User Defined Procedures and Functions

Extending Cypher

What will we learn?

Why do we need user defined procedures and functions?

Syntax & built in procedures

How to write and test your own procedure / function?

Turn Cypher query in procedure.

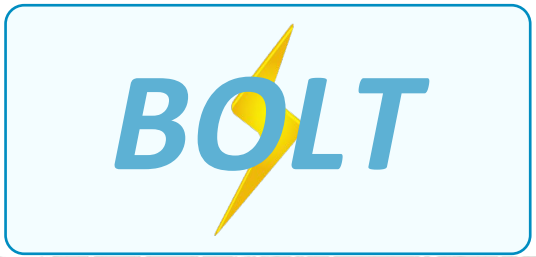
The APOC procedure library

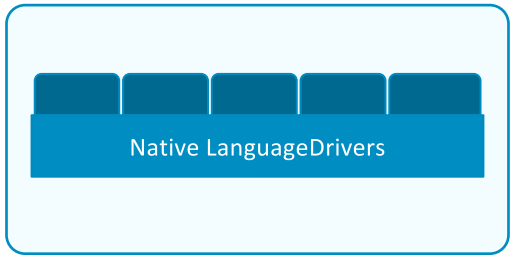
- Helper Functions
- Data Integration
- Graph Algorithms
- Graph Refactoring / Tx Management

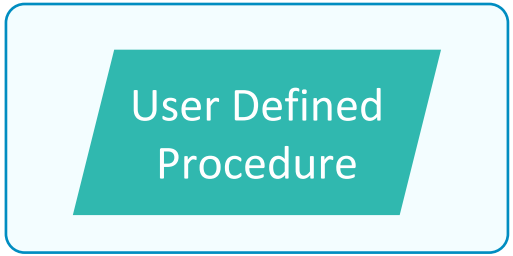
What are User Defined Procedures and Functions?

Neo4j Developer Surface

- **2000-2010:** 0.x - Embedded Java API
- **2010-2014:** 1.x - REST
- **2014-2015:** 2.x - Cypher over HTTP
- **2016 - : 3.0.x** - Bolt + Official Language Drivers + User Defined Procedures as Extension point
- **2016 - : 3.1.x User Defined Functions**



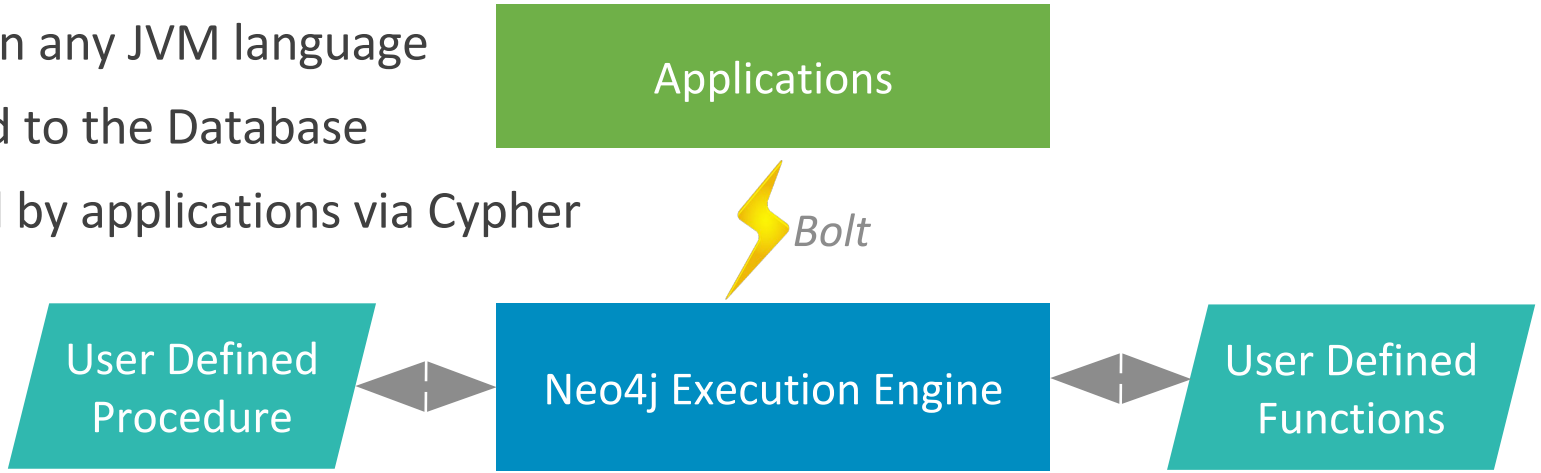




User Defined Procedures & Functions

User Defined Procedures & Functions let you write custom code that is:

- Written in any JVM language
- Deployed to the Database
- Accessed by applications via Cypher



Difference between Procedure and Function?

