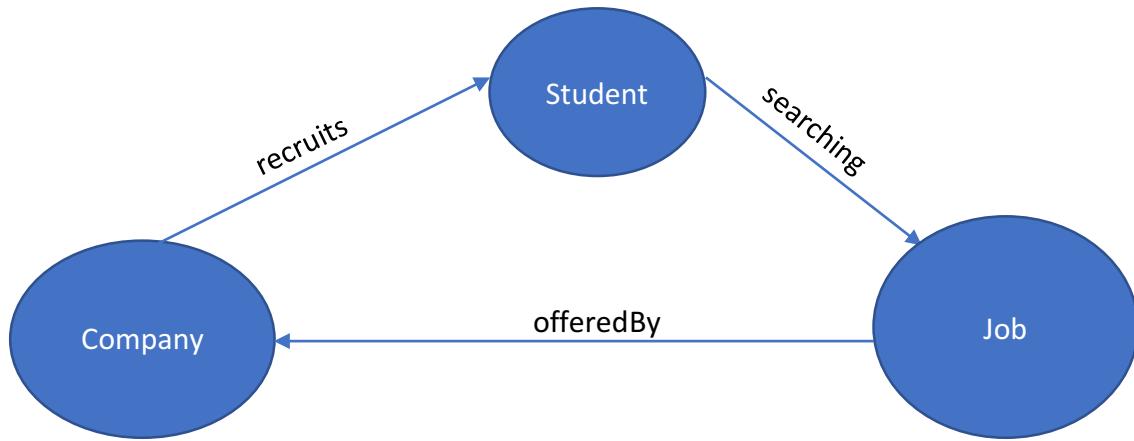


Mini Group Project – IBM Graph Database

Introduction –

We have implemented a “Student Job Search & Recruitment” and the schema design is as below



In this project we have understood and implemented the following:

- Schema, vertices, indexes, edges creation
- Traversing the graph through gremlin query interface

Graph Service Created

The screenshot shows a web browser window for the IBM Bluemix Catalog. The URL is <https://console.ng.bluemix.net/catalog/services/ibm-graph/?taxononyNavigation=services>. The page displays information about the IBM Graph service, including its description, service name (IBM Graph-Project Group 12), credential name (grp12), and features like Highly Available, Managed 24x7, Scale Seamlessly, and Powered by Apache TinkerPop3. A 'Create' button is visible at the bottom right.

Getting the credentials

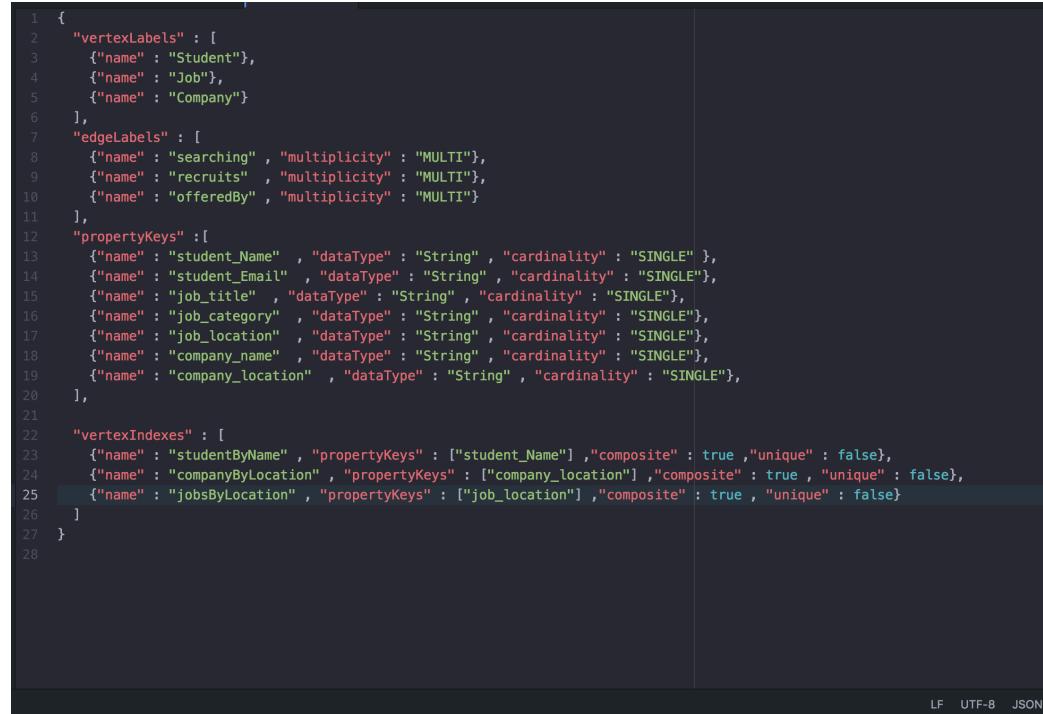
This screenshot is identical to the one above, showing the IBM Bluemix Catalog page for creating an IBM Graph service. It includes the same service details, features, and 'Create' button.

