

SAN JOSE STATE UNIVERSITY

SPRING 2017



Project team 18

Team Members:

1. Aayushi Mittal
2. Amreen Abdul Wahab
3. Anirrudh Venkatraman
4. Rimpay Bharot

Purchase Order Management System in SQLite

Objective:

1. Design a database for Purchase Order Management System.
2. Create a sample schema with necessary tables.
3. Insert sample data into the tables.
4. Execute different queries.

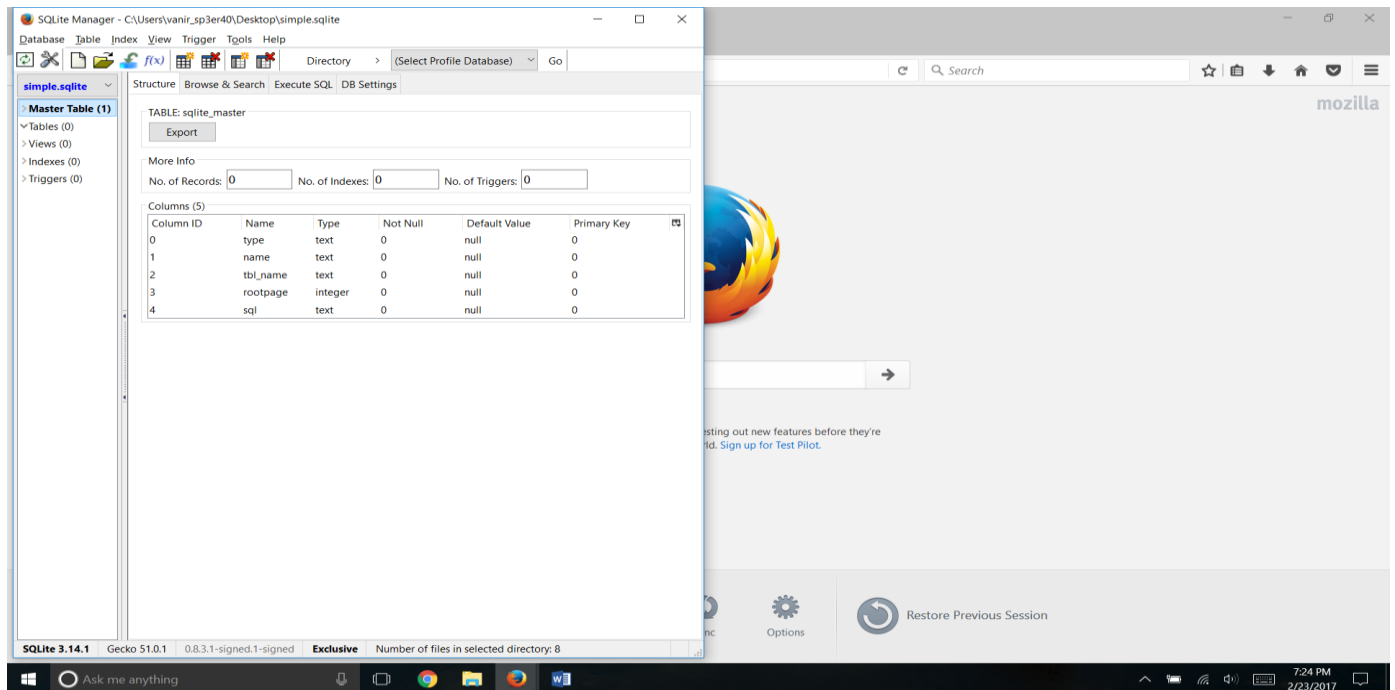
SQLite manager with Mozilla Firefox

1. Creation of database in the SQLite manager for PurchaseOrderManagementSystem

The screenshot displays two applications side-by-side. On the left is the SQLite Manager window, titled 'SQLite Manager - C:\Users\vanir_sp3er40\Desktop\purchase order management system.sqlite'. The 'Import Wizard' tab is active, showing options for importing data from a CSV or SQL file. The 'Character Encoding' is set to 'UTF-8'. The 'Fields separated by' section has 'Comma (,)' selected. The 'Fields enclosed by' section has 'Double quotes (")' selected. The 'OK' button is highlighted. On the right is the Microsoft Excel window, titled 'output - Excel'. The spreadsheet shows data imported from the SQLite database. The data is organized into columns A through K, with rows 1 through 10 containing sample data. The data is as follows:

	A	B	C	D	E	F	G	H	I	J	K
1	ABC	A100	22	yes							
2	APPLE	B100	33	yes							
3	CZY	C100	44	yes							
4	CZY	D100	55	yes							
5	CZY	E100	66	yes							
6	DXB	A101	77	yes							
7	DXB	B101	99	yes							
8	ABC	C101	88	yes							
9	ABC	D101	11	yes							
10	KUC	E101	110	yes							