Procedures are more complex operations and generate streams of results

Functions are simple computations / conversions and return a single value

Procedures must be used within the **CALL** clause as a stand-alone element and **YIELD** their result columns

Functions can be used in any expression or predicate

User Defined Procedures

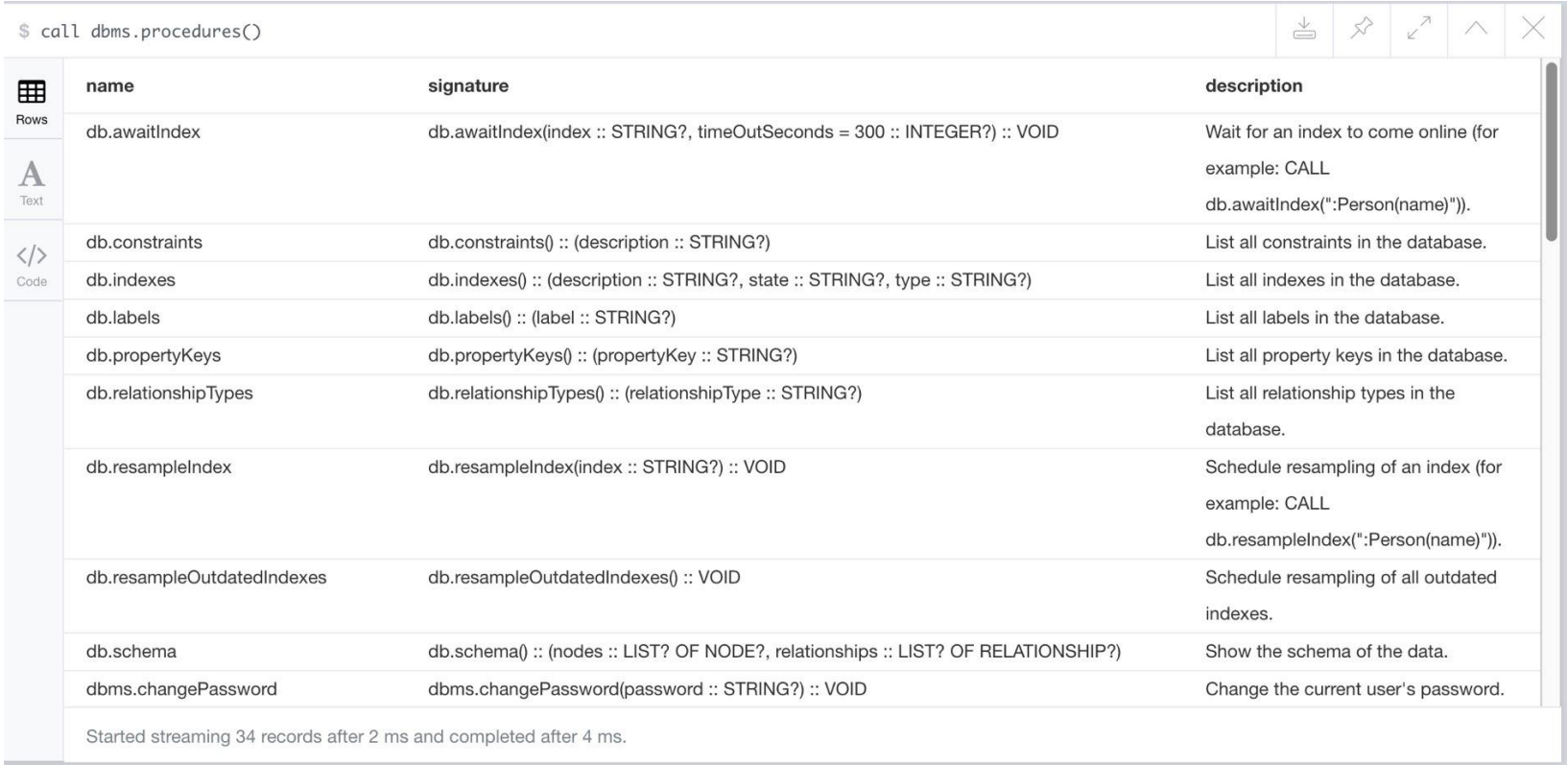
User-defined procedures are

- written in a JVM Language, e.g. Java,
- deployed into the database,
- and called from Cypher with the CALL clause
- take named parameters and return a stream of columns

http://neo4j.com/docs/developer-manual/current/procedures

Built-in Procedures

call dbms.procedures()



```
$ call dbms.procedures()
```

Rows | Text | Code

name	signature	description
db.awaitIndex	db.awaitIndex(index :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID	Wait for an index to come online (for example: CALL db.awaitIndex(":Person(name)")).
db.constraints	db.constraints() :: (description :: STRING?)	List all constraints in the database.
db.indexes	db.indexes() :: (description :: STRING?, state :: STRING?, type :: STRING?)	List all indexes in the database.
db.labels	db.labels() :: (label :: STRING?)	List all labels in the database.
db.propertyKeys	db.propertyKeys() :: (propertyKey :: STRING?)	List all property keys in the database.
db.relationshipTypes	db.relationshipTypes() :: (relationshipType :: STRING?)	List all relationship types in the database.
db.resampleIndex	db.resampleIndex(index :: STRING?) :: VOID	Schedule resampling of an index (for example: CALL db.resampleIndex(":Person(name)")).
db.resampleOutdatedIndexes	db.resampleOutdatedIndexes() :: VOID	Schedule resampling of all outdated indexes.
db.schema	db.schema() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)	Show the schema of the data.
dbms.changePassword	dbms.changePassword(password :: STRING?) :: VOID	Change the current user's password.