Generating the gd-token



A screenshot of a macOS Terminal window titled "PythonPrograms — bash — 204x56". The window shows the command "curl -X GET "https://999bbc7-874d-4e31-8e84-6ad72f4b38c1:c1a485e9-f283-43c1-a53a-14fa7bc22165@ibmgraph-alpha.ng.bluemix.net/e4153a1-c6df-4b76-80dc-8d3be24d65a9/_session"" being run, and the output is a long string of characters representing the gd-token.

Preparing this input schema



```
1  {
2    "vertexLabels" : [
3      {"name" : "Student"},
4      {"name" : "Job"},
5      {"name" : "Company"}
6    ],
7    "edgeLabels" : [
8      {"name" : "searching" , "multiplicity" : "MULTI"},
9      {"name" : "recruits" , "multiplicity" : "MULTI"},
10     {"name" : "offeredBy" , "multiplicity" : "MULTI"}
11   ],
12   "propertyKeys" : [
13     {"name" : "student_Name" , "dataType" : "String" , "cardinality" : "SINGLE" },
14     {"name" : "student_Email" , "dataType" : "String" , "cardinality" : "SINGLE" },
15     {"name" : "job_title" , "dataType" : "String" , "cardinality" : "SINGLE" },
16     {"name" : "job_category" , "dataType" : "String" , "cardinality" : "SINGLE" },
17     {"name" : "job_location" , "dataType" : "String" , "cardinality" : "SINGLE" },
18     {"name" : "company_name" , "dataType" : "String" , "cardinality" : "SINGLE" },
19     {"name" : "company_location" , "dataType" : "String" , "cardinality" : "SINGLE" },
20   ],
21
22   "vertexIndexes" : [
23     {"name" : "studentByName" , "propertyKeys" : ["student_Name"] , "composite" : true , "unique" : false},
24     {"name" : "companyByLocation" , "propertyKeys" : ["company_location"] , "composite" : true , "unique" : false},
25     {"name" : "jobsByLocation" , "propertyKeys" : ["job_location"] , "composite" : true , "unique" : false}
26   ]
27 }
28
```

The code editor interface includes tabs for "LF", "UTF-8", and "JSON" at the bottom right.

Posting and creating the input schema through curl command and schema API

```
1 import subprocess
2
3 schemaFile = open("studentSchema.json" , 'rb').read()
4
5
6 subprocess.call([
7     'curl',
8     '-X',
9     'POST',
10    '-d',
11    schemaFile,
12    '-H',
13    'content-type : application/json',
14    '-H',
15    'Authorization : gds-token N2Y2NDhiNWItN2IxZS000DQ5LWI3MzktMTB1NmQ1NzMyMTE10jE00DgxMDUxDgxNjY6bkxDa0NvV2NQ0WI1cWJHajlOV
16    "https://ibmgraph-alpha.ng.bluemix.net/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/g/schema'
17 ])
18
```

