

SQLite

Purchase Order Management

Schema Diagram:

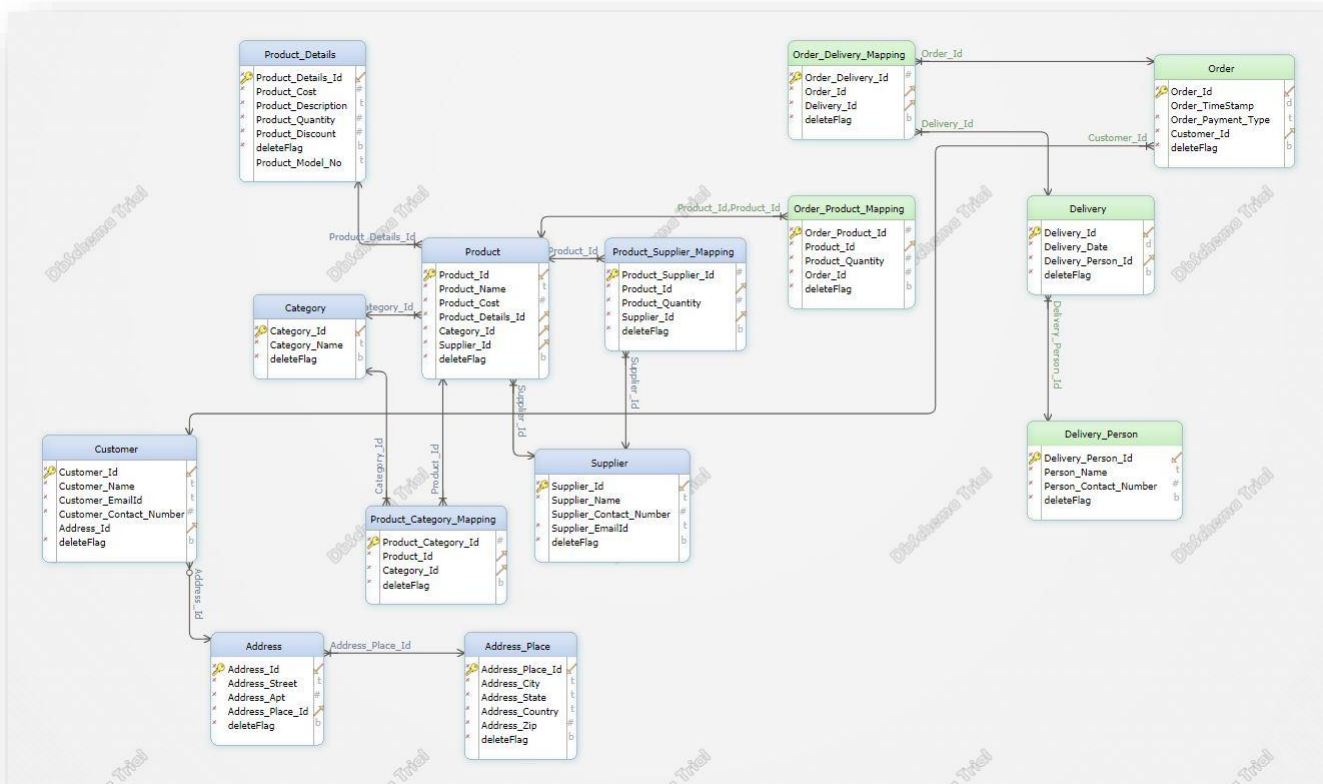


Table Specification:

Table: Address

| Indexes | Field Name | Data Type | Description |
|---------------------|------------------|--|-------------|
| * | Address_Id | integer | |
| * | Address_Street | varchar(20000000000) | |
| * | Address_Apt | integer | |
| * | Address_Place_Id | varchar(20000000000) | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Address | ON Address_Id | |
| Foreign Keys | | | |
| | Fk_Address | (Address_Place_Id) ref Address_Place (Address_Place_Id) | |

Table: Address_Place

| Indexes | Field Name | Data Type | Description |
|----------------|------------------|------------------------|-------------|
| * | Address_Place_Id | integer | |
| * | Address_City | varchar(20000000000) | |
| * | Address_State | varchar(20000000000) | |
| * | Address_Country | varchar(20000000000) | |
| * | Address_Zip | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Address_Place | ON Address_Place_Id | |

Table: Category

| Indexes | Field Name | Data Type | Description |
|----------------|---------------|-----------------------|-------------|
| * | Category_Id | integer | |
| * | Category_Name | varchar(2000000000) | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Category | ON Category_Id | |