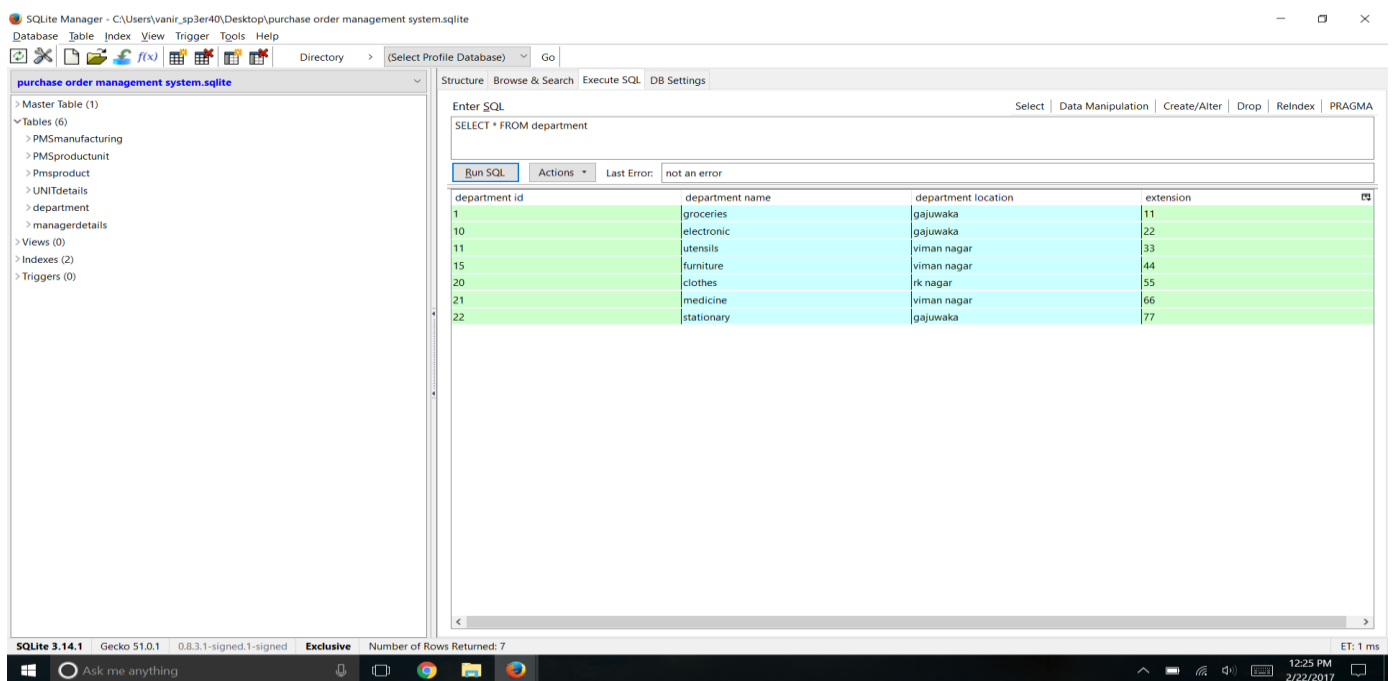
2. Adding tables to database by importing table from data present locally



Queries executed:

3. Displaying the synched tables from SQLite manager using

SELECT * FROM table name query



4. performing inner join between two tables by using the SQL query

SELECT *FROM table 1 INNER JOIN table 2

The screenshot shows the SQLite Manager interface with the database 'purchase_order_management_system.sqlite' open. The left sidebar displays the database structure, including tables like PMSmanufacturing, PMSproductunit, Pmsproduct, UNITdetails, department, and managerdetails. The main window shows the 'Enter SQL' tab with the query: `SELECT * FROM managerdetails INNER JOIN UNITdetails`. The 'Run SQL' button is highlighted, and the 'Last Error' field shows 'not an error'. The query results are displayed in a table with 10 columns: manager id, job, boss code, salary, commission, unit id, unit name, price, and total pieces. The results show 50 rows of data, including various jobs like cleaning, staff incharge, and maintenance, each associated with a boss code, salary, commission, unit id, unit name, price, and total pieces.

manager id	job	boss code	salary	commission	unit id	unit name	price	total pieces
25	cleaning	100	25,000	1000	66	sample 2	25	30
25	cleaning	100	25,000	1000	77	sample 3	35	20
25	cleaning	100	25,000	1000	88	product 1	20	300
25	cleaning	100	25,000	1000	99	product 2	100	20
25	cleaning	100	25,000	1000	11	product 3	100	200
25	cleaning	100	25,000	1000	110	product 4	20	10
26	staff incharge	100	25,000	2000	22	gradient 1	25	20
26	staff incharge	100	25,000	2000	33	gradient 2	30000	10
26	staff incharge	100	25,000	2000	44	gradient 3	300	15
26	staff incharge	100	25,000	2000	55	sample 1	100	20
26	staff incharge	100	25,000	2000	66	sample 2	25	30
26	staff incharge	100	25,000	2000	77	sample 3	35	20
26	staff incharge	100	25,000	2000	88	product 1	20	300
26	staff incharge	100	25,000	2000	99	product 2	100	20
26	staff incharge	100	25,000	2000	11	product 3	100	200
26	staff incharge	100	25,000	2000	110	product 4	20	10
29	maintenance	101	30,000	5000	22	gradient 1	25	20
29	maintenance	101	30,000	5000	33	gradient 2	30000	10
29	maintenance	101	30,000	5000	44	gradient 3	300	15
29	maintenance	101	30,000	5000	55	sample 1	100	20
29	maintenance	101	30,000	5000	66	sample 2	25	30
29	maintenance	101	30,000	5000	77	sample 3	35	20
29	maintenance	101	30,000	5000	88	product 1	20	300
29	maintenance	101	30,000	5000	99	product 2	100	20

5. Selecting a particular value from a table using SQL query SELECT * FROM table1 WHERE value

The screenshot shows the SQLite Manager interface with the database 'purchase_order_management_system.sqlite' open. The left sidebar displays the database structure, including tables like PMSmanufacturing, PMSproductunit, Pmsproduct, UNITdetails, department, and managerdetails. The main window shows the 'Enter SQL' tab with the query: `SELECT * FROM managerdetails WHERE job = 'cleaning'`. The 'Run SQL' button is highlighted, and the 'Last Error' field shows 'not an error'. The query results are displayed in a table with 5 columns: manager id, job, boss code, salary, and commission. The results show 1 row of data for the job 'cleaning'.

manager id	job	boss code	salary	commission
25	cleaning	100	25,000	1000

6. selecting particular column from a table using query

SELECT column name FROM table name

The screenshot shows the SQLite Manager interface. The left pane displays the database structure for 'purchase order management system.sqlite', including tables like PMSproductunit. The right pane shows the 'Enter SQL' input field with the query 'SELECT price FROM PMSproductunit'. Below the input field, the 'Run SQL' button is visible. The output pane displays the results of the query, showing a single column 'price' with 10 rows of values: 25, 30, 30, 10, 25, 35, 20, 100, 1000, and 2000. The status bar at the bottom indicates 'Number of Rows Returned: 10'.

price
25
30
30
10
25
35
20
100
1000
2000

7. Finding sum of values added in a particular column using the SQL query

The screenshot shows the SQLite Manager interface. The left pane displays the database structure for 'purchase order management system.sqlite'. The right pane shows the 'Enter SQL' input field with the query 'SELECT SUM (price) FROM PMSproductunit'. Below the input field, the 'Run SQL' button is visible. The output pane displays the result of the query, showing a single row with the value '3275' under the column 'SUM (price)'. The status bar at the bottom indicates 'Number of Rows Returned: 1'.