Started streaming 34 records after 2 ms and completed after 4 ms.

Cons of User Defined Procedures

- Quite involved **`CALL`** clause for simple, read-only computation, conversion functions or predicates
- Need to **`YIELD`** and select result columns
- Can’t be part of expressions

Meet User-Defined Functions

- Allows users to create their own functions and use them with Cypher
- Useful for expressing common computations, rules, conversions, predicates
- Functions can be used in any expression, predicates
- Extend the Neo4j 3.0 Stored Procedure mechanism

Creating a UUID with a Stored Procedure vs. with a Function

```
CALL apoc.create.uuid() YIELD uuid
CALL apoc.date.formatDefault(timestamp(),"ms") YIELD value as date
CREATE (:Document {id:uuid, created:date})
```

```
CREATE (:Document {id:apoc.create.uuid(), created:apoc.date.format(timestamp())})
```

Using User Defined Procedures and Functions?

Using - User Defined Procedures

```
// shortcut for stand alone call
CALL db.procedures()

// process result columns
CALL db.procedures()
YIELD name, signature, description
RETURN count(name)
```

http://neo4j.com/docs/developer-manual/current/procedures

Exercise: List available Procedures

1. List all procedures in the "dbms" namespace
2. List all procedures whose signature contains "NODE"
3. Group procedures by first (e.g. "db") and then second part of namespace, count them and aggregate their name part
4. Play around with at least 3 procedures you found, tell us what did you find?

Listing Procedures

```
CALL dbms.procedures()
YIELD name, signature, description
WITH * WHERE name STARTS WITH "db."
RETURN *
```

Listing Procedures

```
CALL dbms.procedures()
YIELD name, signature, description
WITH * WHERE signature CONTAINS "NODE"
RETURN *
```

Listing Procedures

```
CALL dbms.procedures()
YIELD name, signature, description
WITH split(name,".") AS parts
RETURN parts[0] AS package,
    count(*) AS count,
    collect(parts[-1]) AS names
```

Using User Defined Functions

- you use them like any other function
- in expressions or predicates
- for conversions, (business) decisions, computation
- Neo4j doesn't come with functions, but APOC does (70+)

Exercise: Use dbms.functions() to explore APOC functions

1. List all functions in the "apoc" namespace
2. List all functions whose signature contains "LONG"
3. Group functions by (e.g. "apoc.coll") namespace, count them and aggregate their name part
4. Play around with at least 3 functions you found,
   tell us what did you find?

User-Defined Functions in APOC - Packages

package	# of functions	example function
date & time conversion	3	`apoc.date.parse("time",["unit"],["format"])`
number conversion	3	`apoc.number.parse("number",["format"])`
general type conversion	8	`apoc.convert.toMap(value)`
type information and checking	4	`apoc.meta.type(value)`
collection and map functions	25	`apoc.map.fromList(["k1",v1,"k2",v2,"k3",v3])`
JSON conversion	4	`apoc.convert.toJson(value)`
string functions	7	`apoc.text.join(["s1","s2","s3"],"delim")`
hash functions	2	`apoc.util.md5(value)`

Simple Example:
Write, Test, Use a Function

User Defined Functions

User-defined functions are

- **`@UserFunction`** annotated, named Java Methods
  - default name: package + method
- take **`@Name`**'ed parameters (with default values)
- return a single value
- are read only
- can use **`@Context`** injected **`GraphDatabaseService`** etc
- run within Transaction of the Cypher Statement

Simple Function (UUID)

```
@UserFunction("create.uuid")
@Description("creates an UUID (universally unique id)")
public String uuid() {
    return UUID.randomUUID().toString();
}
```

```
RETURN create.uuid();
CREATE (p:Person {id: create.uuid(), name:{name}})
```

Test Code

1. Deploy & Register in Neo4j Server via **neo4j-harness**
2. Call & test via **neo4j-java-driver**

```
@Rule
public Neo4jRule neo4j = new Neo4jRule()
                         .withFunction( UUIDs.class );

try( Driver driver = GraphDatabase.driver( neo4j.boltURI() , config
) {
    Session session = driver.session();
    String uuid = session.run("RETURN create.uuid() AS uuid")
                         .single().get( 0 ).asString();
    assertThat( uuid,....);
}
```

Deploying User Defined Code

Build or download (shadow) jar

- Drop jar-file into **`$NEO4J_HOME/plugins`**
- Restart server
- Functions & Procedures should be available
- Otherwise check **`neo4j.log`** / **`debug.log`**