Table: Customer

| Indexes | Field Name | Data Type | Description |
|---------------------|-------------------------|---|-------------|
| * | Customer_Id | integer | |
| * | Customer_Name | varchar(2000000000) | |
| * | Customer_EmailId | varchar(2000000000) | |
| * | Customer_Contact_Number | integer | |
| | Address_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Customer | ON Customer_Id | |
| Foreign Keys | | | |
| | Fk_Customer | (Address_Id) ref Address (Address_Id) | |

Table: Delivery

| Indexes | Field Name | Data Type | Description |
|---------------------|--------------------|--|-------------|
| * | Delivery_Id | integer | |
| * | Delivery_Date | date | |
| * | Delivery_Person_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Delivery | ON Delivery_Id | |
| Foreign Keys | | | |
| | Fk_Delivery | (Delivery_Person_Id) ref Delivery Person (Delivery_Person_Id) | |

Table: Delivery_Person

| Indexes | Field Name | Data Type | Description |
|----------------|-----------------------|------------------------|-------------|
| * | Delivery_Person_Id | integer | |
| * | Person_Name | varchar(20000000000) | |
| * | Person_Contact_Number | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Delivery_Person | ON Delivery_Person_Id | |

Table: Order

| Indexes | Field Name | Data Type | Description |
|---------------------|--------------------|--|-------------|
| * | Order_Id | integer | |
| | Order_TimeStamp | datetime DEFAULT CURRENT_TIMESTAMP | |
| * | Order_Payment_Type | varchar(20000000000) | |
| * | Customer_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Order | ON Order_Id | |
| Foreign Keys | | | |
| | Fk_Order | (Customer_Id) ref Customer (Customer_Id) | |

Table: Order_Delivery_Mapping

| Indexes | Field Name | Data Type | Description |
|---------------------|---------------------------|--|-------------|
| * | Order_Delivery_Id | integer | |
| * | Order_Id | integer | |
| * | Delivery_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Order_Delivery_Mapping | ON Order_Delivery_Id | |
| Foreign Keys | | | |
| | Fk_Order_Delivery_Mapping | (Delivery_Id) ref Delivery (Delivery_Id) | |
| | Fk_Order_Delivery_Mapping | (Order_Id) ref Order (Order_Id) | |

Table: Order_Product_Mapping

| Indexes | Field Name | Data Type | Description |
|---------------------|--------------------------|--|-------------|
| * | Order_Product_Id | integer | |
| * | Product_Id | integer | |
| * | Product_Quantity | integer | |
| * | Order_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Order_Product_Mapping | ON Order_Product_Id | |
| Foreign Keys | | | |
| | Fk_Order_Product_Mapping | (Product_Id, Product_Id) ref Product (Product_Id, Product_Id) | |

Table: Product

| Indexes | Field Name | Data Type | Description |
|---------------------|--------------------|--|-------------|
| * | Product_Id | integer | |
| * | Product_Name | varchar(2000000000) | |
| * | Product_Cost | float(2000000000, 10) | |
| * | Product_Details_Id | integer | |
| * | Category_Id | integer | |
| * | Supplier_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Product | ON Product_Id | |
| Foreign Keys | | | |
| | Fk_Product | (Category_Id) ref Category (Category_Id) | |
| | Fk_Product | (Product_Details_Id) ref Product Details (Product_Details_Id) | |
| | Fk_Product | (Supplier_Id) ref Supplier (Supplier_Id) | |

Table: Product_Category_Mapping

| Indexes | Field Name | Data Type | Description |
|---------------------|---|--|-------------|
| * | Product_Category_Id | integer | |
| * | Product_Id | integer | |
| * | Category_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Product_Category_Mapping | ON Product_Category_Id | |
| Foreign Keys | | | |
| | Fk_Product_Category_Mapping (Category_Id) | ref Category (Category_Id) | |
| | Fk_Product_Category_Mapping (Product_Id) | ref Product (Product_Id) | |

Table: Product_Details

| Indexes | Field Name | Data Type | Description |
|----------------|---------------------|-------------------------|-------------|
| * | Product_Details_Id | integer | |
| * | Product_Cost | integer | |
| * | Product_Description | text | |
| * | Product_Quantity | integer | |
| * | Product_Discount | float(2000000000, 10) | |
| * | deleteFlag | boolean | |
| | Product_Model_No | varchar(2000000000) | |
| Indexes | | | |
| | pk_Product_Details | ON Product_Details_Id | |