SUM (price)
3275

8. Selecting distinct value from the entries made in the table using query

SELECT DISTINCT value FROM table name

The screenshot shows the SQLite Manager application interface. The left pane displays the database structure for 'purchase order management system.sqlite', with the 'managerdetails' table selected. The right pane shows the 'Enter SQL' field with the query 'SELECT DISTINCT salary FROM managerdetails'. Below the query field, the 'Run SQL' button is highlighted. The results pane displays a table with one column, 'salary', and four rows of distinct values: 25,000, 30,000, 31,000, and 40,000. The status bar at the bottom indicates 'Number of Rows Returned: 4' and 'ET: 4 ms'.

salary
25,000
30,000
31,000
40,000

9. Adding a column to a table in the database

ALTER TABLE <table name> ADD <column name> <type>

The screenshot shows the SQLite Manager interface. On the left, the database structure is displayed, including the 'department' table. The main pane shows the 'Enter SQL' field with the command: `ALTER TABLE department ADD remarks varchar`. Below the command field, there are buttons for 'Run SQL', 'Actions', and 'Last Error: not an error'. The status bar at the bottom indicates 'SQLite 3.14.1', 'Gecko 51.0.1', and 'Number of Rows Returned: 0'.

The screenshot shows the SQLite Manager interface with the 'department' table selected. The table data is displayed in a grid view. The table has five columns: 'department id', 'department name', 'department location', 'extension', and 'remarks'. The data is as follows:

department id	department name	department location	extension	remarks
1	groceries	gajuwaka	11	
10	electronic	gajuwaka	22	
11	utensils	viman nagar	33	
15	furniture	viman nagar	44	
20	clothes	rk nagar	55	
21	medicine	viman nagar	66	
22	stationary	gajuwaka	77	

The status bar at the bottom indicates 'SQLite 3.14.1', 'Gecko 51.0.1', and 'Number of Rows Returned: 0'.

10. Removing a column from the table

ALTER TABLE <table> name DROP COLUMN <column name>

The screenshot shows the SQLite Manager interface with the database 'purchase_order_management_system.sqlite' open. The left sidebar shows the database structure, including tables like 'department'. The main window displays the SQL editor with the following statement:

```
ALTER TABLE department DROP COLUMN remarks;
```

Below the editor, a message indicates a syntax error: 'near "DROP": syntax error'. The status bar at the bottom shows 'SQLite 3.14.1' and 'Gecko 51.0.1'.

Below the first screenshot, the same SQLite Manager interface is shown, but the 'department' table is selected, and its data is displayed in a table view. The table has four columns: 'department id', 'department name', 'department location', and 'extension'.

department id	department name	department location	extension
1	groceries	gajuwaka	11
10	electronic	gajuwaka	22
11	utensils	viman nagar	33
15	furniture	viman nagar	44
20	clothes	rk nagar	55
21	medicine	viman nagar	66
22	stationary	gajuwaka	77

DB2 Express C

1. We will create a sample database of DB2.

```
Administrator: DB2 CLP - DB2COPY1 - db2
For more detailed help, refer to the Online Reference Manual.

db2 -> connect to sample

Database Connection Information

Database server      = DB2/NT 10.5.4
SQL authorization ID = MISHI
Local database alias = SAMPLE

db2 -> list tables

Table/View          Schema      Type      Creation time
-----
ACT                 MISHI      T        2017-02-17-17:28:40.969007
ADEFUSR            MISHI      S        2017-02-17-17:28:41.831001
CATALOG            MISHI      T        2017-02-17-17:28:43.207002
CL_SCHED           MISHI      T        2017-02-17-17:28:40.639001
CUSTOMER           MISHI      T        2017-02-17-17:28:43.124001
DEPARTMENT         MISHI      T        2017-02-17-17:28:40.681002
DEPT               MISHI      A        2017-02-17-17:28:40.738000
EMP                MISHI      A        2017-02-17-17:28:40.780000
EMPACT            MISHI      A        2017-02-17-17:28:40.969000
EMPLOYEE          MISHI      T        2017-02-17-17:28:40.738003
EMPMDC            MISHI      T        2017-02-17-17:28:42.223002
EMPPROJECT        MISHI      T        2017-02-17-17:28:40.955002
EMP_ACT           MISHI      A        2017-02-17-17:28:40.969004
EMP_PHOTO         MISHI      T        2017-02-17-17:28:40.780003
EMP_RESUME        MISHI      T        2017-02-17-17:28:40.924002
INVENTORY         MISHI      T        2017-02-17-17:28:43.098001
IN_TRAY           MISHI      T        2017-02-17-17:28:40.996002
ORG               MISHI      T        2017-02-17-17:28:41.005002
PRODUCT           MISHI      T        2017-02-17-17:28:43.036002
PRODUCTSUPPLIER   MISHI      T        2017-02-17-17:28:43.278001
PROJ              MISHI      A        2017-02-17-17:28:40.927000
PROJECT           MISHI      T        2017-02-17-17:28:40.927003
PURCHASEORDER     MISHI      T        2017-02-17-17:28:40.868002
SALES             MISHI      T        2017-02-17-17:28:41.023002
STAFF             MISHI      T        2017-02-17-17:28:41.013002
STAFFG           MISHI      T        2017-02-17-17:28:41.666001
SUPPLIERS         MISHI      T        2017-02-17-17:28:43.244002
VACT              MISHI      V        2017-02-17-17:28:41.108001
VASTRDE1          MISHI      V        2017-02-17-17:28:41.131002
VASTRDE2          MISHI      V        2017-02-17-17:28:41.136001
VDEPMG1           MISHI      V        2017-02-17-17:28:41.117001
VDEPT             MISHI      V        2017-02-17-17:28:41.065001
VENP              MISHI      V        2017-02-17-17:28:41.103002
VENPDPT1          MISHI      V        2017-02-17-17:28:41.129001
VENPLP            MISHI      V        2017-02-17-17:28:41.156001
VENPPROJECT       MISHI      V        2017-02-17-17:28:41.113001
```

```
Administrator: DB2 CLP - DB2COPY1 - db2
db2 -> select * from sales

SALES_DATE  SALES_PERSON  REGION      SALES
-----
12/31/2005  LUCCHESSEI    Ontario-South  1
12/31/2005  LEE           Ontario-South  3
12/31/2005  LEE           Quebec        1
12/31/2005  LEE           Manitoba      2
12/31/2005  GOUNOT        Quebec        1
03/29/2006  LUCCHESSEI    Ontario-South  3
03/29/2006  LUCCHESSEI    Quebec        1
03/29/2006  LEE           Ontario-South  2
03/29/2006  LEE           Ontario-North  2
03/29/2006  LEE           Quebec        3
03/29/2006  LEE           Manitoba      5
03/29/2006  GOUNOT        Ontario-South  3
03/29/2006  GOUNOT        Quebec        1
03/29/2006  GOUNOT        Manitoba      7
03/30/2006  LUCCHESSEI    Ontario-South  1
03/30/2006  LUCCHESSEI    Quebec        2
03/30/2006  LUCCHESSEI    Manitoba      1
03/30/2006  LEE           Ontario-South  7
03/30/2006  LEE           Ontario-North  3
03/30/2006  LEE           Quebec        7
03/30/2006  LEE           Manitoba      4
03/30/2006  GOUNOT        Ontario-South  2
03/30/2006  GOUNOT        Quebec        18
03/31/2006  GOUNOT        Manitoba      1
03/31/2006  LUCCHESSEI    Manitoba      1
03/31/2006  LEE           Ontario-South  14
03/31/2006  LEE           Ontario-North  3
03/31/2006  LEE           Quebec        7
03/31/2006  LEE           Manitoba      3
03/31/2006  GOUNOT        Ontario-South  2
03/31/2006  GOUNOT        Quebec        1
04/01/2006  LUCCHESSEI    Ontario-South  1
04/01/2006  LUCCHESSEI    Manitoba      1
04/01/2006  LEE           Ontario-South  8
04/01/2006  LEE           Ontario-North  1
04/01/2006  LEE           Quebec        8
04/01/2006  LEE           Manitoba      9
04/01/2006  GOUNOT        Ontario-South  3
04/01/2006  GOUNOT        Ontario-North  1
04/01/2006  GOUNOT        Quebec        3
04/01/2006  GOUNOT        Manitoba      7

41 record(s) selected.

db2 ->
```