Example in Groovy

```
cp $GROOVY_HOME/lib/groovy-2.*.jar $NEO4J_HOME/plugins/
$GROOVY_HOME/groovyc function.groovy && jar cf $NEO4J_HOME/plugins/uuid.jar UDF.class
```

```
@Grab(value="org.neo4j:neo4j:3.1.0-BETA1",initClass=false)

class UDF {
   @UserFunction("create.uuid")
   @Description("creates an UUID")
   def String uuid() { UUID.randomUUID().toString() }
}
```

Optional Exercise: Write your own Function

- Get Groovy 2.x
- Copy
  `$GROOVY_HOME/lib/groovy-2.*.jar`
  to
  `$NEO4J_HOME/plugins/`
- Put code in `udf.groovy`
- Compile code
  `$GROOVY_HOME/groovyc udf.groovy`
- Build jar
  `jar cf $NEO4J_HOME/plugins/udf.jar UDF.class`
- Restart Neo4j

```
@Grab(value="org.neo4j:neo4j:3.1.0-BETA1",
      initClass=false)

import org.neo4j.graphdb.*
import org.neo4j.procedure.*

class UDF {
  @Context public GraphDatabaseService db

  @UserFunction("create.uuid")
  @Description("creates an UUID")
  def String uuid() {
    UUID.randomUUID().toString()
  }
}
```

User Defined Procedures

User-defined procedures are

- `@Procedure` annotated, named Java Methods
- additional **mode** attribute (Read, Write, Dbms)
- return a `Stream` of value objects with public fields
- value object fields are turned into columns

https://github.com/neo4j-examples/neo4j-procedure-template

Procedure Code Example (Dijkstra)

```
@Procedure
@Description("apoc.algo.dijkstra(startNode, endNode, 'KNOWS', 'distance') YIELD path," +
       " weight - run dijkstra with relationship property name as cost function")
public Stream<WeightedPathResult> dijkstra(
       @Name("startNode") Node startNode,
       @Name("endNode") Node endNode,
       @Name("type") String type,
       @Name("costProperty") String costProperty) {

   PathFinder<WeightedPath> algo = GraphAlgoFactory.dijkstra(
           PathExpanders.forType(RelationshipType.withName(type)),
           costProperty);
   Iterable<WeightedPath> allPaths = algo.findAllPaths(startNode, endNode);
   return Iterables.asCollection(allPaths).stream()
           .map(WeightedPathResult::new);
}
```

Test Procedure

```
@Rule
public Neo4jRule neo4j = new Neo4jRule()
                         .withProcedure( Dijkstra.class );
try( Driver driver = GraphDatabase.driver( neo4j.boltURI() , config
) {
    Session session = driver.session();
    String query =
     "MATCH ... CALL apoc.algo.dijkstra(...) YIELD path RETURN ...";
    String pathNames = session.run(query)
                         .single().get( 0 ).asString();
    assertThat( pathNames, ....);
}
```

APOC -
Awesome Procedures on Cypher

Why and How?

Apoc Procedures

- Cypher is expressive and great for graph operations but **misses some utilities**
- People built their own **one-off solutions**
- Now that Cypher is extensible, there should be a "**Standard Library**" of helpful procedures and functions
- APOC started as experimentation with procedures before Neo4j 3.0
- Evolved into an active community project with **200+ procedures and 75+ functions**

https://github.com/neo4j-contrib/neo4j-apoc-procedures

What does APOC cover?

Used by customers, users, prospects, field-team, dev-relations and sales,

APOC includes

- Functions for date, time, data conversion, collection handling, and more
- Procedures for Data Integration, Graph Algorithms, Graph Refactoring, Metadata, Cypher Batching
- TimeToLive (TTL), Triggers, Parallelization
- Articles, Documentation, Browser Guides