Table: Product_Supplier_Mapping

| Indexes | Field Name | Data Type | Description |
|---------------------|-----------------------------|------------------------|-------------|
| * | Product_Supplier_Id | integer | |
| * | Product_Id | integer | |
| * | Product_Quantity | integer | |
| * | Supplier_Id | integer | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Product_Supplier_Mapping | ON Product_Supplier_Id | |
| Foreign Keys | | | |

| Indexes | Field Name | Data Type | Description |
|---------|---|--|-------------|
| | Fk_Product_Supplier_Mapping (Product_Id) | ref Product (Product_Id) | |
| | Fk_Product_Supplier_Mapping (Supplier_Id) | ref Supplier (Supplier_Id) | |

Table: Supplier

| Indexes | Field Name | Data Type | Description |
|----------------|-------------------------|------------------------|-------------|
| * | Supplier_Id | integer | |
| * | Supplier_Name | varchar(20000000000) | |
| | Supplier_Contact_Number | integer | |
| * | Supplier_EmailId | varchar(20000000000) | |
| * | deleteFlag | boolean DEFAULT 0 | |
| Indexes | | | |
| | pk_Supplier | ON Supplier_Id | |

Queries and Results:

1. Create Operation:

Create Address table:

```
CREATE TABLE "Address" ("Address_Id" INTEGER PRIMARY KEY NOT NULL ,
"Address_Street" VARCHAR NOT NULL , "Address_Apt" INTEGER NOT NULL,
"Address_Place_Id" VARCHAR NOT NULL, "deleteFlag" BOOL NOT NULL DEFAULT 0,
FOREIGN KEY ("Address_Place_Id") REFERENCES Address_Place("Address_Place_Id"))
```

Create Address_Place table:

```
CREATE TABLE "Address_Place" ("Address_Place_Id" INTEGER PRIMARY KEY NOT NULL ,
"Address_City" VARCHAR NOT NULL , "Address_State" VARCHAR NOT NULL ,
"Address_Country" VARCHAR NOT NULL , "Address_Zip" INTEGER NOT NULL ,
"deleteFlag" BOOL NOT NULL DEFAULT 0)
```

Create Category table:

```
CREATE TABLE "Category" ("Category_Id" INTEGER PRIMARY KEY NOT NULL ,
"Category_Name" VARCHAR NOT NULL , "deleteFlag" BOOL NOT NULL DEFAULT 0)
```

Create Customer table:

```
CREATE TABLE "Customer" ("Customer_Id" INTEGER PRIMARY KEY NOT NULL
,"Customer_Name" VARCHAR NOT NULL ,"Customer_EmailId" VARCHAR NOT NULL
```

```
, "Customer_Contact_Number" INTEGER NOT NULL , "Address_Id" INTEGER, "deleteFlag"  
BOOL NOT NULL DEFAULT 0, FOREIGN KEY (Address_Id) REFERENCES  
Address(Address_Id))
```

Create Delivery table:

```
CREATE TABLE "Delivery" ("Delivery_Id" INTEGER PRIMARY KEY NOT NULL DEFAULT  
(null) , "Delivery_Date" DATE NOT NULL DEFAULT (null) , "Delivery_Person_Id" INTEGER  
NOT NULL , "deleteFlag" BOOL NOT NULL DEFAULT (0) , FOREIGN KEY  
(Delivery_Person_Id) REFERENCES Delivery_Person(Delivery_Person_Id))
```

Create Delivery_Person table:

```
CREATE TABLE "Delivery_Person" ("Delivery_Person_Id" INTEGER PRIMARY KEY NOT  
NULL , "Person_Name" VARCHAR NOT NULL , "Person_Contact_Number" INTEGER NOT  
NULL , "deleteFlag" BOOL NOT NULL DEFAULT (0) )
```

Create Order table:

```
CREATE TABLE "Order" ("Order_Id" INTEGER PRIMARY KEY NOT NULL  
, "Order_TimeStamp" DATETIME DEFAULT (CURRENT_TIMESTAMP)  
, "Order_Payment_Type" VARCHAR NOT NULL , "Customer_Id" INTEGER NOT NULL ,  
"deleteFlag" BOOL NOT NULL DEFAULT 0, FOREIGN KEY (Customer_Id) REFERENCES  
Customer(Customer_Id))
```

