

Team#5 ASSIGNMENT# 1

Database Team based Assignment

Ishwarya Varadarajan 011549473
 Mridula Krishnamurthy 010830209
 Sowmya Viswanathan 011432668
 Vishal Praanesh 011485175

SQLITE

PURCHASE ORDER MANAGEMENT SYSTEM:

The system stores the purchase order in a table that is created when a buyer(company) orders for commodities from its various vendors

The system has three tables namely PURCHASE_ORDR, VENDOR & PRODUCT.

PURCHASE_ORDR contains the details of the purchase order details that is created every time an order is placed to a vendor

VENDOR table contains the details of every vendor the company does business with
 PRODUCT table contains the details of the products for which orders are placed

VIEW PROD_PRICE contains the columns such as UNIT_COST, ORDER_QTY to calculate and update total price of commodities based on the PURCHASE_ORDR tables

The above functionality is done automatically when a new row is inserted in the table PURCHASE_ORDR with the help of a TRIGGER TRG_PO which will update the PURCHASE_ORDR table with the TOTAL_COST with (UNIT_COST * ORDER_QTY) from PROD_PRICE VIEW

To create a TABLE named PURCHASE_ORDR:

```
CREATE TABLE PURCHASE_ORDR (
    PO_NUMBER INT PRIMARY KEY NOT NULL,
    ITEM_NUMBER INT NOT NULL,
    ITEM_DESC CHAR (20) NOT NULL,
    VENDOR_NUMBER INT NOT NULL,
```

```

ORDER_DATE TEXT NOT NULL,
ORDER_QTY INT NOT NULL,
TOTAL_COST REAL);

```

To insert data into the table

```

INSERT INTO PURCHASE_ORDR (PO_NUMBER, ITEM_NUMBER, ITEM_DESC,
VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST)
VALUES (1, 5007, 'X's Digital Clock', 10005, '2017-02-17', 100, '0');

```

The screenshot shows the SQLite Manager interface with the database 'POM.db' selected. The left sidebar lists tables: Master Table (1), Tables (3) including PRODUCT, PURCHASE_ORDR, and VENDOR, Views (0), Indexes (3), and Triggers (0). The PURCHASE_ORDR table is currently selected. The main area shows the table structure and data.

Table Structure:

PO_NUMBER	ITEM_NUMBER	ITEM_DESC	VENDOR_NUMBER	ORDER_DATE	ORDER_QTY	TOTAL_COST
1	5001	iPhone 7 Plus	10001	2017-02-12	5	0
2	5002	iPhone 7	10001	2017-02-13	10	0
3	5003	Google Pixel	10002	2017-02-12	6	0
4	5004	Google Pixel XL	10002	2017-02-12	6	0
5	5005	Romba Vcm Clnr	10003	2017-02-10	5	0
6	5006	Mto G4 Plus	10004	2017-02-11	7	0

SQL Query:

```
SELECT * FROM PURCHASE_ORDR;
```

Status: Last Error: not an error

To create a TABLE called VENDOR:

```
CREATE TABLE VENDOR (
    VENDOR_ID INT NOT NULL,
    VENDOR_NAME CHAR (30) NOT NULL,
    VENDOR_ADDR CHAR (30) NOT NULL,
    VENDOR_STATUS CHAR (1) NOT NULL);
```

To insert data into the table

```
INSERT INTO VENDOR (VENDOR_ID, VENDOR_NAME, VENDOR_ADDR,
VENDOR_STATUS) VALUES (10001, 'APPLE INC.', 'CUPERTINO CA', 'Y');
```

The screenshot shows a database management tool window. On the left is a tree view of database objects:

- Master Table (1): sqlite_master
- Tables (3):
 - PRODUCT: PRODUCT_ID, PROD_DESC, UNIT_COST, VENDOR_ID
 - PURCHASE_ORDER: PO_NUMBER, ITEM_NUMBER, ITEM_DESC, VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST
 - VENDOR: VENDOR_ID, VENDOR_NAME, VENDOR_ADDR, VENDOR_STATUS
- Views (0)
- Indexes (3): sqlite_autoindex_PRODUCT_1, sqlite_autoindex_PURCHASE_1

The central area has tabs: Structure, Browse & Search, Execute SQL (which is selected), DB Settings, and Import Wizard.

The Execute SQL tab contains the following content:

```
Enter SQL
SELECT * FROM VENDOR;
```

Below the SQL input is a status bar with Run SQL, Actions, and Last Error: not an error.

The bottom half of the window displays the VENDOR table data in a grid:

VENDOR_ID	VENDOR_NAME	VENDOR_ADDR	VENDOR_STATUS
10001	APPLE INC.	CUPERTINO CA	Y
10002	ALPHABET INC.	MOUNTAIN VIEW CA	Y
10003	ROMBA INC.	MOUNTAIN VIEW CA	Y
10004	MOTOROLA INC.	SCHAUMBURG IL	Y

To create a TABLE called PRODUCT:

```
CREATE TABLE PRODUCT (
    PRODUCT_ID INT NOT NULL,
    PROD_DESC CHAR (30) NOT NULL,
    UNIT_COST REAL NOT NULL,
    VENDOR_ID INT NOT NULL);
```

To insert data into the table

```
INSERT INTO PRODUCT (PRODUCT_ID, PROD_DESC, UNIT_COST,
VENDOR_ID) VALUES (5001, 'iPhone 7 Plus', 700.00, 10001);
```

The screenshot shows a SQLite database interface with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save, Import, Export, Find, etc.), a search bar, and navigation buttons.
- Database List:** Shows the database name "POM.db" and a tree view of tables:
 - Master Table (1):** sqlite_master
 - Tables (3):**
 - PRODUCT:** Contains columns: PRODUCT_ID, PROD_DESC, UNIT_COST, VENDOR_ID.
 - PURCHASE_ORDR:** Contains columns: PO_NUMBER, ITEM_NUMBER, ITEM_DESC, VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST.
 - VENDOR:** Contains columns: VENDOR_ID, VENDOR_NAME, VENDOR_ADDR, VENDOR_STATUS.
 - Views (0)**
 - Indexes (3):** sqlite_autoindex_PRODUCT_1, sqlite_autoindex_PURCHASE_OR..., sqlite_autoindex_VENDOR_1
 - Triggers (0)**
- SQL Editor:** A panel titled "Enter SQL" containing the query: "SELECT * FROM PRODUCT;".
- Result Grid:** A table displaying the data from the PRODUCT table:

PRODUCT_ID	PROD_DESC	UNIT_COST	VENDOR_ID
5001	iPhone 7 Plus	700	10001
5002	iPhone 7	600	10001
5003	Google Pixel	600	10002
5004	Google Pixel XL	700	10002
5005	Romba Vcm Clnr	250	10003
5006	Mto G4 Plus	200	10004

SELECT QUERY WITH A CONDITION

Screenshot of a database management tool interface showing a query execution process.

The left sidebar displays the database structure:

- Master Table (1)**: sqlite_master
- Tables (3)**:
 - PRODUCT**: PRODUCT_ID, PROD_DESC, UNIT_COST, VENDOR_ID
 - PURCHASE_ORDR**: PO_NUMBER, ITEM_NUMBER, ITEM_DESC, VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST
 - VENDOR**: VENDOR_ID, VENDOR_NAME, VENDOR_ADDR, VENDOR_STATUS
- Views (1)**: PROD_PRICE: ITEM_NUMBER, ITEM_DESC, UNIT_COST, ORDER_QTY, COST

The main area shows the SQL query being run:

```
SELECT * FROM PURCHASE_ORDR WHERE ORDER_DATE > '2017-02-11';
```

The results are displayed in a table:

PO_NUMBER	ITEM_NUMBER	ITEM_DESC	VENDOR_NUMBER	ORDER_DATE	ORDER_QTY	TOTAL_COST
1	5001	iPhone 7 Plus	10001	2017-02-12	5	3500
2	5002	iPhone 7	10001	2017-02-13	10	6000
3	5003	Google Pixel	10002	2017-02-12	6	3600
4	5004	Google Pixel XL	10002	2017-02-12	6	4200

A query to LEFT JOIN a table with two or more tables

Table PURCHASE_ORDR is joined completely with tables VENDOR & PRODUCT based on PRODUCT_ID. Matching records based on the condition are selected for the view.