LF UTF-8 Python

Schema Created

```
Abhishek-MacBook-Pro:PythonPrograms Abhishek$ python createSchema.py
{"requestId": "e81be02a-2d03-44ee-a13b-502dcbe85dc", "status": {"message": "", "code": 200}, "attributes": {}}, "result": {"data": [{"propertyKeys": [{"name": "student_Name", "dataType": "String", "cardinality": "SINGLE"}, {"name": "student_Email", "dataType": "String", "cardinality": "SINGLE"}, {"name": "job_title", "dataType": "String", "cardinality": "SINGLE"}, {"name": "job_category", "dataType": "String", "cardinality": "SINGLE"}, {"name": "job_desc", "dataType": "String", "cardinality": "SINGLE"}, {"name": "company_name", "dataType": "String", "cardinality": "SINGLE"}, {"name": "company_email", "dataType": "String", "cardinality": "SINGLE"}, {"name": "location", "dataType": "String", "cardinality": "SINGLE"}], "edgeLabels": [{"name": "Student"}, {"name": "Job"}, {"name": "Company"}], "directed": true, "multiplicity": "MULTI"}, {"name": "recruits", "directed": true, "multiplicity": "MULTI"}, {"name": "offeredBy", "directed": true, "multiplicity": "MULTI"}, {"vertexIndexes": [{"name": "studentByName", "composite": true, "unique": false, "propertyKeys": [{"name": "student_Name"}], "requiresReindex": false, "type": "vertex"}, {"name": "jobsByLocation", "composite": true, "unique": false, "propertyKeys": [{"name": "job_location"}], "requiresReindex": false, "type": "vertex"}, {"name": "companyByLocation", "composite": true, "unique": false, "propertyKeys": [{"name": "company_location"}], "requiresReindex": false, "type": "vertex"}], "edgeIndexes": [{"name": "job_recruit", "type": "vertex"}, {"name": "student_recruit", "type": "vertex"}, {"name": "job_offered", "type": "vertex"}, {"name": "student_offered", "type": "vertex"}], "meta": {}}}Abhishek-MacBook-Pro:PythonPrograms Abhishek$
```

Creating vertex through vertices API

```
1 import subprocess
2
3 vertexDataJsonQuery = '{ "label": "Company", "properties": {"company_name": "IBM", "company_location": "Sunnyvale"} }'
4
5 subprocess.call([
6     'curl',
7     '-X',
8     'POST',
9     '-d',
10    vertexDataJsonQuery,
11    '-H',
12    'content-type : application/json',
13    '-H',
14    'Authorization : gds-token 0WI5YmJhYzctMDc0ZC00ZTMxLThlODQtNmFkNzJmNGIzOGMxObjE00DgxMDYyOTUx0DQ6NFBoY1hOUW00VnVCQU5BUmpzS
15    "https://ibmgraph-alpha.ng.bluemix.net/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/g/vertices'
16 ])
17
```

LF UTF-8 Python

Vertex Created

```
Abhishek-MacBook-Pro:PythonPrograms Abhishek$ python creatingVertices.py
Abhishek-MacBook-Pro:PythonPrograms Abhishek$
```

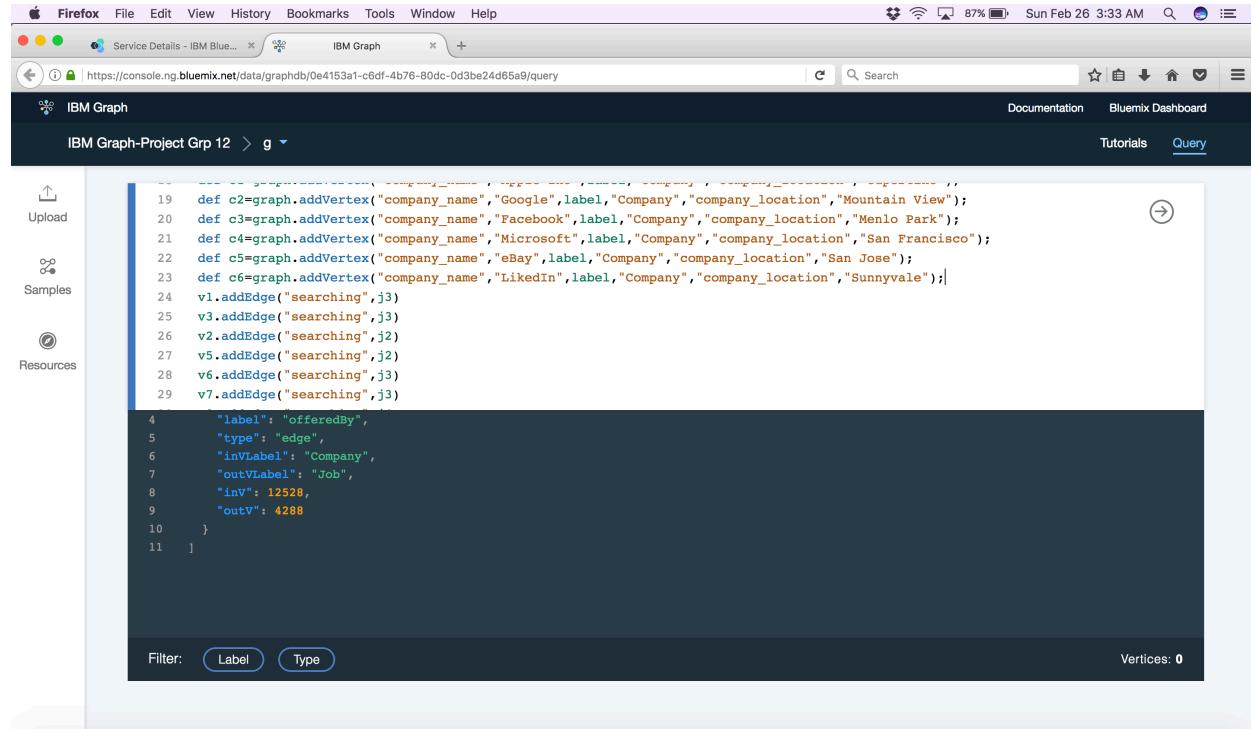
Creating Edges

```
1 import subprocess
2
3 edgeDataJsonQuery = '{ "outV" : 4336,"label": "recruits", "inV" : 4208 }'
4
5 subprocess.call([
6     'curl',
7     '-X',
8     'POST',
9     '-d',
10    edgeDataJsonQuery,
11    '-H',
12    'content-type : application/json',
13    '-H',
14    'Authorization : gds-token 0WI5YmJhYzctMDc0ZC00ZTMxLTh1ODQtNmFkNzJmNGIzOGMx0jE00DgxMDYyOTUxOD06NFBoY1h0UW00VnVCQUSBUmpzS3pFV3YzK1BLbmVreDVLN0x5YjgvaT
15    "https://ibmgraph-alpha.ng.bluemix.net/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/g/edges"
16 ])
17
18
19
```