2. A sample query using where clause and group by clause

```
Administrator: DB2 CLP - DB2COPY1 - db2
36 record(s) selected.

db2 => SELECT WEEK(SALES_DATE) AS WEEK, DAYOFWEEK(SALES_DATE) AS DAY_WEEK, SALES_PERSON, SALES AS UNITS_SOLD FROM SALES WHERE WEEK(SALES_DATE) = 13

WEEK    DAY_WEEK    SALES_PERSON    UNITS_SOLD
-----
13      4    LUCCHESSESI    3
13      4    LUCCHESSESI    1
13      4    LEE            2
13      6    LEE            2
13      4    LEE            3
13      4    LEE            5
13      4    GOUNDT        3
13      4    GOUNDT        1
13      4    GOUNDT        7
13      5    LUCCHESSESI    1
13      5    LUCCHESSESI    2
13      5    LUCCHESSESI    1
13      5    LEE            7
13      5    LEE            3
13      5    LEE            7
13      5    LEE            4
13      5    GOUNDT        2
13      5    GOUNDT        18
13      6    GOUNDT        1
13      6    LUCCHESSESI    1
13      6    LEE            14
13      6    LEE            3
13      6    LEE            7
13      6    LEE            3
13      6    GOUNDT        2
13      6    GOUNDT        1
13      7    LUCCHESSESI    3
13      7    LUCCHESSESI    1
13      7    LEE            8
13      7    LEE            -
13      7    LEE            8
13      7    LEE            9
13      7    GOUNDT        3
13      7    GOUNDT        1
13      7    GOUNDT        3
13      7    GOUNDT        7

36 record(s) selected.

db2 =>
```

```
Administrator: DB2 CLP - DB2COPY1 - db2

13      4    GOUNDT        7
13      5    LUCCHESSESI    1
13      5    LUCCHESSESI    2
13      5    LUCCHESSESI    1
13      5    LEE            7
13      5    LEE            3
13      5    LEE            7
13      5    LEE            4
13      5    GOUNDT        2
13      5    GOUNDT        18
13      6    GOUNDT        1
13      6    LUCCHESSESI    1
13      6    LEE            14
13      6    LEE            3
13      6    LEE            7
13      6    LEE            3
13      6    GOUNDT        2
13      6    GOUNDT        1
13      7    LUCCHESSESI    3
13      7    LUCCHESSESI    1
13      7    LEE            8
13      7    LEE            -
13      7    LEE            8
13      7    LEE            9
13      7    GOUNDT        3
13      7    GOUNDT        1
13      7    GOUNDT        3
13      7    GOUNDT        7

36 record(s) selected.

db2 => SELECT WEEK(SALES_DATE) AS WEEK, DAYOFWEEK(SALES_DATE) AS DAY_WEEK, SALES_PERSON, SUM(SALES) AS UNITS_SOLD FROM SALES WHERE WEEK(SALES_DATE) = 13 GROUP BY WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON ORDER BY WEEK, DAY_WEEK, SALES_PERSON

WEEK    DAY_WEEK    SALES_PERSON    UNITS_SOLD
-----
13      4    GOUNDT        11
13      4    LEE            10
13      4    LUCCHESSESI    4
13      5    GOUNDT        20
13      5    LEE            21
13      5    LUCCHESSESI    4
13      6    GOUNDT        4
13      6    LEE            29
13      6    LUCCHESSESI    1
13      7    GOUNDT        14
13      7    LEE            25
13      7    LUCCHESSESI    4

12 record(s) selected.

db2 =>
```

```
Administrator: DB2 CLP - DB2COPY1
L_VALUE DOUBLE, FOREIGN KEY (EXPLAIN_REQUESTER, EXPLAIN_TIME, SOURCE_NAME, SOURCE_SCHEMA, SOURCE_VERSION, EXPLAIN_LEVEL, STMTNO, SECTNO, OPERATOR_ID) REFERENCES EXPLAIN_OPERATOR ON DELETE CASCADE) ORGANIZE BY ROW
DB200001 The SQL command completed successfully.

COMMIT WORK
DB200001 The SQL command completed successfully.

C:\Program Files (x86)\IBM\SQLLIB\MISC>db2 -tvf D:\test.sql
select week(sales_date) as week, dayofweek(sales_date) as day_week, sales_person, sum(sales) as units_sold from sales where week(sales_date) = 13 group by week(sales_date), dayofweek(sales_date), sales_person order by week, day_week, sales_person
SQL0217W The statement was not executed as only Explain information requests are being processed. SQLSTATE=01604

C:\Program Files (x86)\IBM\SQLLIB\MISC>db2 set current explain mode no
DB200001 The SQL command completed successfully.

C:\Program Files (x86)\IBM\SQLLIB\MISC>db2exfmt -d sample -1 -o D:\test1.sql.exfmt
DB2 Universal Database Version 10.5, 5622-044 (c) Copyright IBM Corp. 1991, 2012
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Connecting to the Database.
Connect to Database Successful.
Binding package - Bind was Successful
Output is in D:\test1.sql.exfmt.
Executing Connect Reset -- Connect Reset was Successful.

C:\Program Files (x86)\IBM\SQLLIB\MISC>start notepad D:\test1.sql.exfmt
C:\Program Files (x86)\IBM\SQLLIB\MISC>
```