The screenshot shows a database management tool with the following interface elements:

- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Find, etc.), a search bar, and navigation buttons.
- Header:** Shows the database name "POM.db" and tabs for "Structure", "Browse & Search", "Execute SQL" (which is selected), "DB Settings", and "Import Wizard".
- Left Panel (Object Browser):**
 - Master Table (1): sqlite_master
 - Tables (3):
 - PRODUCT: PRODUCT_ID, PROD_DESC, UNIT_COST, VENDOR_ID
 - PURCHASE_ORDR: PO_NUMBER, ITEM_NUMBER, ITEM_DESC, VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST
 - VENDOR: VENDOR_ID, VENDOR_NAME, VENDOR_ADDR, VENDOR_STATUS
 - Views (0)
 - Indexes (3): sqlite_autoindex_PRODUCT_1, sqlite_autoindex_PURCHASE_OR..., sqlite_autoindex_VENDOR_1
 - Triggers (0)
- Central Area:**
 - Enter SQL:** A text input field containing the following SQL query:


```
SELECT a.PO_NUMBER, a.ITEM_NUMBER, a.ITEM_DESC, a.VENDOR_NUMBER, a.ORDER_DATE, a.ORDER_QTY, a.TOTAL_COST,
b.VENDOR_NAME, c.UNIT_COST
from PURCHASE_ORDR a
left join VENDOR b
on a.VENDOR_NUMBER = b.VENDOR_ID
left join PRODUCT c
on a.ITEM_NUMBER = c.PRODUCT_ID;
```
 - Run SQL:** A button to execute the query.
 - Last Error:** A message indicating "not an error".
 - Result Grid:** A table displaying the query results. The columns are: PO_NUMBER, ITEM_NUMBER, ITEM_DESC, VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST, VENDOR_NAME, and UNIT_COST. The data is as follows:

PO_NUMBER	ITEM_NUMBER	ITEM_DESC	VENDOR_NUMBER	ORDER_DATE	ORDER_QTY	TOTAL_COST	VENDOR_NAME	UNIT_COST
1	5001	iPhone 7 Plus	10001	2017-02-12	5	0	APPLE INC.	700
2	5002	iPhone 7	10001	2017-02-13	10	0	APPLE INC.	600
3	5003	Google Pixel	10002	2017-02-12	6	0	ALPHABET INC.	600
4	5004	Google Pixel XL	10002	2017-02-12	6	0	ALPHABET INC.	700
5	5005	Romba Vcm Clnr	10003	2017-02-10	5	0	ROMBA INC.	250
6	5006	Mto G4 Plus	10004	2017-02-11	7	0	MOTOROLA INC.	200

To create a VIEW named PROD_PRICE

The screenshot shows a database interface for a SQLite database named POM.db. On the left, the database structure is displayed in a tree view. Under 'Tables (3)', there are three tables: PRODUCT, PURCHASE_ORDR, and VENDOR. Under 'Views (1)', there is a single view named PROD_PRICE. The central area contains an SQL editor window titled 'Enter SQL' with the following query:

```
CREATE VIEW PROD_PRICE AS SELECT a.ITEM_NUMBER, a.ITEM_DESC, b.UNIT_COST, a.ORDER_QTY, (a.ORDER_QTY * b.UNIT_COST) AS COST FROM PURCHASE_ORDR a
LEFT JOIN PRODUCT b
ON a.ITEM_NUMBER = b.PRODUCT_ID;
```

Below the SQL editor, there are buttons for 'Run SQL', 'Actions', and 'Last Error: not an error'.

The screenshot shows the same database interface after running the SQL query. The 'Enter SQL' window now displays the following query:

```
SELECT * FROM PROD_PRICE;
```

Below the SQL editor, the results of the query are shown in a table:

ITEM_NUMBER	ITEM_DESC	UNIT_COST	ORDER_QTY	COST
5001	iPhone 7 Plus	700	5	3500
5002	iPhone 7	600	10	6000
5003	Google Pixel	600	6	3600
5004	Google Pixel XL	700	6	4200
5005	Romba Vcm Clnr	250	5	1250
5006	Mto G4 Plus	200	7	1400

Below the table, there are buttons for 'Run SQL', 'Actions', and 'Last Error: not an error'.

To UPDATE TABLE PURCHASE_ORDR

```
UPDATE PURCHASE_ORDR SET TOTAL_COST = (SELECT COST FROM
PROD_PRICE WHERE PROD_PRICE.ITEM_NUMBER =
PURCHASE_ORDR.ITEM_NUMBER);
```

The screenshot shows a database interface with the following details:

- Database:** POM.db
- Tables:**
 - Master Table (1): sqlite_master
 - Tables (3):
 - PRODUCT: PRODUCT_ID, PROD_DESC, UNIT_COST, VENDOR_ID
 - PURCHASE_ORDR: PO_NUMBER, ITEM_NUMBER, ITEM_DESC, VENDOR_NUMBER, ORDER_DATE, ORDER_QTY, TOTAL_COST
 - VENDOR: VENDOR_ID, VENDOR_NAME, VENDOR_ADDR, VENDOR_STATUS
 - Views (0)
 - Indexes (3): sqlite_autoindex_PRODUCT_1, sqlite_autoindex_PURCHASE_ORDR_1, sqlite_autoindex_VENDOR_1
 - Triggers (0)
- SQL Editor:** Enter SQL: `SELECT * FROM PURCHASE_ORDR;`
- Run SQL:** Run SQL
- Actions:** Actions
- Last Error:** not an error
- Table Data:** The PURCHASE_ORDR table contains the following data:

PO_NUMBER	ITEM_NUMBER	ITEM_DESC	VENDOR_NUMBER	ORDER_DATE	ORDER_QTY	TOTAL_COST
1	5001	iPhone 7 Plus	10001	2017-02-12	5	0
2	5002	iPhone 7	10001	2017-02-13	10	0
3	5003	Google Pixel	10002	2017-02-12	6	0
4	5004	Google Pixel XL	10002	2017-02-12	6	0
5	5005	Romba Vcm Clnr	10003	2017-02-10	5	0
6	5006	Mto G4 Plus	10004	2017-02-11	7	0

To create a trigger named TRG_PO

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure for 'POM.db'. It includes a 'Tables' node containing 'PRODUCT', 'PURCHASE_ORDR', and 'VENDOR'. A 'Views' node contains 'PROD_PRICE'. An 'Indexes' node contains three auto-indexes. A 'Triggers' node contains one trigger named 'TRG_PO'. The main pane is titled 'Enter SQL' and contains the following T-SQL code:

```

CREATE TRIGGER TRG_PO AFTER INSERT ON PURCHASE_ORDER
UPDATE PURCHASE_ORDER SET TOTAL_COST = (SELECT COST FROM PROD_PRICE WHERE PROD_PRICE.ITEM_NUMBER = PURCHASE_ORDER.ITEM_NUMBER);
END;

```

Below the code, there are buttons for 'Run SQL', 'Actions', and 'Last Error: not an error'.

Automatically updates the PURCHASE_ORDER table's column's column by calculating the TOTAL_COST as per the LOGIC set inside the BEGIN and END statements in the TRIGGER query.

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure for 'POM.db'. It includes a 'Tables' node containing 'PRODUCT', 'PURCHASE_ORDER', and 'VENDOR'. A 'Views' node contains 'PROD_PRICE'. An 'Indexes' node contains three auto-indexes. A 'Triggers' node contains one trigger named 'TRG_PO'. The main pane is titled 'Enter SQL' and contains the following T-SQL code:

```

SELECT a.PO_NUMBER, a.ITEM_NUMBER, a.ITEM_DESC, a.VENDOR_NUMBER, a.ORDER_DATE, a.ORDER_QTY, a.TOTAL_COST,
b.VENDOR_NAME, c.UNIT_COST
from PURCHASE_ORDER a
left join VENDOR b
on a.VENDOR_NUMBER = b.VENDOR_ID
left join PRODUCT c
on a.ITEM_NUMBER = c.PRODUCT_ID;

```

Below the code, there are buttons for 'Run SQL', 'Actions', and 'Last Error: not an error'. To the right of the code, a grid displays the data from the PURCHASE_ORDER table:

PO_NUMBER	ITEM_NUMBER	ITEM_DESC	VENDOR_NUMBER	ORDER_DATE	ORDER_QTY	TOTAL_COST	VENDOR_NAME	UNIT_COST
1	5001	iPhone 7 Plus	10001	2017-02-12	5	3500	APPLE INC.	700
2	5002	iPhone 7	10001	2017-02-13	10	6000	APPLE INC.	600
3	5003	Google Pixel	10002	2017-02-12	6	3600	ALPHABET INC.	600
4	5004	Google Pixel XL	10002	2017-02-12	6	4200	ALPHABET INC.	700
5	5005	Romba Vcm Clrr	10003	2017-02-10	5	1250	ROMBA INC.	250
6	5006	Mto G4 Plus	10004	2017-02-11	7	1400	MOTOROLA INC.	200
7	5007	Xs Digital Clock	10005	2017-02-17	100	10000	X INC.	100
8	5008	Contigo Wtr Bottles	10006	2017-02-17	50	500	CONTIGO INC.	10

To delete a row from a table

The screenshot shows the SQL Pro for SQLite interface. The left pane displays the database structure of POM.db, including tables like PRODUCT, PURCHASE_ORDER, and VENDOR. The right pane shows the results of a DELETE query:

```
DELETE FROM VENDOR WHERE VENDOR_ID = 10007;
```

The results table shows the following data:

VENDOR_ID	VENDOR_NAME	VENDOR_ADDR	VENDOR_STATUS
10001	APPLE INC.	CUPERTINO CA	Y
10002	ALPHABET INC.	MOUNTAIN VIEW CA	Y
10003	ROMBA INC.	MOUNTAIN VIEW CA	Y
10004	MOTOROLA INC.	SCHAUMBURG IL	Y
10005	X INC.	SANTA CLARA CA	Y
10007	CONTIGO INC.	SAN JOSE CA	Y

The screenshot shows the SQL Pro for SQLite interface. The left pane displays the database structure of POM.db. The right pane shows the results of a SELECT query:

```
SELECT * FROM VENDOR;
```

The results table shows the following data:

VENDOR_ID	VENDOR_NAME	VENDOR_ADDR	VENDOR_STATUS
10001	APPLE INC.	CUPERTINO CA	Y
10002	ALPHABET INC.	MOUNTAIN VIEW CA	Y
10003	ROMBA INC.	MOUNTAIN VIEW CA	Y
10004	MOTOROLA INC.	SCHAUMBURG IL	Y
10005	X INC.	SANTA CLARA CA	Y

DB2 Express C

In this part of the assignment, we create a Sample database using ‘db2sampl’ command, which creates a database called ‘SAMPLE’, and adds some tables to it along with values filling the tables. Then, we run a sample query for one of these tables using ‘where’ clause and ‘group by’. Then, we generate the query explain plan using the ‘db2exfmt’ tool.

