization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Relational only for returning complex results</td>
<td>Reuse of resource management</td>
<td>Effective Program Rewritings</td>
</tr>
</tbody>
</table>
GraphScript Type System

Graph Types
- Path
- Edge
- Vertex

Container Types
- Bag<T>
- Sequence<T>
- Table<T…>

SQL Scalar Types
- Decimal
- Text
- Int
- Timestamp
- ST_Point
Graph Data Exposure in GraphScript

Vertex Table

```
CREATE COLUMN TABLE MYSCHEMA.VERTICES (  
   ID VARCHAR(100) PRIMARY KEY,  
   TYPE VARCHAR(100),  
   NAME VARCHAR(100),  
   TITLE VARCHAR(100)  
);
```

Edge Table

```
CREATE COLUMN TABLE MYSCHEMA.EDGES (  
   ID INTEGER PRIMARY KEY,  
   SRC VARCHAR(100) NOT NULL  
     REFERENCES MYSCHEMA.VERTECS (ID)  
   TRGT VARCHAR(100) NOT NULL  
     REFERENCES MYSCHEMA.VERTECS (ID)  
   TYPE VARCHAR(50)  
);
```

CREATE GRAPH WORKSPACE MYSCHEMA.MY_GRAPH  
   EDGE TABLE MYSCHEMA.EDGES  
   SOURCE COLUMN SRC  
   TARGET COLUMN TRGT  
   KEY COLUMN ID  
   VERTEX TABLE MYSCHEMA.VERTECS  
   KEY COLUMN ID;

Graph Workspace Metadata Object
Graph Data Exposure in GraphScript /2

Vertex Table View

CREATE VIEW MYSCHEMA.VERTEX_VIEW AS
SELECT * FROM MYSCHEMA.VERTEICES
WHERE TYPE = 'Person';

Create View MySchema. Vertex View As
Select * From MySchema.Vertices
Where Type = 'Person';

Edge Table View

CREATE VIEW MYSCHEMA.EDGE_VIEW AS
SELECT * FROM MYSCHEMA.EDGES
WHERE TYPE = 'knows';

Create View MySchema. Edge View As
Select * From MySchema.Edges
Where Type = 'knows';

Graph Workspace Metadata Object

CREATE GRAPH WORKSPACE MYSCHEMA.MY_SUBGRAPH
EDGE TABLE MYSCHEMA.EDGE_VIEW
SOURCE COLUMN SRC
TARGET COLUMN TRGT
KEY COLUMN ID
VERTEX TABLE MYSCHEMA.VERTEX_VIEW
KEY COLUMN ID;

Create Graph Workspace MySchema.My_Subgraph
Edge Table MySchema.Edge_View
Source Column Src
Target Column Trgt
Key Column Id
Vertex Table MySchema.Vertex_View
Key Column Id;
A Simple GraphScript Example

```
CREATE PROCEDURE "myGraphProc"(OUT numNeighbors BIGINT)
LANGUAGE GRAPH READS SQL DATA AS
BEGIN
  Graph g = Graph("mySchema","myGraph");
  ALTER g ADD TEMPORARY VERTEX ATTRIBUTE(BIGINT cnt = 0);
  FOREACH v IN Vertices(:g) {
    v.cnt = Count(Neighbors(:g, :v, 1, 3));
  }
  FOREACH v IN Vertices(:g) {
    numNeighbors += :v.cnt;
  }
END
```
### Inducing Subgraphs

#### "Induce a graph over all blue edges"

```
Graph g = Subgraph(:g, e IN Edges(:g)
    WHERE :e.color == 'blue');
```

#### "Induce a graph over all red edges that connect a green and a yellow vertex"

```
Graph g = Subgraph(:g, e IN Edges(:g) WHERE
    Source(:e).color == 'green' AND
    Target(:e).color == 'yellow' AND :e.color == 'red');
```

#### "Induce a graph overall all vertices that are reachable from vertex 4"

```
vertex v1 = Vertex(:g, 4);
Graph g = Subgraph(:g, v IN Vertices(:g)
    WHERE IS_REACHABLE(:g, :v1, :v);
```
Integration with other Data Models/Scalar Types

Creation of Relational Output from GraphScript

```
Graph g = Graph("myWorkspace");
ALTER g ADD TEMPORARY VERTEX ATTRIBUTE(DOUBLE length = 0);
FOREACH v IN Vertices(:g) {
    Path p = Shortest_Path(:g, :v, Vertex(:g, 1));
    v.length = Length(:p);
}
outTab = SELECT :v.id, :v.length FOREACH v IN Vertices(:g);
```

Integration with Geospatial Processing

```
Graph g = Graph("myWorkspace");
ST_Geometry area = Vertex(:g, 'Munich').area;
Graph g1 = Subgraph(:g, v IN Vertices(:g) WHERE :v.type == 'Person'
                        AND ST_Within(:v.location , :area));
```
Conclusion

Language Constructs

• Rich type system with native graph types
• Powerful imperative constructs

Code Generation

• Generation of low-level code against internal Graph Storage interface
• Elimination of query processing on external vertex/edge keys
• Pushdown of filter conditions to relational engine

Future Work

• More language extensions towards fast traversals and user-defined function invocations
• More advanced GraphScript program rewritings and optimizations
Thank you.

Contact information:

Marcus Paradies
marcus.paradies@sap.com