3. Query and exfmt

Query

SELECT WEEK(SALES_DATE) AS WEEK, DAYOFWEEK(SALES_DATE) AS DAY_WEEK, SUM(SALES) AS UNITS_SOLD FROM SALES GROUP BY ROLLUP (WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE)) ORDER BY WEEK, DAY_WEEK

Snapshot

```
test1.sql - Notepad
File Edit Format View Help

DB2 Universal Database Version 10.5, 5622-044 (c) Copyright IBM Corp. 1991, 2012
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 10.05.4
FORMATTED ON DB: SAMPLE
SOURCE_NAME: SQLC2K26
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2017-02-19-02.27.12.420000
EXPLAIN_REQUESTER: MISHI

Database Context:
*****
Parallelism: None
CPU Speed: 1.417033e-007
Comm Speed: 0
Buffer Pool size: 250
Sort Heap size: 256
Database Heap size: 600
Lock List size: 4096
Maximum Lock List: 22
Average Applications: 1
Locks Available: 28835

Package Context:
*****
SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

***** STATEMENT 1 SECTION 201 *****
QUERYNO: 32
QUERYTAG: CLP
Statement Type: Select
```

```
test1.sql - Notepad
File Edit Format View Help

Updatable: No
Deletable: No
Query Degree: 1

Original Statement:
-----
select
  week(sales_date) as week,
  dayofweek(sales_date) as day_week,
  sales_person,
  sum(sales) as units_sold
from
  sales
where
  week(sales_date) = 13
group by
  week(sales_date),
  dayofweek(sales_date),
  sales_person
order by
  week,
  day_week,
  sales_person

Optimized Statement:
-----
SELECT
  Q3.$C3 AS "WEEK",
  Q3.$C2 AS "DAY_WEEK",
  Q3.SALES_PERSON AS "SALES_PERSON",
  Q3.$C1 AS "UNITS_SOLD"
FROM
  (SELECT
    Q2.SALES_PERSON,
    SUM(Q2.SALES),
    Q2.$C2,
    Q2.$C1
  FROM
    (SELECT
      Q1.SALES_PERSON,
      "SYSFUN"."WEEK"(Q1.SALES_DATE),
```

```
test1.sql - Notepad
File Edit Format View Help

-----
+Q1.$RID$+Q1.SALES+Q1.SALES_PERSON
+Q1.SALES_DATE

Output Streams:
-----
2) To Operator #4

Estimated number of rows: 0.788462
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:
-----
+Q2.SALES+Q2.$C2+Q2.$C1+Q2.SALES_PERSON

Objects Used in Access Plan:
-----

Schema: MISHI
Name: SALES
Type: Table

Time of creation: 2017-02-17-17:28:41.023002
Last statistics update: 2017-02-18-22:38:18.197000
Number of columns: 4
Number of rows: 41
Width of rows: 34
Number of buffer pool pages: 1
Number of data partitions: 1
Distinct row values: No
Tablespace name: USERSPACE1
Tablespace overhead: 6.725000
Tablespace transfer rate: 0.080000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32
Table overflow record count: 0
Table Active Blocks: -1
Average Row Compression Ratio: 0
Percentage Rows Compressed: 0
```

IBM Graph

Graph Data store

- Sign up for IBM Bluemix at www.bluemix.net (Links to an external site.)
- Navigate the catalog for Data and Analytics section
- Click on IBM Graph service, create the service and follow the documentation to create a sample graph application using the API documentation: <https://ibm-graph-docs.ng.bluemix.net/api.html> (Links to an external site.)

According to IBM Bluemix website:

"IBM Graph is an easy-to-use, fully managed graph database service for storing and querying data points, their connections, and properties. IBM Graph offers an Apache TinkerPop3 compatible API and plugs into your Bluemix application seamlessly. This service can be used for building recommendation engines, analyzing social networks, and fraud detection."

The idea is to create object nodes for everything, which pertains some properties, be it a person, an organization, a place etc. represented as Vertices and the relationship between these vertices are called Edges which also exhibit some properties.

In the following document, we will see systematic instructions for creating a sample Graph database using POSTMAN, and IBM BLUEMIX GRAPH Service.

Pre-Requirements: POSTMAN desktop app, internet connection.

Signup on Bluemix, and search for IBM Graph service. Open "Service Credentials" and note the values.