Creating Sample database using ‘db2sampl’

```
db2inst1@mridula: ~
db2inst1@mridula:~$ db2sampl

Creating database "SAMPLE"...
Existing "SAMPLE" database found...
The "-force" option was not specified...
Attempt to create the database "SAMPLE" failed.
SQL1005N The database alias "SAMPLE" already exists in either the local
database directory or system database directory.

'db2sampl' processing complete.

db2inst1@mridula:~$
```

We are querying the ‘dept’ table within the database, which has the following rows and columns.

```
db2inst1@mridula: ~
db2 => select * from dept

DEPTNO DEPTNAME                      MGRNO ADMRDEPT LOCATION
----- -----
A00    SPIFFY COMPUTER SERVICE DIV.   000010 A00      -
B01    PLANNING                       000020 A00      -
C01    INFORMATION CENTER             000030 A00      -
D01    DEVELOPMENT CENTER              -         A00      -
D11    MANUFACTURING SYSTEMS          000060 D01      -
D21    ADMINISTRATION SYSTEMS        000070 D01      -
E01    SUPPORT SERVICES               000050 A00      -
E11    OPERATIONS                     000090 E01      -
E21    SOFTWARE SUPPORT                000100 E01      -
F22    BRANCH OFFICE F2                 -         E01      -
G22    BRANCH OFFICE G2                 -         E01      -
H22    BRANCH OFFICE H2                 -         E01      -
I22    BRANCH OFFICE I2                 -         E01      -
J22    BRANCH OFFICE J2                 -         E01      -

14 record(s) selected.

db2 =>
```

The query we are using is

“select deptname, count(*) from dept where admrdept = ‘E01’ group by deptname”.

The result is shown in the below capture.

```
db2 => select * from dept where ADMRDEPT = E01 group by DEPTNAME
SQL0206N "E01" is not valid in the context where it is used. SQLSTATE=42703
db2 => SELECT * FROM DEPT where ADMRDEPT = 'E01' group by DEPTNAME
SQL0119N An expression starting with "DEPTNO" specified in a SELECT clause,
HAVING clause, or ORDER BY clause is not specified in the GROUP BY clause or
it is in a SELECT clause, HAVING clause, or ORDER BY clause with a column
function and no GROUP BY clause is specified. SQLSTATE=42803
db2 => select deptname, count(*) from dept where admrdept = 'E01' group by deptn
ame

DEPTNAME          2
-----
BRANCH OFFICE F2      1
BRANCH OFFICE G2      1
BRANCH OFFICE H2      1
BRANCH OFFICE I2      1
BRANCH OFFICE J2      1
OPERATIONS           1
SOFTWARE SUPPORT     1

7 record(s) selected.

db2 => |
```

Then, we set the mode in the database to explain mode, so that the result of the query is converted to an explain plan. To generate this, we store the above query in a text file called ‘query2.sql’, and use this as an input to generate the plan.

```
db2inst1@mridula: ~
db2inst1@mridula:~$ db2 connect to SAMPLE

Database Connection Information

Database server      = DB2/LINUXX8664 11.1.1.1
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

db2inst1@mridula:~$ db2 set current explain mode explain
DB20000I  The SQL command completed successfully.
db2inst1@mridula:~$ db2 -tvf query2.sql
select deptname, count(*) from dept where admrdept = 'E01' group by deptname
SQL0217W  The statement was not executed as only Explain information requests
are being processed.  SQLSTATE=01604

db2inst1@mridula:~$
```

Then, to convert this into a human readable format, we use ‘db2exfmt’ tool. We give the ‘SAMPLE’ database as the input, and store the output in the file ‘query2.sql.exfmt’, where as the other options are taken as default.

```
db2inst1@mridula:~/sqllib/misc
db2inst1@mridula:~/sqllib/misc$ cd ./misc
db2inst1@mridula:~/sqllib/misc$ db2exfmt -d SAMPLE -1 -o query2.sql.exfmt
DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM Corp. 1991, 2015
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Connecting to the Database.
Connect to Database Successful.
Output is in query2.sql.exfmt.
Executing Connect Reset -- Connect Reset was Successful.
db2inst1@mridula:~/sqllib/misc$
```

When the output file 'query2.sql.exfmt' is viewed, we can see the following in the explain plan file.

```

query2.sql.exfmt (~/sqllib/misc) - gedit
Open Save Undo Redo Cut Copy Paste Find Replace
query2.sql.exfmt x
DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM Corp. 1991, 2015
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

*****
EXPLAIN INSTANCE *****
DB2_VERSION: 11.01.1
FORMATTED ON DB: SAMPLE
SOURCE_NAME: SQLC2026
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2017-02-21-17.05.27.359372
EXPLAIN_REQUESTER: DB2INST1

Database Context:
-----
Parallelism: None
CPU Speed: 8.384110e-07
Comm Speed: 0
Buffer Pool size: 1000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 4096
Maximum Lock List: 10
Average Applications: 1
Locks Available: 13107

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

```

```

query2.sql.exfmt (~/sqllib/misc) - gedit
Original Statement:
-----
select
  deptname,
  count(*)
from
  dept
where
  admrdept = 'E01'
group by
  deptname

Optimized Statement:
-----
SELECT
  Q3.DEPTNAME AS "DEPTNAME",
  Q3.$C1
FROM
  (SELECT
    Q2.DEPTNAME,
    COUNT(*)
  FROM
    (SELECT
      Q1.DEPTNAME
    FROM
      DB2INST1.DEPARTMENT AS Q1
    WHERE
      (Q1.ADMRDEPT = 'E01')
    ) AS Q2
  GROUP BY
    Q2.DEPTNAME
  ) AS Q3

Access Plan:
-----

```

query2.sql.exfmt (~/sqlllib/misc) - gedit

```

Access Plan:
-----
      Total Cost:          6.88206
      Query Degree:        1

      Rows
      RETURN
      ( 1 )
      Cost
      I/O
      |
      7
      GRBY
      ( 2 )
      6.87907
      1
      |
      7
      TBSCAN
      ( 3 )
      6.87737
      1
      |
      7
      SORT
      ( 4 )
      6.87462
      1
      |
      7
      FETCH
      ( 5 )
      6.86813
      1
      /-----+----\
      7           1A
  
```

query2.sql.exfmt (~/sqlllib/misc) - gedit

```

Plan Details:
-----
1) RETURN: (Return Result)
Cumulative Total Cost: 6.88206
Cumulative CPU Cost: 91911.2
Cumulative I/O Cost: 1
Cumulative Re-Total Cost: 0.0299455
Cumulative Re-CPU Cost: 35717
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 6.87649
Estimated Bufferpool Buffers: 0

Arguments:
-----
BLDLEVEL: (Build level)
DB2 v11.1.1.1 : s1612051900
HEAPUSE : (Maximum Statement Heap Usage)
96 Pages
PLANID : (Access plan identifier)
c9b5592c2047d936
PRETIME: (Statement prepare time)
150 milliseconds
SEMEVID : (Semantic environment identifier)
431f78d03d9bb07e
STMTH HEAP: (Statement heap size)
8192
STMTID : (Normalized statement identifier)
e3275ff38f310f16

Input Streams:
-----
7) From Operator #2
  
```

query2.sql.exfmt (~/.sqllib/mslc) - gedit

Objects Used in Access Plan:

```
Schema: DB2INST1
Name: DEPT
Type: Alias (reference only)

Schema: DB2INST1
Name: XDEPT3
Type: Index
    Time of creation: 2017-02-20-13.20.16.060109
    Last statistics update: 2017-02-20-13.47.32.010164
    Number of columns: 1
    Number of rows: 14
    Width of rows: -1
    Number of buffer pool pages: 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
    Index clustering statistic: 100.000000
    Index leaf pages: 1
    Index tree levels: 1
    Index full key cardinality: 3
    Index first key cardinality: 3
    Index first 2 keys cardinality: -1
    Index first 3 keys cardinality: -1
    Index first 4 keys cardinality: -1
    Index sequential pages: 0
    Index page density: 0
    Index avg sequential pages: 0
    Index avg gap between sequences: 0
```