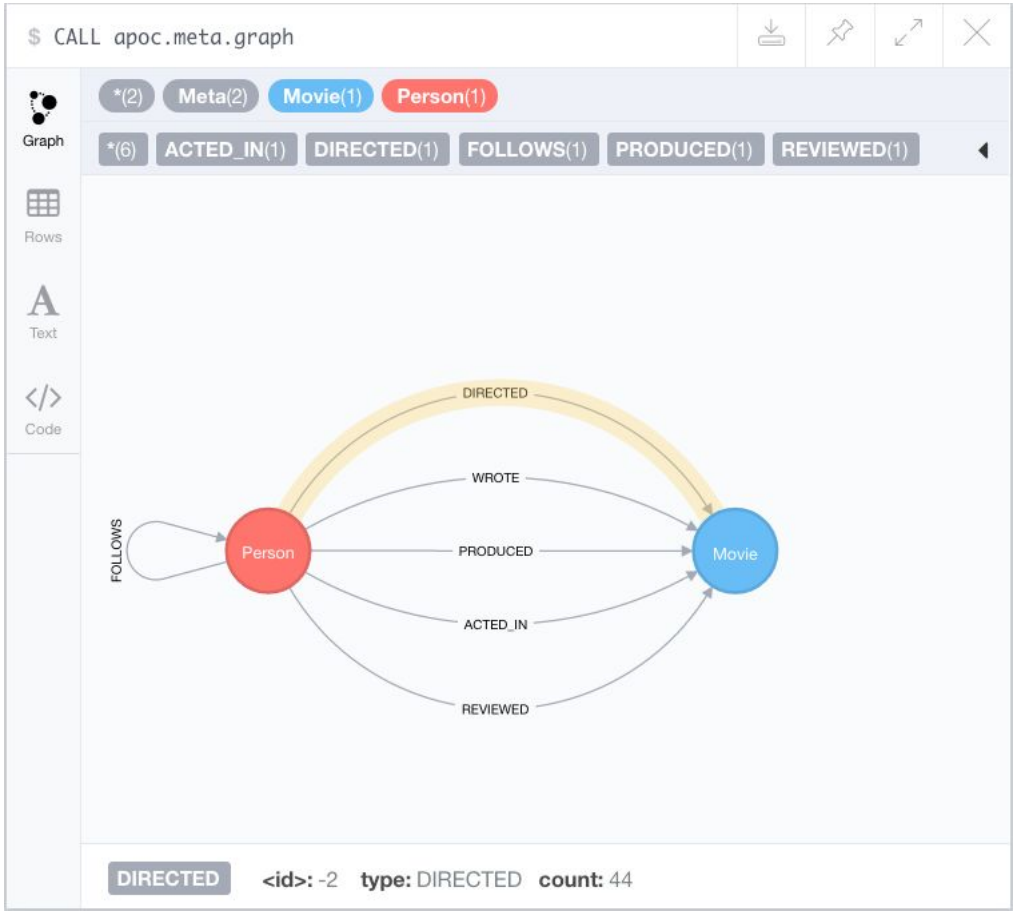
Apoc Procedures - Documentation

- documentation site github.com/neo4j-contrib/neo4j-apoc-procedures
- browser guide `:play http://guides.neo4j.com/apoc`
- many articles & blog posts neo4j.com/tag/apoc

(Meta)-Utilities, Converters

(Meta)-Utilities, Converters

- apoc.date.format(timestamp(),"ms","YYYY-MM-dd")
- apoc.number.parse("12.000,00")


$ CALL apoc.meta.graph
Graph
Rows
Text
Code
*(2)
Meta(2)
Movie(1)
Person(1)
*(6)
ACTED_IN(1)
DIRECTED(1)
FOLLOWS(1)
PRODUCED(1)
REVIEWED(1)
DIRECTED
WROTE
FOLLOWS
Person
PRODUCED
Movie
ACTED_IN
REVIEWED
DIRECTED
<id>: -2
type: DIRECTED
count: 44

Data Import and Export

Data Integration / Import

- databases
  - jdbc
  - mongodb
  - cassandra
  - elasticsearch
  - couchdb
- file formats
  - json
  - xml

LOAD JDBC

Load from relational database, either a full table or a sql statement

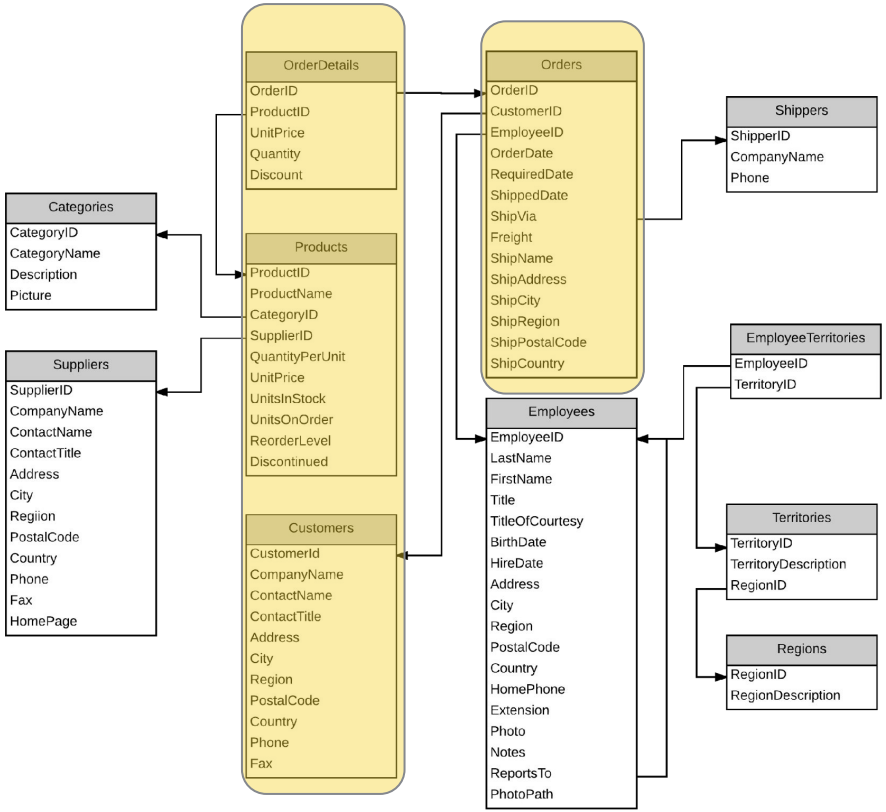
```
CALL apoc.load.jdbc('jdbc-url','TABLE') YIELD row
```

