COSC 416 Neo4J Solutions

By: Giuseppe Burtini, Graeme Douglas, Yipin Guo

1.
a. START n=node(*)
b. START n=node:nodes(name = "dude")
c.

i. MATCH b-->a	Source:	14.10.3
ii. MATCH a-[:isA]->b	Source:	14.10.5
iii. MATCH a-[:wants|bought|sells]->b	Source:	14.10.6
iv. MATCH a-[*2]->b	Source:	14.8.4
v. MATCH a-[*1..5]->b	Source:	14.8.4
vi. MATCH a-[*]->b	Source:	14.8.4
vii. MATCH a-[r:*3..3]-b	Source:	14.8.4
viii. MATCH a-[r:wants*2..5]-b	Source:	15.10

2.
a.
```
START n=node(2) MATCH a<--n RETURN a.twid;
```

- Accept answers with "RETURN a" as well. The directionality of the match clause is important (if incorrect subtract 1 mark). The START node can also take a few other forms (match the node where the “category” field is right).

- The direction of the arrow a<--n is very important here. ANY change is incorrect. Note that the actual direction of the arrow isn’t the issue, but that the arrow goes from the node listed in the start clause to the one listed in the return clause.

There should be 380 results returned. More may indicate they included the grouping node, which is incorrect.

b.
```
START n=node(2) MATCH a-[r:TWEETED]-b<--n WHERE b.twid='nosqlweekly'
RETURN a.text;
```

- There should be two results returned.
- Partial marks if they do not start at node(2) but rather try to use an index to find the nosqlweekly twid. This solution will not run, because an index does not exist (and that is clear in the question), but it is “valid” in that it makes sense.

3. a. Answer can take many forms, but they should all have the general structure of an ID and a parent ID in a table. Correctness here should be fairly obvious, as long as there is the ability to maintain the hierarchy and names, it is correct.

```
CREATE TABLE Category
(
      id int primary key,
      name varchar(255),
      parentid int,
      level int,
      FOREIGN KEY parentId REFERENCES Category(id)
);

INSERT INTO Category VALUES (1, 'Games', null);
INSERT INTO Category VALUES (2, 'Role Playing Games', 1);
INSERT INTO Category VALUES (8, 'Adventure', 2);
INSERT INTO Category VALUES (3, 'Massively Online-Multiplayer', 2);
INSERT INTO Category VALUES (5, 'Strategy', 1);
INSERT INTO Category VALUES (4, 'RTS', 5);
INSERT INTO Category VALUES (6, 'Turn-based', 5);
INSERT INTO Category VALUES (7, 'Shooters', 1);
INSERT INTO Category VALUES (11, 'First person', 7);
INSERT INTO Category VALUES (12, 'Multiplayer', 11);
INSERT INTO Category VALUES (10, 'Third Person', 7);
```

Retrieve category list with a loop:

1) Retrieve all Categories with no parents (level 1 categories).
2) For each of these Categories:
   a) Retrieve its immediate descendants and print them.
   b) Recursively (in code not SQL) request descendants of this category (handles any level of nesting).

Example code from Derrick Pelletier:

```
<?
     function renderList($i) {
          $out = "<ul>\n";
          foreach ($i as $cat) :
               $out .= "\t<li>".$cat['name'];
               if(isset($cat['children'])&&count($cat['children']> 0)
                    $out .= renderList($cat['children']);
               $out .= "</li>";
          endforeach;
          $out .= "</ul>";
          return $out;
     }

     echo renderList($categories);
?>
```

Example code from Alex Yakovlev:

```
function getChildren( $level, $i )
{
     $padding = $i * 20;
     if ( $level == 0 )
          $sql = "SELECT * FROM categories WHERE ref_id IS NULL";
     else
     {
          $sql = "SELECT * FROM categories WHERE ref_id = $level;";
     }

     $query = mysql_query( $sql );

     while( $r = mysql_fetch_assoc( $query ) )
     {
          echo '<div style="padding-left:' . $padding . 'px;">';
               echo $r['name'];
          echo '</div>';
          getChildren( $r['id'], $i + 1 );
     }
}
```

Bonus WITH RECURSIVE query:

```
WITH Categories(id, name, parentId, sortKey, cdepth) AS
(
   SELECT id, name, parentId, cast(name+'' as varchar(2000)) as
sortKey, 1 as cdepth
    FROM Category WHERE parentId IS NULL
   UNION ALL
   SELECT C2.id, C2.name, C2.parentId,
cast(C1.sortKey+'_'+rtrim(C1.id)+'_'+C2.name as varchar(2000)) as
sortKey, C1.cdepth+1 as cdepth
   FROM Categories C1, Category C2
   WHERE C1.id = C2.parentId
)
SELECT id, name, parentId, sortKey, cdepth FROM Categories ORDER BY
sortKey ASC
```

b. Grabs all pairs down the tree. Cat, subcat pairs. Requires some post processing, there are better ways to do this (i.e., start at the node you want and walk the whole tree with range notation), but this is the most general.

```
START s = node(*)
MATCH s-->a
RETURN s, a
```

Please be very reasonable in marking question ‘b’. There are many correct answers, and so long as they have figured out how to display a hierarchical category structure, life is good. This can be at the command line, on a web interface or even outputted to a textfile (so long as the generating code is included).

Example code from Derrick Pelletier in Python:

```
from __future__ import print_function

# Import Neo4j modules
from py2neo import neo4j, cypher

# Get GraphDB instance
graph_db =
neo4j.GraphDatabaseService("http://localhost:7474/db/data/")

# Cypher query, {A} will be matched in the execution
query = "START begin=node({A}) MATCH p = begin-[*0..]->end RETURN end,
length(p)"

# Return a sequence of a particular string
def hr(count, char = "-", out = ""):
	while len(out) < count:
		out += char
	return out

# Row handler, prints out the data
def print_row(row):
    node, depth = row
    print(hr(depth) + node["category"])

# Begin Output
print("\nGenerating the category tree...")
print(hr(31,"="))
cypher.execute(graph_db, query, {"A": 0}, row_handler=print_row)
```

Example code from Alex Yakovlev: (PHP)

```
function getChildrenNEO4J( $games, $i, $client )
{
      $padding = $i * 20;

      // ROOT NODE
      if( $games == 'Games' )
      {
            echo '<div>';
                  echo $games;
            echo '</div>';
      }

      $cypherquery = "START n:node(category:{games}) MATCH n-[:parent-
of]->a RETURN a";

      $q = new Cypher\Query($client, $cypherquery,
                  array( 'games' => $games ) );

      $results = $q -> getResultSet();

      foreach( $results as $row )
      {
            echo '<div style="padding-left:' . $padding . 'px;">';
                  echo $row -> getProperty( 'category' );
            echo '</div>';
            getChildrenNEO4J( $row -> getProperty( 'category' ),
                        $i + 1, $client );
      }
}
```