Manage

Service Credentials

Connections

Service Credentials

Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

Service Credentials

New Credential +

KEY NAME	DATE CREATED	ACTIONS
Credentials-1	Feb 21, 2017 - 08:15:57	View Credentials

```
{
  "apiURL": "https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/g",
  "username": "c3b54935-38f4-4042-9419-33fe8382b392",
  "apiURI": "https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237",
  "password": "559af5fb-e45a-45f8-9b26-c12c93ecf9cf"
}
```

Now on POSTMAN , execute the following commands:

1. Get gds-token

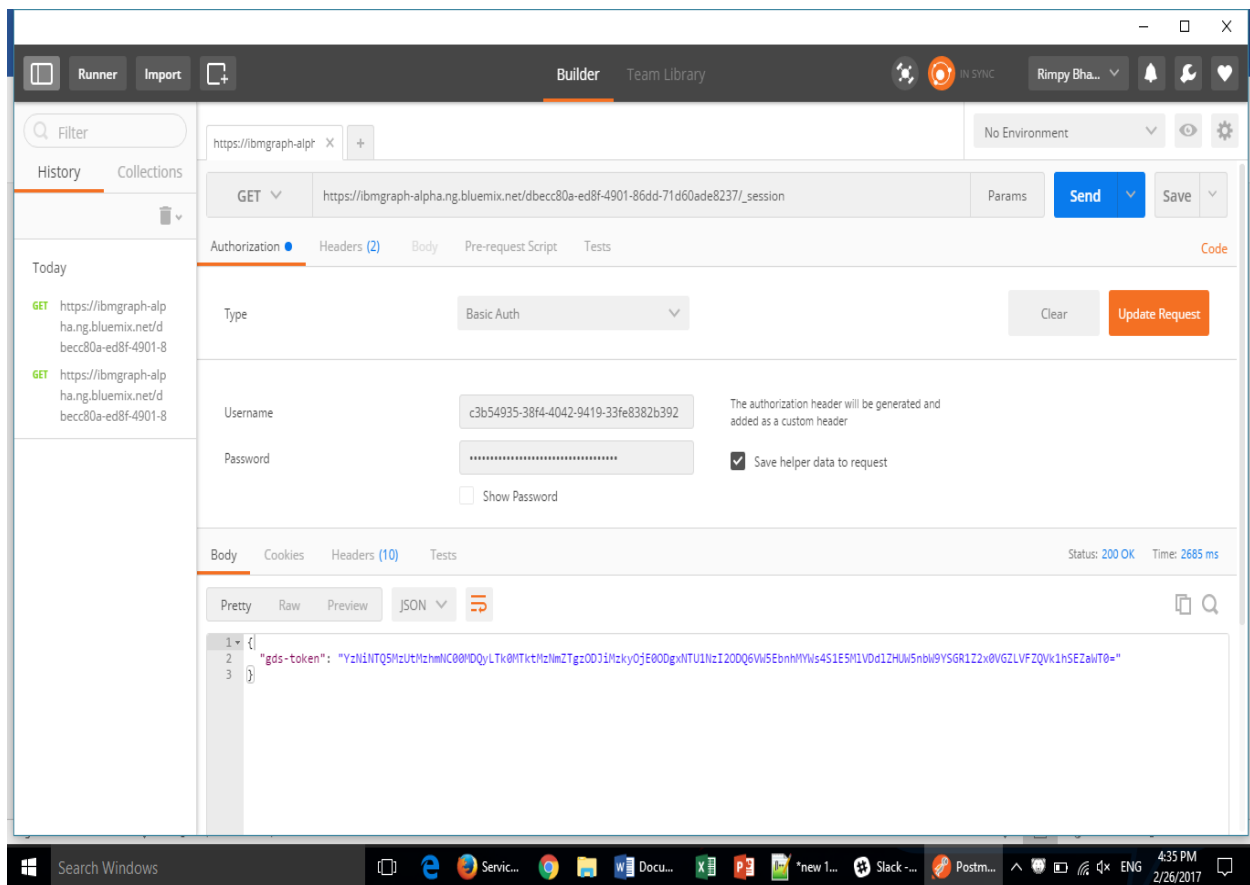
URL: https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/_session

Method: GET

Status : 200 OK

Response:

```
{
  "gds-token":
  "YzNiNTQ5MzUtMzhmNC00MDQyLTk0MTktMzNmZTgzODJiMzkyOjE0ODgxNTU1NzI2ODQ6VW5EbnhMYWw4S1E5
  MlVDDlZHUW5nbW9YSGR1Z2x0VGZLVFZQVk1hSEZaWT0="
}
```



2. Create a graph

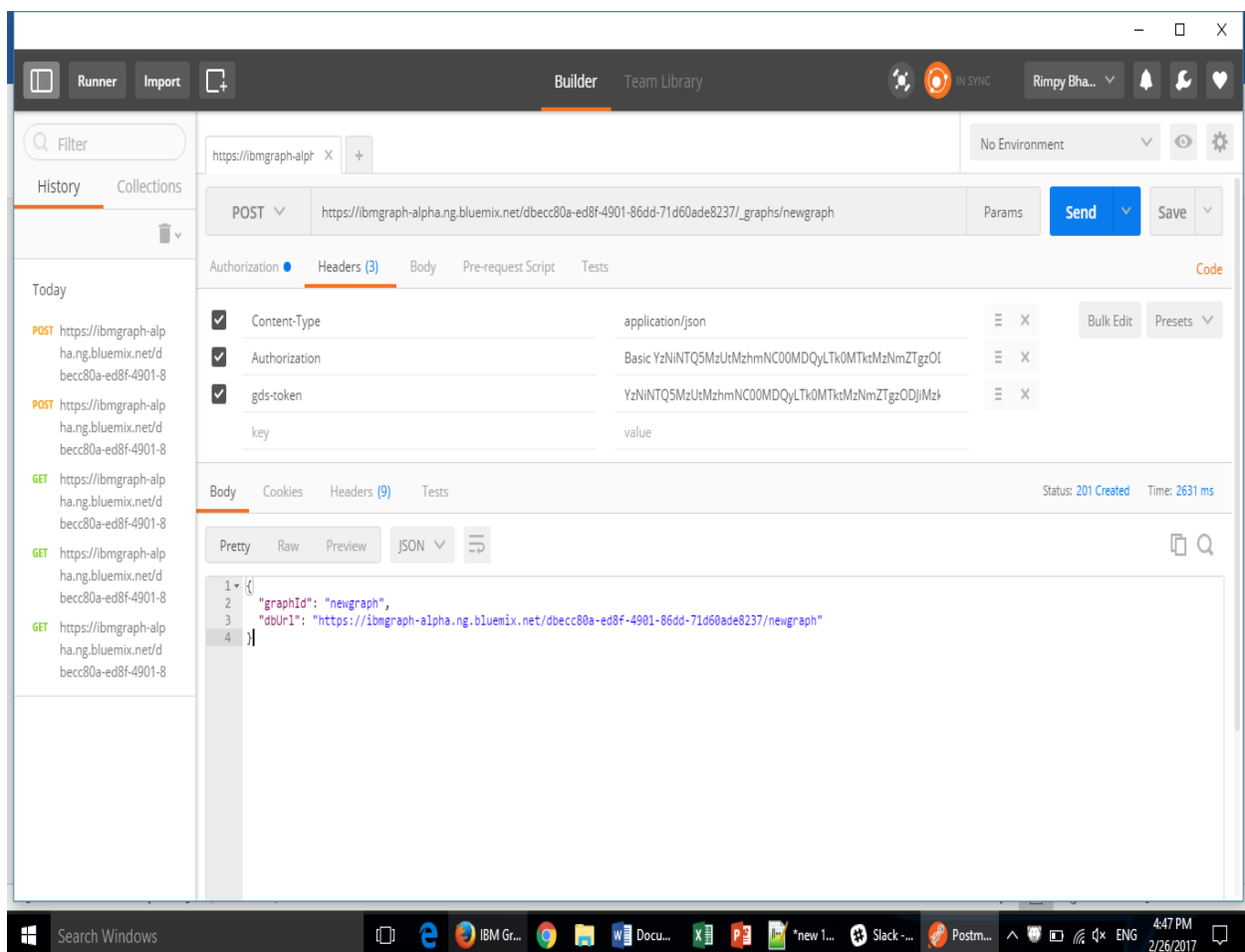
URL: https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/_graphs/bookgraph (where bookgraph is the name of the graph we wish to create)