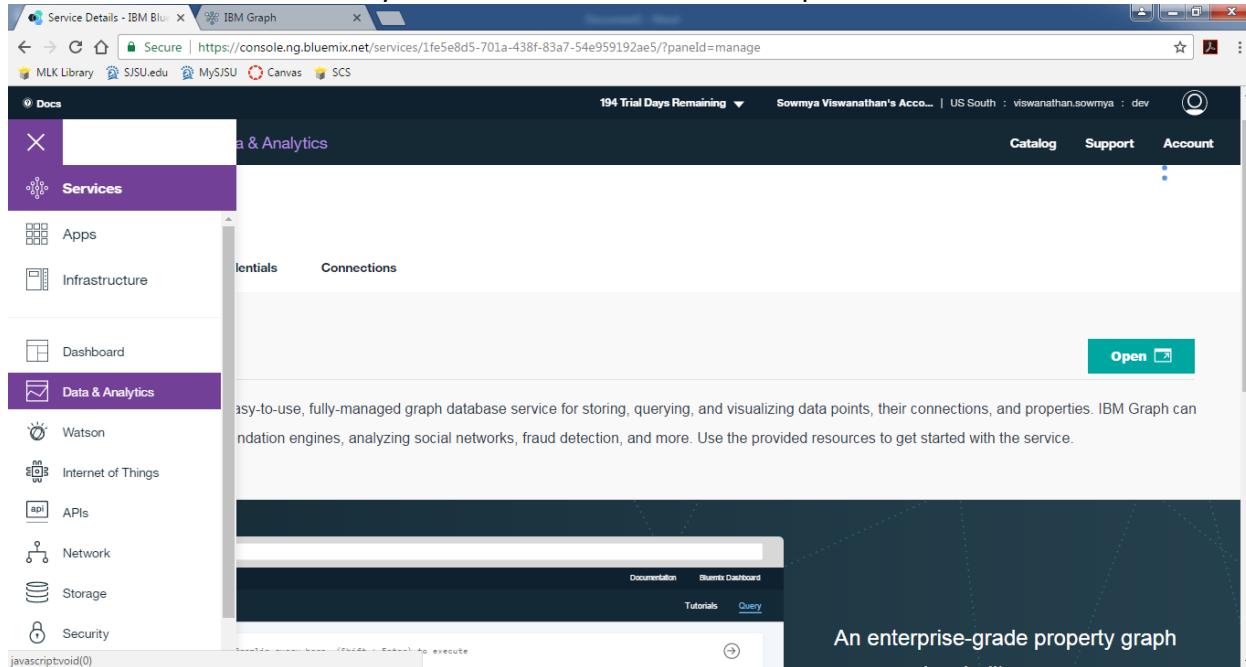
Plain Text ▾ Tab Width: 8 ▾ Ln 9, Col 27 INS

IBM GRAPH DB

IBM Graph provides an HTTP API for manipulating and querying graphs. Requests and responses use the JSON format.

Graph DB set up and UI features:

- Select ‘Data and Analytics’ from the Menu on the left pane.



- Select IBM Graph service which has been set up

The screenshot shows the IBM Bluemix Data & Analytics service management interface. At the top, there's a navigation bar with tabs for 'Docs', 'IBM Bluemix Data & Analytics', 'Catalog', 'Support', and 'Account'. Below the navigation bar, there are four main tabs: 'Services', 'Data Connections', 'Analytics', and 'Exchange'. The 'Services' tab is selected. A banner below the tabs reads: 'Explore, analyze, and visualize your data with interactive notebooks. Accelerate your analytics with Apache Spark.' A link 'Learn more about IBM Analytics for Apache Spark' is present. Under the 'Exchange' section, there's a sub-section titled 'Combine your data with public data to get fresh insights.' In the main content area, there's a list titled 'IBM Graph (1)' containing one item: 'IBM Graph-jt'. Below the item, it says 'IBM Graph DEV Plan: Standard'. There are 'Actions' and a gear icon for managing the service. The URL in the browser address bar is <https://console.ng.bluemix.net/services/1fe5e8d5-701a-438f-83a7-54e959192ae5/?panelId=manage>.

- IBM graph will be shown. Click on 'Open'

The screenshot shows the 'Service Details - IBM Bluemix Data & Analytics' page for the 'IBM Graph' service. The top navigation bar includes 'Docs', 'IBM Bluemix Data & Analytics', 'Catalog', 'Support', and 'Account'. Below the navigation bar, there's a breadcrumb trail 'Data & Analytics' and a back arrow. The main content area is titled 'IBM Graph-jt'. It features three tabs: 'Manage' (selected), 'Service Credentials', and 'Connections'. On the right side, there's a large green 'Open' button with a document icon. Below the button, the text 'An enterprise-grade property graph' is displayed. At the bottom of the page, there's a Gremlin query editor with the placeholder text '1 // Enter your Gremlin query here. (Shift + Enter) to execute'. The URL in the browser address bar is <https://console.ng.bluemix.net/services/1fe5e8d5-701a-438f-83a7-54e959192ae5/?panelId=manage>.

- Open the service instance of graph db.

The screenshot shows the IBM Graph interface within a browser window. The URL is <https://console.ng.bluemix.net/data/graphdb/1fe5e8d5-701a-438f-83a7-54e959192ae5/query>. The main content area displays a 'Welcome to IBM Graph!' card with the text: 'If it's your first time here, read our walkthrough of the interface to learn the basics of the IBM Graph query builder and response cards.' Below this card is a section titled 'Other things to do...' with three links: 'Read Graph Tutorials' (lightbulb icon), 'Load Sample Data' (scissors icon), and 'Read API Documentation' (book icon).

- Load the Sample db, and click on 'See sample queries card'

The screenshot shows the IBM Graph interface with the 'Samples' card expanded. The URL is <https://console.ng.bluemix.net/data/graphdb/1fe5e8d5-701a-438f-83a7-54e959192ae5/query>. The left sidebar shows the 'Samples' tab selected. The main content area displays a 'Samples' card with the text: 'All samples automatically load a schema alongside the dataset. Loading time can take up to a minute.' Below this is a 'Music Festival' card with the text: 'Data for ticket purchases, advertisements, and performance details at a music festival. See Sample Queries Card'. The right side of the interface shows the same 'Welcome to IBM Graph!' card and 'Interface Walkthrough' link as in the previous screenshot.

- Schema is displayed along with a query console and a set of sample queries.

The screenshot shows the IBM Graph interface. On the left, there are navigation links for 'Upload', 'Samples', and 'Resources'. The main area has tabs for 'Query' and 'Tutorials'. The 'Query' tab is active, displaying a Gremlin query input field containing the placeholder text: '1 // Enter your Gremlin query here. (Shift + Enter) to execute.' Below this is a section titled 'Sample Queries: Music Festival' which includes a 'Dataset Model' diagram and a list of sample queries.

Dataset Model:

```

graph TD
    attendee((attendee)) -- "name, gender, age" --> band((band))
    band -- "name, genre, monthly_listeners" --> venue((venue))
    venue -- "name, address, average_rating" --> attendee
    attendee -- "time, date" --> bought_ticket[ ]
    bought_ticket -- "bought_ticket" --> band
    band -- "time, date" --> performing_at[ ]
    performing_at -- "performing_at" --> venue
    venue -- "time, date" --> advertised_to[ ]
    attendee -- "advertised_to" --> venue
  
```

Sample Queries:

- Create an attendee who bought a ticket to a band performance.
- Visualize all venues that are hosting folk performances.
- List all performances that an attendee has bought tickets for.
- Visualize tickets bought by a specific attendee.
- List attendees who have bought indie tickets at a specific venue.
- Visualize all hip hop ticket purchases at the festival.
- Recommend folk venues for male attendees to visit.
- Visualize all female purchases for any performance at the festival.

- Query:** def gt = graph.traversal();gt.V().hasLabel("attendee").has("name", "Chris Tipps").outE("bought_ticket").inV().hasLabel("band").has("genre","Hip Hop").path();
The above query, lists all tickets of genre: 'Hip Hop' bought by attendee - Chris Tipps

The screenshot shows the IBM Graph interface with the 'Query' tab active. The query input field contains the Gremlin code provided in the previous list item. To the right, the results are visualized as a graph where vertices represent attendees and bands, and edges represent 'bought_ticket' relationships. A tooltip for one vertex indicates it is a 'band' vertex with ID 28768. The bottom status bar shows 'Vertices: 12'.

- Query: def v1 = graph.addVertex("name", "Sowmya V", label, "attendee", "age", 28, "gender", "female"); def v2 = graph.addVertex("name", "Declan McKenna", label, "band", "genre", "Folk", "monthly_listeners", "192302"); v1.addEdge("bought_ticket", v2);
This query creates an attendee who bought a ticket to the band performance.