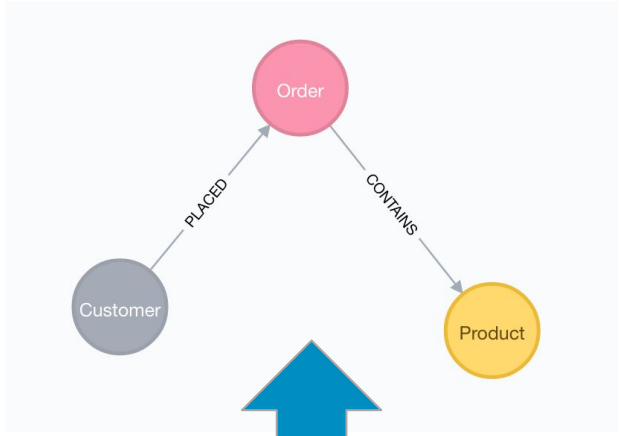
```
CALL apoc.load.jdbc('jdbc-url','SQL-STATEMENT') YIELD row
```

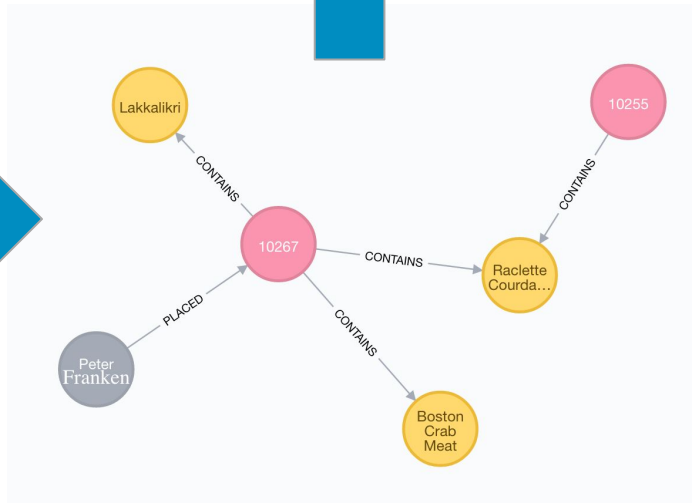
To simplify the JDBC URL syntax and protect credentials, you can configure aliases in conf/neo4j.conf:

```
apoc.jdbc.alias.url=jdbc:mysql://localhost:3306/<database>?user=<username>
CALL apoc.load.jdbc('alias','TABLE')
```

Load Northwind data from MySQL







Load Northwind data from MySQL - Products & Orders

```
// Create Product nodes
cypher CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/northwind?user=root","products")
YIELD row
CREATE (p:Product {ProductID: row.ProductID})
SET p.ProductName = row.ProductName,
    p.CategoryID  = row.CategoryID,
    p.SupplierID  = row.SupplierID
```

```
// Create Order nodes
cypher CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/northwind?user=root","orders")
YIELD row
CREATE (o:Order {OrderID: row.OrderID})
SET o.CustomerID = row.CustomerID,
    o.EmployeeID = row.EmployeeID
```