Create Order_Delivery_Mapping table:

```
CREATE TABLE Order_Delivery_Mapping ("Order_Delivery_Id" INTEGER PRIMARY KEY  
NOT NULL , "Order_Id" INTEGER NOT NULL , "Delivery_Id" INTEGER NOT NULL,  
"deleteFlag" BOOL NOT NULL DEFAULT 0, FOREIGN KEY ("Order_Id") REFERENCES  
"Order"("Order_Id"), FOREIGN KEY ("Delivery_Id") REFERENCES Delivery("Delivery_Id") )
```

Create Order_Product_Mapping table:

```
CREATE TABLE "Order_Product_Mapping" ("Order_Product_Id" INTEGER PRIMARY KEY  
NOT NULL , "Product_Id" INTEGER NOT NULL, "Product_Quantity" INTEGER NOT NULL ,  
"Order_Id" INTEGER NOT NULL, "deleteFlag" BOOL NOT NULL DEFAULT 0 , FOREIGN KEY  
(Product_Id) REFERENCES Product(Product_Id), FOREIGN KEY (Product_Id) REFERENCES  
Product(Product_Id))
```

Create Product table:


```
CREATE TABLE "Product" ("Product_Id" INTEGER PRIMARY KEY NOT NULL ,  
"Product_Name" VARCHAR NOT NULL , "Product_Cost" FLOAT NOT NULL ,  
"Product_Details_Id" INTEGER NOT NULL , "Category_Id" INTEGER NOT NULL ,  
"Supplier_Id" INTEGER NOT NULL , "deleteFlag" BOOL NOT NULL DEFAULT 0, FOREIGN  
KEY (Product_Details_Id) REFERENCES Product_Details(Product_Details_Id), FOREIGN KEY  
(Category_Id) REFERENCES Category(Category_Id) , FOREIGN KEY (Supplier_Id)  
REFERENCES Supplier(Supplier_Id))
```

Create Product_Category_Mapping table:

```
CREATE TABLE "Product_Category_Mapping" ("Product_Category_Id" INTEGER PRIMARY  
KEY NOT NULL , "Product_Id" INTEGER NOT NULL , "Category_Id" INTEGER NOT NULL,  
"deleteFlag" BOOL NOT NULL DEFAULT 0, FOREIGN KEY (Category_Id) REFERENCES  
Category(Category_Id) , FOREIGN KEY (Product_Id) REFERENCES Product(Product_Id) )
```

Create Product_Details table:

```
CREATE TABLE "Product_Details" ("Product_Details_Id" INTEGER PRIMARY KEY NOT  
NULL , "Product_Cost" INTEGER NOT NULL , "Product_Description" TEXT NOT NULL ,  
"Product_Quantity" INTEGER NOT NULL , "Product_Discount" FLOAT NOT NULL ,  
"deleteFlag" BOOL NOT NULL , "Product_Model_No" VARCHAR)
```

Create Product_Supplier_Mapping table:

```
CREATE TABLE "Product_Supplier_Mapping" ("Product_Supplier_Id" INTEGER PRIMARY  
KEY NOT NULL , "Product_Id" INTEGER NOT NULL, "Product_Quantity" INTEGER NOT  
NULL , "Supplier_Id" INTEGER NOT NULL, "deleteFlag" BOOL NOT NULL DEFAULT 0 ,  
FOREIGN KEY (Product_Id) REFERENCES Product(Product_Id), FOREIGN KEY (Supplier_Id)  
REFERENCES Supplier(Supplier_Id))
```

Create Supplier table:

```
CREATE TABLE "Supplier" ("Supplier_Id" INTEGER PRIMARY KEY NOT NULL ,  
"Supplier_Name" VARCHAR NOT NULL , "Supplier_Contact_Number" INTEGER,  
"Supplier_EmailId" VARCHAR NOT NULL , "deleteFlag" BOOL NOT NULL DEFAULT 0)
```

2. Select Operation:

NATURAL Join of Delivery and Delivery_Person Tables:

Enter SQL Select | Data Manipulation | Create/Alter | Drop | Reindex | PRAGMA

```
SELECT *
FROM Delivery
NATURAL JOIN Delivery_Person;
```

Last Error: not an error

| Delivery_Id | Delivery_Date | Delivery_Person_Id | deleteFlag | Person_Name | Person_Contact_Number |
|-------------|---------------|--------------------|------------|-------------|-----------------------|
| 1 | 2017-09-21 | 1 | 0 | Ravi | 958-975-6423 |
| 2 | 2017-09-22 | 2 | 0 | Ram | 896-564-7896 |
| 3 | 2017-09-23 | 3 | 0 | Shashikant | 408-569-5647 |
| 4 | 2017-09-24 | 4 | 0 | Alan | 669-564-6532 |
| 5 | 2017-09-25 | 5 | 0 | Kyle | 408-569-8976 |