The screenshot shows the IBM Graph interface with the following components:

- Left Sidebar:** Includes buttons for Upload, Samples, and Resources.
- Top Bar:** Shows "Service Details - IBM Blu" and "IBM Graph".
- Central Area:**
 - Dataset Model:** A diagram showing entities: attendee (green), band (blue), and venue (orange). Edges include "bought_ticket" from attendee to band, "performing_at" from band to venue, and "advertised_to" from attendee to venue.
 - Sample Queries:** A list of 8 queries with icons:
 - Create an attendee who bought a ticket to a band performance.
 - Visualize all venues that are hosting folk performances.
 - List all performances that an attendee has bought tickets for.
 - Visualize tickets bought by a specific attendee.
 - List attendees who have bought indie tickets at a specific venue.
 - Visualize all hip hop ticket purchases at the festival.
 - Recommend folk venues for male attendees to visit.
 - Visualize all female purchases for any performance at the festival.
- Bottom Bar:** Buttons for "Learn the Interface", a help icon, and close/collapse buttons.

- Query: def gt = graph.traversal(); gt.V().hasLabel("attendee").has("name", "Sowmya V").outE("bought_ticket").inV().hasLabel("band").path();
This query is to retrieve tickets bought by attendee 'Sowmya V'

The screenshot shows the IBM Graph interface with the following components:

- Left Sidebar:** Includes buttons for Upload, Samples, and Resources.
- Top Bar:** Shows "Service Details - IBM Blu" and "IBM Graph".
- Central Area:**
 - Graph:** A visualization showing a green circle labeled "atten." connected to a blue circle labeled "band" by a single edge.
 - Code Editor:** Displays the traversal query:


```
def gt = graph.traversal(); gt.V().hasLabel("attendee").has("name", "Sowmya V").outE("bought_ticket")
```
 - Results:** A detailed JSON object representing the traversed path:


```
1: [2: {"label": "attendee", "id": 40964184, "type": "vertex", "properties": {"age": 28, "gender": "female", "name": "Sowmya V"}}, 3: {"label": "band", "id": 40964185, "type": "vertex", "properties": {"genre": "Folk", "monthly_listeners": "192302", "name": "Declan McKenna"}}, 4: {"label": "bought_ticket", "id": 40964186, "type": "edge", "properties": {"label": "bought_ticket", "source": 2, "target": 3}}]
```
- Bottom Bar:** Buttons for Filter (Label, Type, Properties) and Vertices: 2.

- Query: def gt =
`graph.traversal();gt.V().hasLabel("attendee").has("gender","female").outE("bought_ticket").inV().hasLabel("band").has("genre", "Hip Hop").path();`

This Query lists all the females who have purchased tickets in the festival for genre 'Hip Hop'

The screenshot shows the IBM Graph interface with a Gremlin query entered in the query editor:

```
def gt = graph.traversal();gt.V().hasLabel("attendee").has("gender", "female").outE("bought_ticket")
```

The results pane displays a graph visualization of the query results. Vertices are colored by type: blue for bands and orange for attendees. Edges represent the 'bought_ticket' relationship. A summary at the bottom right indicates there are 56 vertices.

Deleting a graph

The screenshot shows the IBM Graph interface with a Gremlin query entered in the query editor:

```
g
test_graph
delete
```

The results pane displays a graph visualization of the query results. Vertices are colored by type: blue for bands and orange for attendees. Edges represent the 'bought_ticket' relationship. A summary at the bottom right indicates there are 56 vertices.

Using APIs to perform operations on database

1. Authentication – Session API

From the Security tab in the IBM Graph service, you can find the API_URL, username and password. We need these credentials to access graph DB. If you get response code 200, the connection has succeeded

```
Falcon — bash — 80x24
Adithyas-MacBook-Pro:~ Falcon$ quit()
[>
[Adithyas-MacBook-Pro:~ Falcon$ curl -u c2bbe748-38f5-4d09-9b52-cb45ab3eaeee4 http://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g
[Enter host password for user 'c2bbe748-38f5-4d09-9b52-cb45ab3eaeee4':
{"requestId":"96b9073a-82bd-49ec-8586-40a64b75e1ef","status":{"message":"","code":200,"attributes":{}}, "result":{"data":["StandardTitanGraph"],"meta":{}}}
Adithyas-MacBook-Pro:~ Falcon$
```

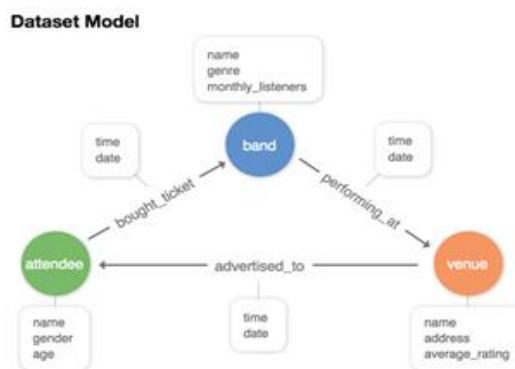
Now generate token to establish connection. We fire a GET request with the same credentials to the API_URL/_session to get the session token

The screenshot shows the Postman interface with a successful API call. The URL is set to `https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/_session`. In the Authorization tab, Basic Auth is selected with Username `c2bbe748-38f5-4d09-9b52-cb45ab3eaeee4` and Password masked. The response body is a JSON object:

```
{"gds-token": "YzJiYmU3NDgtMzhmNS00ZDA5LTlINTItY2I0NWFiM2VhZWU0OjE0ODc2NTU4MDU4MTY6VzNpeEJhdynVXhCMXZjOF1qbGhJQ8lUmF1R18RbFh1Y0hsTnZwQzBvVT0="}
```

This session will be used to query schema at API_URL/g/

A graphical equivalent and the json schema received is as below:



```

"result": {
  "data": [
    {
      "propertyKeys": [
        {
          "name": "address",
          "dataType": "String",
          "cardinality": "SINGLE"
        },
        {
          "name": "average_rating",
          "dataType": "Float",
          "cardinality": "SINGLE"
        },
        {
          "name": "name",
          "dataType": "String",
          "cardinality": "SINGLE"
        },
        {
          "name": "gender",
          "dataType": "String",
          "cardinality": "SINGLE"
        },
        {
          "name": "age",
          "dataType": "Integer",
          "cardinality": "SINGLE"
        },
        {
          "name": "genre",
          "dataType": "String",
          "cardinality": "SINGLE"
        },
        {
          "name": "monthly_listeners",
          "dataType": "String",
          "cardinality": "SINGLE"
        }
      ],
      "vertexIndexes": [
        {
          "name": "date",
          "dataType": "String",
          "cardinality": "SINGLE"
        },
        {
          "name": "time",
          "dataType": "String",
          "cardinality": "SINGLE"
        }
      ],
      "vertexLabels": [
        {
          "name": "person"
        },
        {
          "name": "Attendee"
        },
        {
          "name": "Band"
        },
        {
          "name": "Venue"
        }
      ],
      "edgeLabels": [
        {
          "name": "bought_ticket",
          "directed": true,
          "multiplicity": "MULTI"
        },
        {
          "name": "advertised_to",
          "directed": true,
          "multiplicity": "MULTI"
        },
        {
          "name": "performing_at",
          "directed": true,
          "multiplicity": "MULTI"
        }
      ],
      "edgeIndexes": [
        {
          "name": "eByBoughtTicket",
          "composite": true,
          "unique": false,
          "propertyKeys": [
            "time"
          ],
          "requiresReindex": false,
          "type": "edge"
        } ] } ]
    }
  ]
}

```

2. Index APIs

a. Get existing indices:

The screenshot shows the IBM Graph Alpha API Explorer interface. The URL is https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/index/. The response status is 200 OK, time 3600 ms, and size 1.02 KB. The response body is a JSON object:

```

1 {
2   "requestId": "01a93228-1a91-4450-a9ff-730fc10019ed",
3   "status": {
4     "message": "",
5     "code": 200,
6     "attributes": {}
7   },
8   "result": {
9     "data": [
10       {
11         "name": "eByBoughtTicket",
12         "composite": true,
13         "unique": false,
14         "propertyKeys": [
15           "time"
16         ],
17         "requiresReindex": false,
18         "type": "edge"
19       },
20       {
21         "name": "vByName",
22         "composite": true,
23         "unique": false,
24         "propertyKeys": [
25           "name"
26         ]
27     }
28   }
29 }

```

response:

```
{
  "requestId": "01a93228-1a91-4450-a9ff-730fc10019ed",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "name": "eByBoughtTicket",
        "composite": true,
        "unique": false,
        "propertyKeys": [
          "time"
        ],
        "requiresReindex": false,
        "type": "edge"
      },
      {
        "name": "vByName",
        "composite": true,
        "unique": false,
        "propertyKeys": [
          "name"
        ]
      }
    ]
  }
}
```

```

"unique": false,
"propertyKeys": [
  "time"
],
"requiresReindex": false,
"type": "edge"
},
{
  "name": "vByName",
  "composite": true,
  "unique": false,
  "propertyKeys": [
    "name"
  ],
  "requiresReindex": false,
  "type": "vertex"
},