Method: POST

Status: 201 Created

Response:

```
{  
  "graphId": "bookgraph",  
  "dbUrl": "https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/bookgraph"  
}
```



3. Get properties of the new graph created

URL: <https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/bookgraph/>

Method: GET

Status: 200 OK

Response:

```
{ "requestId": "afbefa6b-f56b-4d4d-a2e2-95b5c1e7b4cf", "status": { "message": "", "code": 200,
"attributes": {} }, "result": { "data": [ "StandardTitanGraph" ], "meta": {} } }
```

The screenshot shows a REST client interface with the following details:

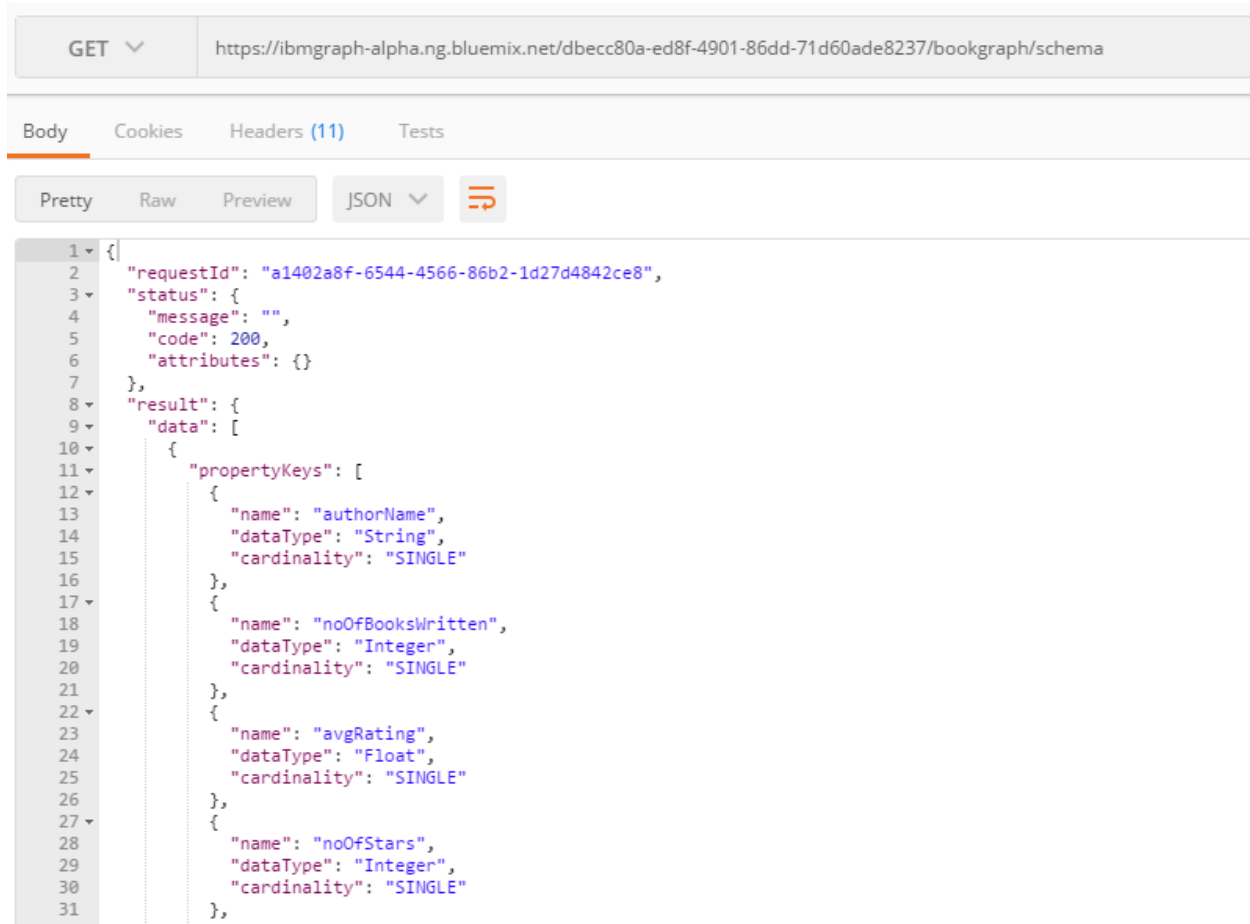
- Method:** GET
- URL:** <https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/bookgraph/>
- Type:** Basic Auth
- Username:** c3b54935-38f4-4042-9419-33fe8382b392
- Password:** [Redacted]
- Save helper data to request:** ☒
- Response (JSON):**

```
{
  "requestId": "a9f6696e-da9d-48fe-be90-d0067f258c1e",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      "StandardTitanGraph"
    ],
    "meta": {}
  }
}
```


