*Sparksee

Slavomír Krupa, Oliver Mrázik, Martin Strhársky

PA195

* It’s a trap

Introduction

- Graph database
- Written in C++
- Labeled Directed Attributed Multigraph
- Double licensing - free for development, research and personal use
- Supports multiple operating systems (Windows, Mac, Linux, iOS, Android, BB)
- Provides API for multiple programming languages (Java, C#, C++, Python, Objective-C)
- DB is standalone file
- Supports full ACID

Installation and configuration

- Installation - embedable database - library for your desired language
- SparkseeConfig class vs properties file

```
sparksee.license=Your-license-key
sparksee.io.cache.maxsize=2048
sparksee.io.rollback=true
sparksee.log.file=HelloSparksee.log
```

- After start just cache size change possible

http://sparsity-technologies.com/UserManual/Configuration.html#variables

Database construction

```
// loads 'sparksee.cfg' from working directory
SparkseeConfig cfg = new SparkseeConfig();

// creates a database factory allowing to create/load db
Sparksee sparksee = new Sparksee(cfg);

try (Database db = sparksee.create("HelloSparksee.gdb", "HelloSparksee"); ) {
    ...
    //  code using db
    ...
   }

sparksee.close();  // or use try with resources in java 7
```

Sessions and transactions

- full ACID
- temporary data is associated to the session
- session should be exclusive for a thread
- begin, commit and rollback
  - or auto-commit
- read only vs. write transaction

```
Session sess = db.newSession();
Graph g = sess.getGraph();
```

Schema

- **Graph database** - set of nodes and edges
- Nodes and edges belongs to a type
- Types are mandatory in order to create new objects
- Every object has unique and immutable OID

DB schema interaction

- Entities are represented as node types

```
int movieType = g.newNodeType("MOVIE");
int peopleType = g.newNodeType("PEOPLE");
```

- Graph#removeType
  - removes type from schema
- Graph#findType
  - discover if type exists
- Graph#findTypes
  - get all existing types
- Graph#getType
  - get specific type

Creating attributes

- We can enrich information about entities by using attributes

```
int movieIdType = g.newAttribute(movieType, "ID", DataType.Long, AttributeKind.Unique);
int movieTitleType = g.newAttribute(movieType, "TITLE", DataType.String, AttributeKind.Indexed);
int movieYearType = g.newAttribute(movieType, "YEAR", DataType.Integer, AttributeKind.Indexed);
int movieOtherType = g.newAttribute(movieType, "OTHER", DataType.String, AttributeKind.Basic);
```

- Three types of attributes
  - cannot be used for query operations
    - Basic
  - can be used for query operations
    - Unique
    - Indexed

Attribute datatypes and scopes

- Datatypes
  - Boolean, Integer, Long, Double, Timestamp, String, Text and OID

- Scopes
  - type specific
  - node specific
  - edge specific
  - global
- Session attribute

```
g.newAttribute(Type.NodesType, "OTHER", DataType.String, AttributeKind.Basic);
g.newAttribute(Type.EdgesType, "OTHER", DataType.String, AttributeKind.Basic);
g.newAttribute(Type.GlobalType, "OTHER", DataType.String, AttributeKind.Basic);



g.newSessionAttribute(movieType, "OTHER", DataType.String, AttributeKind.
Basic);
```

Creating relationships

- directed vs. undirected
- restricted vs. unrestricted
- edges can have attributes

```
int castType = g.newEdgeType("CAST", false, false);


int directsType = g.newRestrictedEdgeType("DIRECTS", peopleType, movieType,
false);
```

Adding data

- Value class can be reused for setting all attributes

```
Value value = new Value();

long mLostInTranslation = g.newNode(movieType);
g.setAttribute(mLostInTranslation, movieIdType, valuesetLong(1));
g.setAttribute(mLostInTranslation, movieTitleType, valuesetString("Lost in
Translation"));
g.setAttribute(mLostInTranslation, movieYearType, valuesetInteger(2003));


long anEdge;
anEdge = g.newEdge(castType, mLostInTranslation, pScarlett);
g.setAttribute(anEdge, castCharacterType, valuesetString("Charlotte"));
```