```

b. GET a specific index by value

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

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/index/vByGender/
- Method:** GET
- Headers:** No Environment
- Auth:** Username: c2bbe748-38f5-4d09-9b52-cb45ab3eaee- and Password: c2bd817d-067c-477e-b77c-cb9b64d7438
- Body:**

```

1  {
2    "requestId": "1c2f68e7-9dab-4d1d-847f-5b83a2e5e228",
3    "status": {
4      "message": "",
5      "code": 200,
6      "attributes": {}
7    },
8    "result": {
9      "data": [
10        {
11          "name": "vByGender",
12          "composite": true,
13          "unique": false,
14          "propertyKeys": [
15            "gender"
16          ],
17          "requiresReindex": false,
18          "type": "vertex"
19        }
20      ],
21      "meta": {}
22    }
23  }

```
- Status:** 200 OK
- Time:** 2960 ms

response:

```
{
  "requestId": "1c2f68e7-9dab-4d1d-847f-5b83a2e5e228",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "name": "vByGender",
        "composite": true,
        "unique": false,
        "propertyKeys": [
          "gender"
        ],
        "requiresReindex": false,
        "type": "vertex"
      }
    ],
    "meta": {}
  }
}
```

c. GET status of a particular index by name (vByName) to see if it's enabled or disabled:

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

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/index/eByBoughtTicket/status
- Method:** GET
- Headers:** No Environment
- Body:**
 - Pretty: {
 - 1: "requestId": "d30a57f1-f16f-43f7-bc75-543f66991f4e",
 - 2: "status": {
 - 3: "message": "",
 - 4: "code": 200,
 - 5: "attributes": {}
 - 6: },
 - 7: "result": {
 - 8: "data": [
 - 9: {
 - 10: "eByBoughtTicket": "ENABLED"
 - 11: }
 - 12:],
 - 13: "meta": {}
 - 14: }
 - 15: }
 - 16: }
- Responses:**
 - Status: 200 OK
 - Time: 3023 ms
 - Size: 604

response:

```
{
  "requestId": "d30a57f1-f16f-43f7-bc75-
543f66991f4e",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "eByBoughtTicket": "ENABLED"
      }
    ],
    "meta": {}
  }
}
```

d. DELETE an existing index

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

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/index/vByName
- Method:** DELETE
- Status:** 202 Accepted
- Time:** 4266 ms
- Size:** 559 B
- Body:** (1 item) [{"requestId": "efe3557f-b945-4f13-89fc-084f4abc84ba", "status": {"message": "", "code": 200, "attributes": {}}, "result": {"data": [], "meta": {}}}]

Check with GET if this index exists

3. Vertex APIs

- POST method to create a new vertex. In below example, a vertex is created with id 40964168 but with no properties as we need to specify a property in payload to assign one.

The screenshot shows a Postman interface with the following details:

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/
- Method:** POST
- Authorization:** (selected)
- Headers:** (3)
- Body:** (selected)
- Form Data:** (selected)
- Body Content:**

```

1  {
2    "requestId": "99c13137-4730-466d-98b7-df8008789928",
3    "status": {
4      "message": "",
5      "code": 200,
6      "attributes": {}
7    },
8    "result": {
9      "data": [
10        {
11          "id": 40964168,
12          "label": "vertex",
13          "type": "vertex",
14          "properties": {}
15        }
16      ],
17      "meta": {}
18    }
19  }

```

b. POST method with payload containing properties:

The screenshot shows a Postman interface with the following details:

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096146
- Method:** POST
- Authorization:** (selected)
- Headers:** (3)
- Body:** (selected)
- Form Data:** (selected)
- JSON (application/json) Content:**

```

1  {
2    "label": "person",
3    "properties": {
4      "name": "Sowmya V",
5      "verified": false
6    }
7  }

```

Request:

{

```

"label": "person",
"properties":
{
    "name": "Sowmya V",
    "verified": false
}
}

```

The screenshot shows a browser-based REST client interface. The URL in the address bar is `https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416`. The request method is POST. The request body is a JSON object:

```

{
  "requestId": "b7329e62-8dbe-48cb-b15c-fc49dc753691",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": 40964168,
        "label": "vertex",
        "type": "vertex",
        "properties": {
          "name": [
            {
              "id": "odxqh-oe05k-sl",
              "value": "Sowmya V"
            }
          ],
          "verified": [
            {
              "id": "ody4p-oe05k-4eth",
              "value": false
            }
          ]
        }
      ]
    ],
    "meta": {}
  }
}

```

Response:

```

{
  "requestId": "b7329e62-8dbe-48cb-b15c-fc49dc753691",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": 40964168,
        "label": "vertex",
        "type": "vertex",
        "properties": {
          "name": [
            {
              "id": "odxqh-oe05k-sl",
              "value": "Sowmya V"
            }
          ],
          "verified": [
            {
              "id": "ody4p-oe05k-4eth",
              "value": false
            }
          ]
        }
      }
    ]
  }
}

```

```

    "type": "vertex",
    "properties": {
      "name": [
        ],
        "meta": {}
      }
    }
  
```

c. Update the properties using POST

The screenshot shows a Postman interface with the following details:

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416
- Method:** POST
- Body:** JSON (application/json)
- Body Content:**

```

1  {
2    "label": "person",
3    "properties": {
4      "name": "Sowmya Viswanathan",
5      "verified": false
6    }
7  }
8
9
  
```

The screenshot shows a Postman interface with the following details:

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416
- Method:** POST
- Status:** 200 OK
- Body:** JSON
- Body Content:**

```

1  {
2    "requestId": "1a94d02a-7966-46ba-a708-8f48acab364c",
3    "status": {
4      "message": "",
5      "code": 200,
6      "attributes": {}
7    },
8    "result": {
9      "data": [
10        {
11          "id": 40964168,
12          "label": "vertex",
13          "type": "vertex",
14          "properties": {
15            "name": [
16              {
17                "id": "odzbd-oe05k-sl",
18                "value": "Sowmya Viswanathan"
19              }
20            ],
21            "verified": [
22              {
23                "id": "odzpl-oe05k-4eth",
24                "value": false
25              }
26            ]
27          }
28        },
29        "meta": {}
30      ]
31    }
  
```

Response:

```
{
  "requestId": "1a94d02a-7966-46ba-a708-8f48acab364c",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": 40964168,
        "label": "vertex",
        "type": "vertex",
        "properties": {
          "name": [
            {
              "id": "odzbd-oe05k-sl",
              "value": "Sowmya Viswanathan"
            }
          ],
          "verified": [
            {
              "id": "odzpl-oe05k-4eth",
              "value": false
            }
          ]
        },
        "meta": {}
      }
    ]
  }
}
```

d. PUT method to delete previous updates and update it with the new value

The screenshot shows a Postman request configuration. The URL is `https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416`. The method is set to `PUT`. The `Body` tab is selected, showing the following JSON payload:

```

1 { "properties": 2   { 3     "name": "Sowmya V", 4     "verified": false 5   } 6 } 7 } 8

```

The screenshot shows a Postman interface with the following details:

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416
- Status:** 200 OK
- Body (Pretty):**

```

1 {  
2   "requestId": "7619c180-4a78-499c-b92c-ef2342660969",  
3   "status": {  
4     "message": "",  
5     "code": 200,  
6     "attributes": {}  
7   },  
8   "result": {  
9     "data": [  
10       {  
11         "id": 40964168,  
12         "label": "vertex",  
13         "type": "vertex",  
14         "properties": {  
15           "name": [  
16             {  
17               "id": "1crv2h-oe05k-sl",  
18               "value": "Sowmya V"  
19             }  
20           ],  
21           "verified": [  
22             {  
23               "id": "1crvgp-oe05k-4eth",  
24               "value": false  
25             }  
26           ]  
27         }  
28       },  
29     ],  
30     "meta": {}  
31   }  
32 }
```

Response:

```
{
  "requestId": "7619c180-4a78-499c-b92c-ef2342660969",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": 40964168,
        "label": "vertex",
        "type": "vertex",
        "properties": {
          "name": [
            {
              "id": "1crv2h-oe05k-sl",
              "value": "Sowmya V"
            }
          ],
          "verified": [
            {
              "id": "1crvgp-oe05k-4eth",
              "value": false
            }
          ]
        }
      }
    ],
    "meta": {}
  }
}
```