4. Add Schema in POSTMAN

URL : <https://ibmgraph-alpha.ng.bluemix.net/dbecc80a-ed8f-4901-86dd-71d60ade8237/bookgraph/schema>

Method : POST



5. Add vertices and edges in Bluemix using Gremlin queries

```
def v1 = graph.addVertex("name", "Mr. Salman Rushdie", label, "author", "noOfBooksWritten", 10);

def v2 = graph.addVertex("firstName", "Amreen", "lastName", "Wahab", label, "reader", "username",
"amreenwahab", "email", "amreenwahab@gmail.com");

def v3 = graph.addVertex("bookName", "Midnight's Children", "price", 25.0, label, "book", "genre",
"Magic realism");

v2.addEdge("rate", v1, "noOfStars", 4);
```

```
v1.addEdge("writes", v3, "publishedYear", 1981)
```

```
v2.addEdge("buys", v3, "dateTime", "10 Dec 2016", "address1", "Avalon Street", "address2", "292",  
"city", "san jose", "state", "ca", "zip", 95129, "paymentMethod", "credit card")
```

Response:

The screenshot shows the IBM Graph console interface. The top navigation bar includes 'IBM Graph', 'Documentation', and 'Bluemix Dashboard'. The main header shows 'IBM Graph-e4' and 'bookgraph'. The left sidebar has 'Upload', 'Samples', and 'Resources'. The main area displays a query in a text editor and its response in a JSON viewer.

```
1 def v1 = graph.addVertex("name", "Mr. Salman Rushdie", label, "author", "noOfBooksWritten", 10);  
2 def v2 = graph.addVertex("firstName", "Amreen", "lastName", "Wahab", label, "reader", "username", "amreenwahab", "email",  
  "amreenwahab@gmail.com");  
3 def v3 = graph.addVertex("bookName", "Midnight's Children", "price", 25.0, label, "book", "genre", "Magic realism");  
4 v2.addEdge("rate", v1, "noOfStars", 4);  
5 v1.addEdge("writes", v3, "publishedYear", 1981)  
6 v2.addEdge("buys", v3, "dateTime", "10 Dec 2016", "address1", "Avalon Street", "address2", "292", "city", "san jose",  
  "state", "ca", "zip", 95129, "paymentMethod", "credit card")
```

```
{  
  "id": "36k-3c0-i6t-oe0a0",  
  "label": "buys",  
  "type": "edge",  
  "inVLabel": "book",  
  "outVLabel": "reader",  
  "inV": 40964328,  
  "outV": 4320,  
  "properties": {  
    "dateTime": "10 Dec 2016",  
    "address1": "Avalon Street",  
    "address2": "292",  
    "city": "san jose",  
    "state": "ca",  
    "zip": 95129,  
    "paymentMethod": "credit card"  
  }  
}
```

6. Get books read by a user using their first name :

```
def gt = graph.traversal();
```

```
gt.V().hasLabel("reader").has("firstName", "Rimpy").outE("buys").inV().hasLabel("book").path();
```

```

1 def gt = graph.traversal();
2 gt.V().hasLabel("reader").has("firstName", "Rimpy").outE("buys").inV().hasLabel("book").path();

```



```

1 ▾ [
2 ▾ {
3 ▾ "labels": [
4   [],
5   [],
6   []
7 ],
8 ▾ "objects": [
9 ▾ {
10  "id": 4208,
11  "label": "reader",
12  "type": "vertex",
13 ▾ "properties": {
14 ▾ "firstName": [
15   {
16    "id": "479-701-21"

```

Filter:



Vertices: 2

7. Find the author of book named “The Lost Symbol”

```

1 def gt = graph.traversal();
2 gt.V().hasLabel("book").has("bookName", "The Lost Sysmbol").inE("writes").outV().hasLabel("author").path();

```



Graph: bookgraph

```
def gt = graph.traversal();gt.V().hasLabel("book").has("bookName", "The Lost Sysmbol").inE("writes")
```



```

46 ▾ {
47   "id": 4120,
48   "label": "author",
49   "type": "vertex",
50 ▾ "properties": {
51 ▾ "name": [
52   {
53     "id": "16r-36g-27t1",
54     "value": "Dan Brown"
55   }
56 ],
57 ▾ "noOfBooksWritten": [
58   {
59     "id": "1kz-36g-dfp",
60     "value": 15
61   }
62 ]

```



8. Find the author who has written the book read by a reader.

```
1 def gt = graph.traversal();
2 gt.V().hasLabel("reader").has("firstName", "Sahithi").outE("buys").inV().hasLabel("book").has("bookName", "The Lost
   Symbol").inE("writes").outV().hasLabel("author").path();
```

Graph: bookgraph

```
def gt = graph.traversal();gt.V().hasLabel("reader").has("firstName", "Sahithi").outE("buys").inV().
```

```
1  ▾ [
2  ▾ {
3  ▾   "labels": [
4      [],
5      [],
6      [],
7      [],
8      []
9  ],
10 ▾   "objects": [
11 ▾     {
12         "id": 8216,
13         "label": "reader",
14         "type": "vertex",
15         "properties": {
16           "firstName": [
```




9. Get the rating for the book “Sahithi” reads traversing through the author.

```
1 def gt = graph.traversal();
2 gt.V().hasLabel("reader").has("firstName", "Sahithi").outE("buys").inV().hasLabel("book").has("bookName", "The Lost
   Symbol").inE("writes").outV().hasLabel("author").inE("rate").outV().hasLabel("reader").path();
```

Graph: bookgraph

```
def gt = graph.traversal();gt.V().hasLabel("reader").has("firstName", "Sahithi").outE("buys").inV().
```

```
1  [
2  {
3    "labels": [
4      [],
5      [],
6      [],
7      [],
8      [],
9      [],
10     []
11   ],
12   "objects": [
13     {
14       "id": 8216,
15       "label": "reader",
16       "name": "Sahithi"
```



Conclusion:

Therefore, using IBM Bluemix, we can work on IBM graph using REST API's and Gremlin queries. Graph databases are used in complex scenarios such as big data analysis, social media and continuously flowing information like stock etc. The node-oriented queries allow the data to be retrieved as indexes instead of the tabular manner of traditional RDBMS. A real time scenario example is credit card fraudulency, where the pattern can help us reach a point of start of abnormal (spike in purchases) activities.