Load Northwind data from MySQL - Order Details

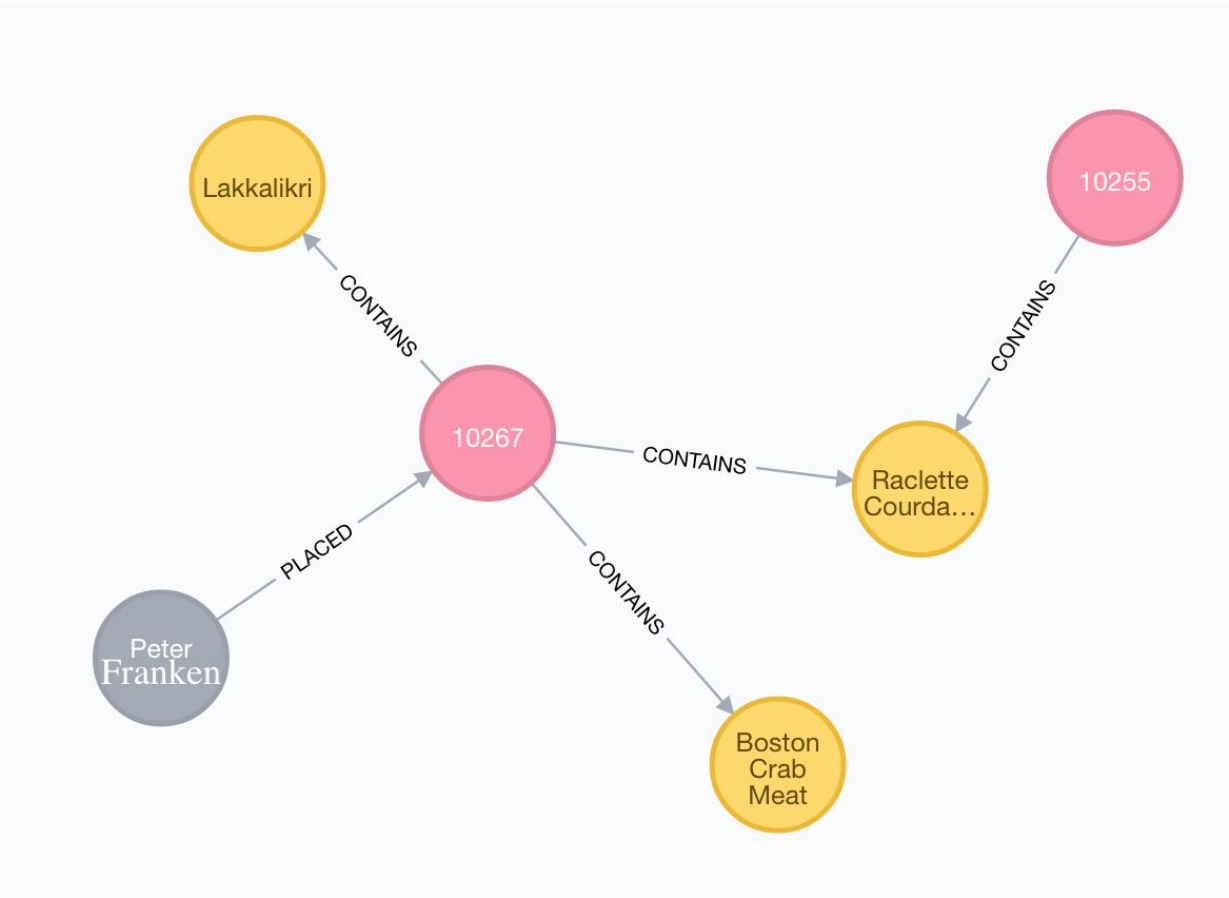
```
// Create CONTAINS relationships
cypher CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/northwind?
user=root","OrderDetails") YIELD row
MATCH (p:Product {ProductID: row.ProductID})
MATCH (o:Order {OrderID: row.OrderID})
CREATE (o)-[r:CONTAINS]->(p)
SET r.UnitPrice = row.UnitPrice
    r.Quantity  = row.Quantity,
    r.Discount  = row.Discount
```

Load Northwind data from MySQL - Customers

```
// Create Customer nodes
cypher CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/northwind?
user=root","Customers") YIELD row
CREATE (c:Customer {CustomerID: row.CustomerID})
SET c.Companyname  = row.CompanyName,
    c.ContactName  = row.ContactName,
    c.ContactTitle = row.ContactTitle
```

```
// create PLACED relationships
MATCH (o:Order)
MATCH (c:Customer {CustomerID: o.CustomerID})
CREATE (c)-[:PLACED]->(o)
```

Load Northwind data from MySQL - Graph Result



```
// simple collaborative filtering product recommendations
MATCH (c:Customer) WHERE c.ContactName = "Roland Mendel"
MATCH (c)-[:PLACED]->(o:Order)-[:CONTAINS]->(p:Product)
MATCH (p)<-[:CONTAINS]-(:Order)<-[:PLACED]-(other:Customer)
MATCH (other)-[:PLACED]->(:Order)-[:CONTAINS]->(p2:Product)
RETURN p2.ProductName, count(*) AS weight ORDER BY weight DESC
```

p2.ProductName	weight
Gorgonzola Telino	3933
Chang	3527
Guaran Fantstica	3509
Camembert Pierrot	3440
Raclette Courdavault	3396
Flotemysost	3379
Gnocchi di nonna Alice	3307
Jack's New England Clam Chowder	3179
Rhnbru Klosterbier	3151
Alice Mutton	3105
Pavlova	3075
Tarte au sucre	2969
Konbu	2773
Wimmers gute Semmelkndel	2766
Steeleye Stout	2689
Boston Crab Meat	2535

LOAD DATA

```
// load from JSON URL (e.g. web-api) to import JSON as stream of values if the JSON
was an array or a single value if it was a map
CALL apoc.load.json('json-url')
YIELD value as row

// import XML as single nested map with attributes and `_type`, `_text` fields
// and `_<childtype>` collections per child-element-type.
CALL apoc.load.xmlSimple('xml-url') YIELD value as doc

// load CSV fom URL as stream of values
CALL apoc.load.csv('csv-url',{config}) YIELD lineNo, list, map
// config contains any of:
{skip:1,limit:5,header:false,sep:'TAB',ignore:['tmp'],arraySep:';',
mapping:{years:{type:'int',arraySep:'-',array:false,name:'age',ignore:false}}`
```