GET call for the same id. The last made changes are available

```
1 {  
2   "requestId": "c2c60473-f600-46a3-9396-9088a534b495",  
3   "status": {  
4     "message": "",  
5     "code": 200,  
6     "attributes": {}  
7   },  
8   "result": {  
9     "data": [  
10       {  
11         "id": 40964168,  
12         "label": "vertex",  
13         "type": "vertex",  
14         "properties": {  
15           "verified": [  
16             {  
17               "id": "1crvgp-oe05k-4eth",  
18               "value": false  
19             }  
20           ],  
21           "name": [  
22             {  
23               "id": "1crv2h-oe05k-s1",  
24               "value": "Sowmya V"  
25             }  
26           ]  
27         }  
28       },  
29     ],  
30     "meta": {}  
31   }  
32 }
```

e. Check the out and in vertices to a particular vertex. Below id mentioned is that of a 'venue' in db.

```
https://ibmgraph-alpha.x  + No Environment

GET https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/32928/b Params Send

Body Cookies (1) Headers (12) Tests Status: 200 OK Time: 2966 ms

Pretty Raw Preview JSON ▾

1 ↴ { "requestId": "38b5c9d0-973b-4899-9b32-544076c17595",
2   "status": {
3     "message": "",
4     "code": 200,
5     "attributes": {}
6   },
7   "result": {
8     "data": [
9       {
10         "id": 8216,
11         "label": "attendee",
12         "type": "vertex",
13         "properties": {
14           "gender": [
15             {
16               "id": "35v-6c8-111",
17               "value": "female"
18             }
19           ],
20           "name": [
21             {
22               "id": "3k3-6c8-s1",
23               "value": "Kelsey Adams"
24             }
25           ],
26           "age": [
27             {
28               "id": "3yb-6c8-2dh",
29               "value": 30
30             }
31           ]
32         }
33       }
34     ]
35   }
36 }
```

Response:

Contains both bands(2) and attendees(5)

],

- f. GET command to retrieve IDs of both the in and out connections at a particular vertex.
Use 'bothIds'

```

1  {
2   "requestId": "1e5eda66-31bb-43b4-bd6d-a9a14bb98edf",
3   "status": {
4     "message": "",
5     "code": 200,
6     "attributes": {}
7   },
8   "result": {
9     "data": [
10    8216,
11    16552,
12    24600,
13    40964192,
14    40992768,
15    8352,
16    16544
17  ],
18  "meta": {}
19 }
20

```

Response:

```
{
  "requestId": "1e5eda66-31bb-43b4-bd6d-a9a14bb98edf",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      8216,
      16552,
      24600,
      40964192,
      40992768,
      8352,
      16544
    ],
    "meta": {}
  }
}
```

}

g. DELETE properties by a key for a vertex. Using id: 40964168 created earlier.

The screenshot shows a REST API client interface. The URL is https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/40964168. The method is set to DELETE. The response status is 200 OK. The response body is a JSON object:

```

1 [ {
2   "requestId": "cdfa930a-fd83-4d44-90c5-cf7078462ccb",
3   "status": {
4     "message": "",
5     "code": 200,
6     "attributes": {}
7   },
8   "result": {
9     "data": [
10       {
11         "id": 40964168,
12         "label": "vertex",
13         "type": "vertex",
14         "properties": {
15           "name": [
16             {
17               "id": "1crv2h-oe05k-s1",
18               "value": "Sowmya V"
19             }
20           ]
21         }
22       }
23     ],
24     "meta": {}
25   }
26 }
]

```

Request: https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/40964168?verified

Response:

```
{
  "requestId": "cdfa930a-fd83-4d44-90c5-cf7078462ccb",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {} },
  "result": {
    "data": [
      {
        "id": 40964168,
        "label": "vertex",
        "type": "vertex",
        "properties": {
          "name": [
            {
              "id": "1crv2h-oe05k-s1",
              "value": "Sowmya V"
            }
          ]
        }
      }
    ],
    "meta": {}
  }
}
```

```

    "id": "1crv2h-oe05k-sl",
    "value": "Sowmya V"
  } ] } ], "meta": {} }
}

```

To DELETE the vertex, send the vertex ID without any property.

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

- Method:** DELETE
- URL:** <https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416>
- Status:** 200 OK
- Body:** (The body is empty, indicating the vertex was successfully deleted.)

To verify if deleted, use the GET command

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

- Method:** GET
- URL:** <https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/vertices/4096416>
- Status:** 404 Not Found
- Body:** [{"code": "NotFoundError", "message": ""}]

4. Edge APIs

- a. POST method to create an edge. iD: 81924168 and 81924136

The screenshot shows the Postman application interface. A header bar at the top has a search field containing "https://ibmgraph-alpha" and a "+" button. Below this is a card with "POST" and the URL "https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/edges". A navigation bar below the card includes tabs for "Authorization", "Headers (3)", "Body", "Pre-request Script", and "Tests". The "Body" tab is active, indicated by a blue dot. Under "Body", there are four options: "form-data", "x-www-form-urlencoded", "raw", and "binary". "raw" is selected and highlighted in orange. A dropdown menu next to "raw" shows "JSON (application/json)". The main area contains the raw JSON payload:

```

1 ▾ {
2   "outV": 81924168 ,
3   "label": "is_a",
4   "inV": 81924136 |
5
6 }
7

```

Response:

```
{
  "requestId": "0eb34e32-2b12-4ec4-b417-afd4565f2cd6",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": "2pjo5l-1crx3c-8sut-1crx2g",
        "label": "is_a",
        "type": "edge",
        "inVLabel": "person",
        "outVLabel": "person",
        "inV": 81924136,
        "outV": 81924168
      }
    ],
    "meta": {}
  }
}
```

b. Adding a property to the above edge using 'was_' label and using POST

The screenshot shows the Postman application interface. The URL in the header is `https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/edges`. The 'Body' tab is selected, and the 'raw' option is chosen under the dropdown. The JSON payload is:

```

1 {
2   "outV": 81924168 ,
3   "label": "was_a",
4   "inV": 81924136,
5   "properties":
6   {
7     "year":2017
8   }
9
10 }
11

```

Response:

```
{
  "requestId": "f206a92f-ce2d-42db-aa42-ace834647786",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": "3dy4uh-1crx3c-azv9-1crx2g",
        "label": "was_a",
        "type": "edge",
        "inVLabel": "person",
        "outVLabel": "person",
        "inV": 81924136,
        "outV": 81924168,
        "properties": {
          "year": 2017
        }
      }
    ],
    "meta": {}
  }
}
```

c. Updating the property value to a new value

The screenshot shows the Postman application interface. The URL in the header is `https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/edges`. The method is set to `POST`. The `Body` tab is selected, and the content type is set to `JSON (application/json)`. The raw JSON payload is:

```

1  {
2   "outV": 81924168 ,
3   "label": "was_a",
4   "inV": 81924136,
5   "properties":
6   {
7     "year":2016|
8   }
9
10 }
11

```

Response:

```
{
  "requestId": "e44d4fa6-7883-4f90-b18b-f6dd7de64bc7",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": "2pjojt-1crx3c-azv9-1crx2g",
        "label": "was_a",
        "type": "edge",
        "inVLabel": "person",
        "outVLabel": "person",
        "inV": 81924136,
        "outV": 81924168,
        "properties": {
          "year": 2016
        }
      }
    ],
    "meta": {}
  }
}
```

d. Using edge id to update the property using PUT

The screenshot shows the Postman interface with the following details:

- Request URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/edges/2pjojt-1crx3c-azv9-1crx2g
- Method:** PUT
- Body (raw JSON):**

```

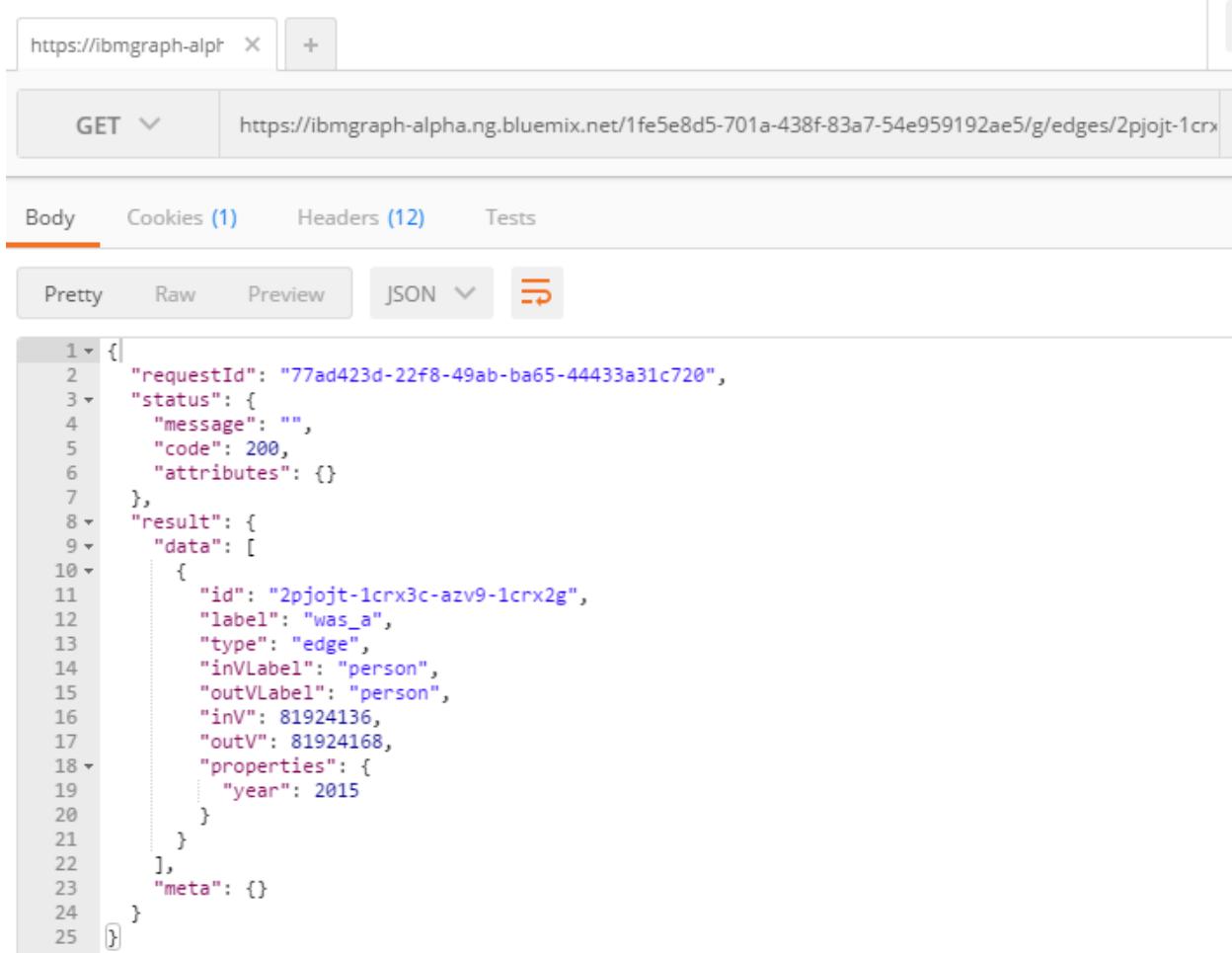
1 {
2   "properties": {
3     "year": 2015
4   }
5 }
6
7
8
  