Edges Created

```
Abhishek-MacBook-Pro:PythonPrograms Abhishek$ python creatingEdges.py
{"requestId":"8b670b34-4878-419a-a4de-ba49ff35e80","status":{"message":"","code":200,"attributes":{}}, "result": {"data": [{"id": "odxe-38w-2l91-37k", "label": "searching", "type": "edge", "inVLabel": "Job", "outVLabel": "Student", "inV": 4160, "outV": 4208}], "meta": {}}}Abhishek-MacBook-Pro:PythonPrograms Abhishek$
```

Adding some more data to vertices and edges through gremlin



The screenshot shows the IBM Graph interface in a browser window. On the left, there are three sidebar panels: 'Upload' (with a file icon), 'Samples' (with a gear icon), and 'Resources' (with a magnifying glass icon). The main area is a code editor containing a Gremlin script:

```
19 def c2=graph.addVertex("company_name","Google",label,"Company","company_location","Mountain View");
20 def c3=graph.addVertex("company_name","Facebook",label,"Company","company_location","Menlo Park");
21 def c4=graph.addVertex("company_name","Microsoft",label,"Company","company_location","San Francisco");
22 def c5=graph.addVertex("company_name","eBay",label,"Company","company_location","San Jose");
23 def c6=graph.addVertex("company_name","LinkedIn",label,"Company","company_location","Sunnyvale");
24 v1.addEdge("searching",j3)
25 v3.addEdge("searching",j3)
26 v2.addEdge("searching",j2)
27 v5.addEdge("searching",j2)
28 v6.addEdge("searching",j3)
29 v7.addEdge("searching",j3)

4     "label": "offeredBy",
5     "type": "edge",
6     "inVLabel": "Company",
7     "outVLabel": "Job",
8     "inV": 12528,
9     "outV": 4288
10   }
11 }
```

At the bottom of the code editor, there are three buttons: 'Filter:', 'Label' (which is selected), and 'Type'. To the right of the code editor, it says 'Vertices: 0'.

All data for vertices and edges added successfully

The screenshot shows the IBM Graph interface in a Firefox browser window. The title bar indicates "Service Details - IBM Blue..." and the URL is "https://console.ng.bluemix.net/data/graphdb/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/query". The main area displays a graph visualization with a single vertex labeled "1". On the left sidebar, there are sections for "Upload", "Samples", and "Resources". A code editor window titled "Graph: g" contains the following Python code:

```
def v1=graph.addVertex("student_Name","Abhishek Upadhyay",label,"Student","student_Email","abhishek.upadhyay")
```

The code defines a vertex `v1` with properties: `student_Name` ("Abhishek Upadhyay"), `label` ("Student"), and `student_Email` ("abhishek.upadhyay"). The code editor has a "Filter" dropdown set to "Label" and a "Vertices: 0" indicator.