Exercise: Import StackOverflow

Use the APOC's **apoc.load.json** to import stackoverflow from this URL

You can also find it in the documentation:
neo4j-contrib.github.io/neo4j-apoc-procedures

```
https://api.stackexchange.com/2.2/questions?pagesize=100&order=desc&sort=creation&tagged=neo4j&site=stackoverflow&filter=!5-i6Zw8Y)4W7vpy91PMYsKM-k9yzEsSC1_Uxlf
```

Periodic Execution & Transaction Control

Large operations? Consider TX batching

```
CALL apoc.periodic.iterate('

call
apoc.load.jdbc("jdbc:mysql://localhost:3306/northwind?user=root","
company")',

'CREATE (p:Person) SET p += value',

{batchSize:10000, parallel:true})

RETURN batches, total
```

Exercise 2: Import all pages of StackOverflow

Use the APOC's `apoc.cypher.iterate` with your previous **apoc.load.json** solution.

You can also find information about in the APOC documentation:
neo4j-contrib.github.io/neo4j-apoc-procedures

Graph Algorithms

Graph Algorithms Included

- Pathfinding: **apoc.algo.dijkstra**, a**poc.algo.aStar**
- PageRank: **apoc.algo.pageRankStats**,

  apoc.algo.pageRankWithCypher
- Centrality: **apoc.algo.betweenness**, **apoc.algo.closeness**
- Clustering: **apoc.algo.community**, **apoc.algo.wcc**, **apoc.algo.cliques**
- Node-Cover: **apoc.algo.cover**

Indexing

Example Step-by-Step Optimization

Example Query - Genre Overlap

```
WITH [] AS genreNames
UNWIND genreNames AS name
MATCH (g:Genre {name:name})<-[:HAS_GENRE]-(m:Movie)
WITH m, collect(g) AS genres
WHERE size(genres) = size(genreNames)
RETURN m
```

Example Query - Genre Overlap - Optimized

```
WITH [] AS genreNames
UNWIND genreNames AS name
MATCH (g:Genre {name:name})
WITH g ORDER BY size( (g)<-[:HAS_GENRE]-() ) ASC
WITH collect(g) AS genres
WITH head(genres) AS first, tail(genres) AS rest
MATCH (first)<-[:HAS_GENRE]-(m:Movie)
WHERE all(g IN rest | (m)<-[:HAS_GENRE]-(g))
RETURN m
```

Example Query - Genre Overlap - Optimized (2)

```
WITH [] AS genreNames
UNWIND genreNames AS name
MATCH (g:Genre {name:name})
WITH g ORDER BY size( (g)<-[:HAS_GENRE]-() ) ASC
WITH collect(g) AS genres
WITH head(genres) AS first, tail(genres) AS rest
MATCH (first)<-[:HAS_GENRE]-(m:Movie)-[:HAS_GENRE]->(other)
WITH m, collect(other) AS movieGenres, rest
WHERE all(g IN rest | g IN movieGenres)
RETURN m
```

Example - Genre Overlap - Procedure

- pass in list of genre names
- find genre with least movies (degree) as driver
- put other genres into **Set**
- iterate over movies of minimal genre
- for each movie
- for each genre that is in set increment counter
- if counter = Set size
  - add/stream movie to result

- separator -

Example - Genre Overlap - Procedure - Code

User-defined Procedures (Index Usage)

```
@Procedure("example.search")
@PerformsWrites // TODO: This is here as a workaround, because index().forNodes() is not read-only
public Stream<SearchHit> search( @Name("label") String label,
                                 @Name("query") String query )
{
    String index = indexName( label );

    // Avoid creating the index, if it's not there we won't be
    // finding anything anyway!
    if( !db.index().existsForNodes( index ))
    {
        // Just to show how you'd do logging
        log.debug( "Skipping index query since index does not exist: `%s`", index );
        return Stream.empty();
    }

    // If there is an index, do a lookup and convert the result
    // to our output record.
    return db.index()
            .forNodes( index )
            .query( query )
            .stream()
            .map( SearchHit::new );
}
```

https://github.com/neo4j-examples/neo4j-procedure-templat

User-defined Procedures (Index Usage)

```
@Procedure("example.search")
@PerformsWrites // TODO: This is here as a workaround, because index().forNodes() is not read-only
public Stream<SearchHit> search( @Name("label") String label,
                                 @Name("query") String query )
{
    String index = indexName( label );

    // Avoid creating the index, if it's not there we won't be
    // finding anything anyway!
    if( !db.index().existsForNodes( index ))
    {
        // Just to show how you'd do logging
        log.debug( "Skipping index query since index does not exist: `%s`", index );
        return Stream.empty();
    }

    // If there is an index, do a lookup and convert the result
    // to our output record.
    return db.index()
            .forNodes( index )
            .query( query )
            .stream()
            .map( SearchHit::new );
}
```

https://github.com/neo4j-examples/neo4j-procedure-templat