```
- Content Type:** application/json

Response:

```

{
  "requestId": "6717d1c6-dab1-4ea4-82d2-5c412ae802d1",
  "status": {
    "message": "",
    "code": 200,
    "attributes": {}
  },
  "result": {
    "data": [
      {
        "id": "2pjojt-1crx3c-azv9-1crx2g",
        "label": "was_a",
        "type": "edge",
        "inVLabel": "person",
        "outVLabel": "person",
        "inV": 81924136,
        "outV": 81924168,
        "properties": {
          "year": 2015
        }
      }
    ],
    "meta": {}
  }
}
  
```

e. GET the updated properties of the edge using its id



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

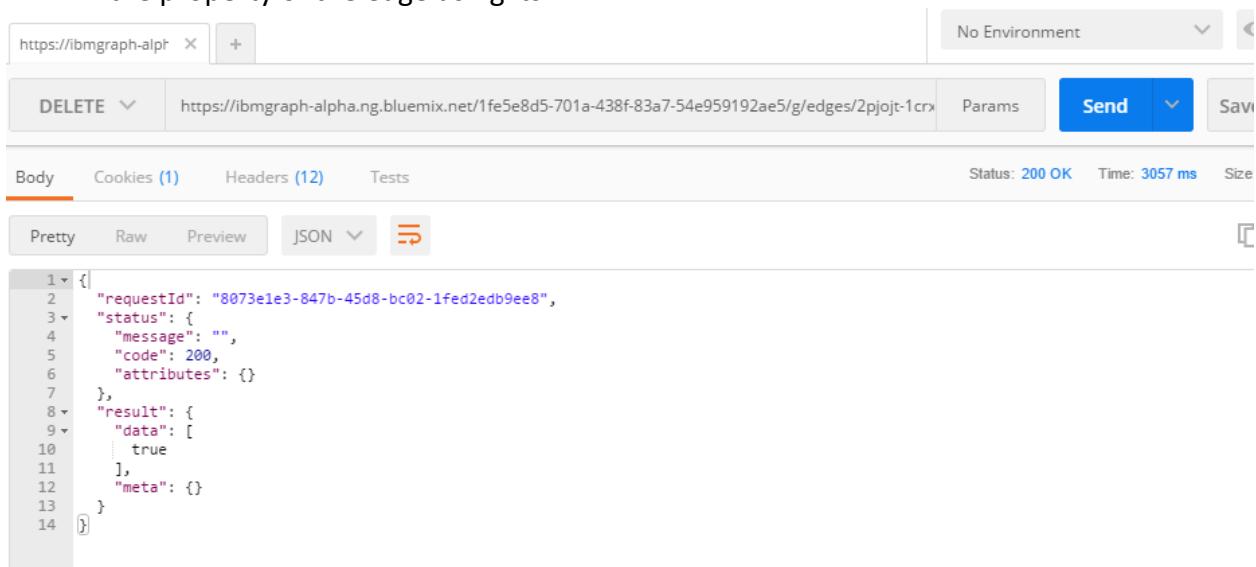
- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/edges/2pjojt-1crx3c-azv9-1crx2g
- Method:** GET
- Body:** JSON (Pretty)
- Response:**

```

1 {{"requestId": "77ad423d-22f8-49ab-ba65-44433a31c720",
2   "status": {
3     "message": "",
4     "code": 200,
5     "attributes": {}
6   },
7   "result": {
8     "data": [
9       {
10        "id": "2pjojt-1crx3c-azv9-1crx2g",
11        "label": "was_a",
12        "type": "edge",
13        "inVLabel": "person",
14        "outVLabel": "person",
15        "inV": 81924136,
16        "outV": 81924168,
17        "properties": {
18          "year": 2015
19        }
20      }
21    ],
22    "meta": {}
23  }
24}
25

```

DELETE the property of the edge using its ID



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

- URL:** https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/g/edges/2pjojt-1crx3c-azv9-1crx2g
- Method:** DELETE
- Body:** JSON (Pretty)
- Response:**

```

1 {{"requestId": "8073e1e3-847b-45d8-bc02-1fed2edb9ee8",
2   "status": {
3     "message": "",
4     "code": 200,
5     "attributes": {}
6   },
7   "result": {
8     "data": [
9       true
10     ],
11     "meta": {}
12   }
13}
14

```

Verify if its deleted

The screenshot shows a browser or API tool interface. At the top, there is a header bar with a URL field containing "https://ibmgraph-alpha.ng.bluemix.net/edges/2pjot-1crx" and a status indicator "No Envir". Below this is a navigation bar with "GET" selected, a URL field with "https://ibmgraph-alpha.ng.bluemix.net/edges/2pjot-1crx", and a "Params" button. The main area has tabs for "Body", "Cookies (1)", "Headers (11)", and "Tests", with "Body" being the active tab. It displays a status message "Status: 404 Not Found". Below the tabs are buttons for "Pretty", "Raw", "Preview", "Text", and "JSON". The "Text" tab is active, showing the response body: "1 {"code": "NotFoundError", "message": ""}".

5. Graph APIs

a. Fetch the existing graphs using GET command

The screenshot shows the IBM Graph interface. At the top, there is a header bar with a URL field containing "https://ibmgraph-alpha.ng.bluemix.net/_graphs" and a status indicator "No Envir". Below this is a navigation bar with "GET" selected, a URL field with "https://ibmgraph-alpha.ng.bluemix.net/_graphs", and a "Params" button. The main area has tabs for "Body", "Cookies (1)", "Headers (11)", and "Tests", with "Body" being the active tab. It displays a status message "Status: 200 OK". Below the tabs are buttons for "Pretty", "Raw", "Preview", "JSON", and "Text". The "Text" tab is active, showing the response body: "1 { 2 \"graphs\": [3 \"g\", 4 \"test_graph\" 5] 6 }

Below the interface, a terminal window titled "IBM Graph" shows a Gremlin session. The session starts with "IBM Graph-jt > g <". The user types "def gt = graph.traversal();gt.V().hasLabel("band").has("genre", "Hip Hop").inE()", followed by "Shift + Enter" to execute. The terminal then shows the results of the traversal, including vertex 16400 with label "band" and type "vertex".

b. Create a new graph using POST method

The screenshot shows the Postman application interface. The URL in the header is `https://ibmgraph-alpha.ng.bluemix.net/_graphs`. The request method is set to `POST`. The `Body` tab is selected, showing a raw JSON payload:

```

1 {
2   "graphID": "a1b2c3",
3   "dbUrl": "https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/a1b2c3"
4 }
5

```

Below the body, the response status is `201 Created` and the time taken is `2852 ms`. The response body is also shown in JSON format:

```

1 []
2   "graphId": "3b0fc10-aa65-4ee4-a8b8-451a4299d617",
3   "dbUrl": "https://ibmgraph-alpha.ng.bluemix.net/1fe5e8d5-701a-438f-83a7-54e959192ae5/3b0fc10-aa65-4ee4-a8b8-451a4299d617"
4 []

```

At the bottom, there is a screenshot of the IBM Graph interface showing the newly created graph `3b0fc10-aa65-4ee4-a8b8-451a4299d617`.

c. DELETE a graph

The screenshot shows the Postman application interface. The URL in the header is `https://ibmgraph-alpha.ng.bluemix.net/_graphs/test_graph`. The request method is set to `DELETE`. The `Body` tab is selected, showing a raw text payload:

```

1 {"data":{}}

```

Graph is deleted