Querying DB

- returns “Objects” - special object
  - basically set of ids
  - with iterator
  - Possible operations - Union, Intersection, Difference
- querying methods
  - Graph#select(type)
  - Graph#select(attribute, condition, value)
    - standard conditions supported - including like (case sensitive), RegExp
  - Graph#findObject(attribute, value)
  - Graph#neighbors(node, edgeType, edgesDirection) - (Graph#explode for edges)

Example querying

```
// retrieve all 'PEOPLE' node objects
Objects peopleObjs1 = graph.select(peopleTypeId);
...
// retrieve Scarlett Johansson from the graph, which is a "PEOPLE" node
Objects peopleObjs2 = graph.select(nameAttrId, Condition.Equal, v.setString("Scarlett
Johansson"));
...
// retrieve all 'PEOPLE' node objects containing "Allen" in the name.
Objects peopleObjs3 = graph.select(nameAttrId, Condition.Like, v.setString("Allen"));
...
```

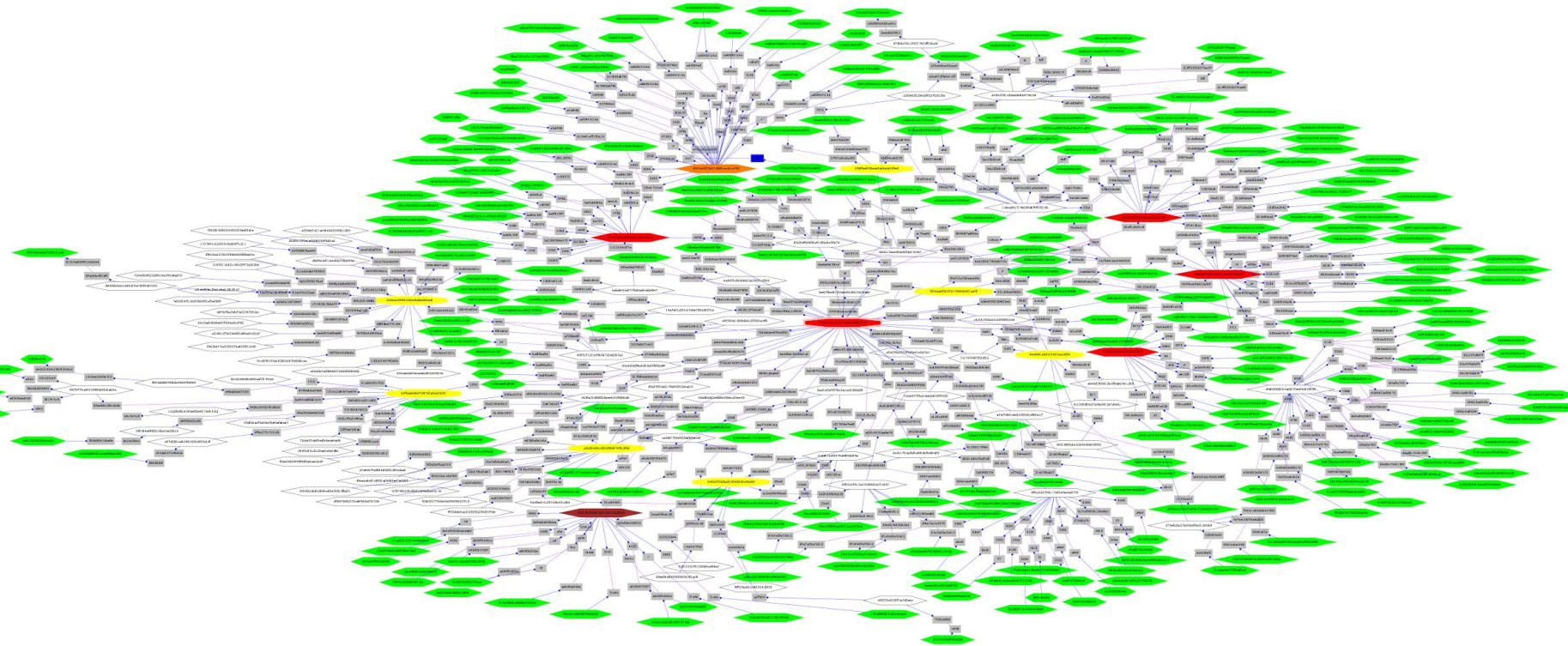
Example querying

```
long node = ... // a person node
int friendTypeId = graph.findType("FRIEND");
int lovesTypeId = graph.findType("LOVES");
...
// retrieve all in-comming LOVES edges
Objects edges = graph.explode(node, lovesTypeId, EdgesDirection.Ingoing);
...
// retrieve all nodes through FRIEND edges - 1 hop
Objects friends = graph.neighbors(node, friendTypeId, EdgesDirection.Any);
// friends of friends (2-hop)
Objects friends2 = graph.neighbors(friends, friendTypeId, EdgesDirection.Any);
```