Finding Student Vertex with a particular name

The screenshot shows the IBM Graph interface in a Firefox browser window. The title bar indicates "Service Details - IBM Blue..." and the URL is "https://console.ng.bluemix.net/data/graphdb/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/query". The main area displays a graph visualization with a single vertex labeled "1". On the left sidebar, there are sections for "Upload", "Samples", and "Resources". A code editor window titled "Graph: g" contains the following Python code:

```
def gt = graph.traversal();
gt.V().has("student_Name","Abhishek Upadhyay")
```

The code defines a traversal `gt` and filters it by `student_Name` ("Abhishek Upadhyay"). The code editor has a "Filter" dropdown set to "Label" and a "Vertices: 1" indicator. A green circular node labeled "Student" is visible in the graph visualization.

Finding the incoming node to Student node

The screenshot shows the IBM Graph interface in a browser window. The URL is <https://console.ng.bluemix.net/data/graphdb/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/query>. The main area displays a code editor with the following Cypher query:

```
1 def gt = graph.traversal();
2 gt.V().has("student_Name","Abhishek Upadhyay").in()
```

Below the code editor, the graph visualization shows a single green vertex labeled "Company". To the right of the graph, it says "Vertices: 1".

Finding both the incoming and outgoing nodes of Student Node

The screenshot shows the IBM Graph interface in a browser window. The URL is <https://console.ng.bluemix.net/data/graphdb/0e4153a1-c6df-4b76-80dc-0d3be24d65a9/query>. The main area displays a code editor with the following Cypher query:

```
1 def gt = graph.traversal();
2 gt.V().has("student_Name","Abhishek Upadhyay").both()
```

Below the code editor, the graph visualization shows two vertices: a green one labeled "Job" and a blue one labeled "Company". To the right of the graph, it says "Vertices: 2".

Finding the student details who have been recruited by companies based out of Cupertino

Screenshot of the IBM Graph interface showing a query for students recruited by companies in Cupertino.

Query:

```
1 def gt = graph.traversal();
2 def v = gt.V().has("company_location","Cupertino");
3 v.outE("recruits").inV().path();
```

Graph: g

```
= graph.traversal();def v = gt.V().has("company_location","Cupertino");v.outE("recruits").inV().path();
```

Results:

```
1 + [
2 + {
3 +   "labels": [
4 +     [],
5 +     [],
6 +     []
7 +   ],
8 +   "objects": [
9 +     {
10 +       "id": 4160,
11 +       "label": "Company",
12 +       "type": "vertex",
13 +       "properties": {
14 +         "company_name": [
15 +           {"name": "Hewlett-Packard"}]
```

Vertices: 8

Finding the student details been recruited by company located at Mountain View

Screenshot of the IBM Graph interface showing a query for students recruited by companies in Mountain View.

Query:

```
1 def gt = graph.traversal();
2 def v = gt.V().has("company_location","Mountain View");
3 v.outE("recruits").inV().values("student_Email").path();
```

Graph: g

```
def gt = graph.traversal();def v = gt.V().has("company_location","Mountain View");v.outE("recruits").inV().values("student_Email").path();
```

Results:

```
1 + [
2 + {
3 +   "labels": [
4 +     [],
5 +     [],
6 +     []
7 +   ],
8 +   "objects": [
9 +     {
10 +       "id": 8384,
11 +       "label": "Company",
12 +       "type": "vertex",
13 +       "properties": {
14 +         "company_name": [
```

Vertices: 4

Finding the students searching for Jobs and been recruited by companies located at Cupertino

The screenshot shows the IBM Graph interface in a Firefox browser. The query entered is:

```
(() .has("company_location", "Cupertino"); v.outE("recruits").inV().outE("searching").inV().values().path());
```

The results show a graph with 5 vertices: Company (green), Job (purple), and three Students (blue). The connections are: Company → Job → Student, and Company → Student.

Finding the students searching for Jobs and been recruited by companies located at Mountain View

The screenshot shows the IBM Graph interface in a Firefox browser. The query entered is:

```
as("company_location", "Mountain View"); v.outE("recruits").inV().outE("searching").inV().values().path();
```

The results show a graph with 4 vertices: Company (green), Job (purple), and two Students (blue). The connections are: Company → Job → Student, and Company → Student.