Import & Export

- export to visual (Graphviz, GraphML, YGraphML)
  - Graph#export("exportName.dot", ExportType.Graphviz, **exportManager**)
  - exportManager - label, label color, color, shape, width, font, font size ... for node/edge, global
    - DefaultExport - default implementation
- .csv format used for export and import in bulk
  - CSVReader (RowReader) class for parsing the .csv file
  - CSVWriter (RowWriter) class for writing the .csv file
  - TypeLoader class for importing readed .csv rows to database (transformed to objects)
  - TypeExporter class for exporting objects from database to .csv file

Example of visual export (Graphviz)



Scripting

- their own scripting language
- used for creating schemas or loading data from CSV

- example of DB schema creation

```
CREATE GDB alias INTO filename
```

Scripting - node creation

```
CREATE NODE type_name "("
    [attribute_name

(INTEGER|LONG|DOUBLE|STRING|BOOLEAN|TIMESTAMP|TEXT)
        [BASIC|INDEXED|UNIQUE]
        [DEFAULT value],
    ...]
")"
create node PEOPLE (
    ID string unique,
    NAME string indexed,
    BIRTH timestamp,
    AGE integer indexed
)
```

```
int peopleType = g.newNodeType("PEOPLE");

int peopleIdType = g.newAttribute(peopleType, "ID",
DataType.Long, AttributeKind.Unique);

int peopleNameType = g.newAttribute(peopleType,
"NAME", DataType.String, AttributeKind.Indexed);

int peopleBirthIdType = g.newAttribute(peopleType,
"BIRTH", DataType.Timestamp, AttributeKind.Basic);

int peopleAgeType = g.newAttribute(peopleType, "AGE",
DataType.Integer, AttributeKind.Indexed);
```

Scripting - edge creation

```
CREATE [UNDIRECTED] EDGE type_name
[FROM tail_node_type_name TO head_node_type_name]"("
    [attribute_name

(INTEGER|LONG|DOUBLE|STRING|BOOLEAN|TIMESTAMP|TEXT)
        [BASIC|INDEXED|UNIQUE]
        [DEFAULT value],
    ...]
") [MATERIALIZE NEIGHBORS]"
create edge KNOWS
from PEOPLE to PEOPLE (
    SINCE timestamp indexed,
    WHERE string indexed
) materialize neighbors
```

```
int knowsType = g.newEdgeType("KNOWS", false,
true);

int knowsSinceType = g.newAttribute(knowsType,
"SINCE", DataType.Timestamp, AttributeKind.
Indexed);

int knowsWhereType = g.newAttribute(knowsType,
"WHERE", DataType.String, AttributeKind.
Indexed);
```

Graph algorithms

- Algorithm categories - with filters
  - Traversal / Context class
    - TraversalBFS
    - TraversalDFS
  - Shortest path - SinglePairShortestPathBFS, SinglePairShortestPathDijkstra
    - addWeightedEdgeType method - add value of an attribute as a cost
    - getCost method - cost of the computed shortest path
  - Connectivity
    - StrongConnectivity - Gabow's algorithm strategy
    - WeakConnectivity - DFS strategy
  - Community detection (only unidirectional)
    - CommunitiesSCD

Conclusion

- Need to save a lot of constans (for node types and edge types)
- Retrieving data is pretty complicated
- Documentation for all supported languages together on one page
- Use their script language where possible - easily readable
- Missing some management tools - at least on command line :(

Questions?