INNER Join of Product and Product_Delivery Tables:

Enter SQL Select | Data Manipulation | Create/Alter | Drop | Relr

```
SELECT *
FROM Product INNER JOIN Product_Details
ON Product.Product_Details_Id = Product_Details.Product_Details_Id;
```

Last Error: not an error

| Product_Id | Product_Name | Product_Details_Id | Category_Id | Supplier_Id | deleteFlag | Product_Details_Id | Product_Cost | Product_Description | Product_Quantity | Product_Discount | deleteFlag | Product_Model_No |
|------------|--------------|--------------------|-------------|-------------|------------|--------------------|--------------|---------------------|------------------|------------------|------------|------------------|
| 1 | iPhone X | 1 | 3 | 5 | 0 | 1 | 999 | iPhone X | 50 | 0 | | 6986598 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 2 | 500 | iPhone 8 | 25 | 5 | | 56984562 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 3 | 1000 | Apple watch | 10 | 10 | | 1236489 |

OUTER Join of Customer and Address Tables:

Enter SQL Select | Data Manipulation | Create/Alter | Drop | ReIndex | PRAGMA

```
SELECT * FROM Customer , Address
WHERE Customer.Address_id = Address.Address_id;
```

Last Error: not an error

| Custom... | Customer_... | Customer_... | Customer_... | Address_Id | deleteFlag | Address_Id | Address_S... | Address_A... | Address_Pl... | deleteFlag |
|-----------|---------------|---------------|--------------|------------|------------|------------|---------------|--------------|---------------|------------|
| 1 | Vishwesh P... | Vishwesh@... | 8542369871 | 1 | 0 | 1 | 1234 The A... | 256 | 2 | 0 |
| 2 | Rajesh Kha... | rajesh@ya... | 5623124569 | 2 | 0 | 2 | 265 The Ro... | 542 | 1 | 0 |
| 3 | Ram Malh... | ram@yaho... | 4085698532 | 3 | 0 | 3 | 101 San Fa... | 145 | 3 | 0 |
| 4 | Keval Shah | keval@outl... | 8562314569 | 3 | 0 | 3 | 101 San Fa... | 145 | 3 | 0 |
| 5 | Kunal Gos... | kunal@outl... | 6698523654 | 4 | 0 | 4 | 356 El Cam... | 365 | 4 | 0 |

CROSS Join of Product and Category Tables:

Enter SQL Select | Data Manipulation | Create/Alter | Drop | ReIndex | PRAGMA

SELECT *
FROM Product
CROSS JOIN Category;

Run SQL Actions Last Error: not an error

| Product_Id | Product_Name | Product_Details_Id | Category_Id | Supplier_Id | deleteFlag | Category_Id | Category_Name | deleteFlag |
|------------|--------------|--------------------|-------------|-------------|------------|-------------|-----------------------|------------|
| 1 | iPhone X | 1 | 3 | 5 | 0 | 1 | Music & Entertainment | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 2 | Books | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 3 | Electronics | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 4 | Computers | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 5 | Home Appliances | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 6 | Home Decoration | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 7 | Food & Grocery | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 8 | Beauty & Health | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 9 | Apperals | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 10 | Sports Gears | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 1 | Music & Entertainment | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 2 | Books | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 3 | Electronics | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 4 | Computers | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 5 | Home Appliances | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 6 | Home Decoration | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 7 | Food & Grocery | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 8 | Beauty & Health | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 9 | Apperals | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 10 | Sports Gears | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 1 | Music & Entertainment | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 2 | Books | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 3 | Electronics | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 4 | Computers | 0 |

Enter SQL Select | Data Manipulation | Create/Alter | Drop | ReIndex | PRAGMA

SELECT *
FROM Product
CROSS JOIN Category;

Run SQL Actions Last Error: not an error

| Product_Id | Product_Name | Product_Details_Id | Category_Id | Supplier_Id | deleteFlag | Category_Id | Category_Name | deleteFlag |
|------------|--------------|--------------------|-------------|-------------|------------|-------------|-----------------------|------------|
| 1 | iPhone X | 1 | 3 | 5 | 0 | 8 | Beauty & Health | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 9 | Apperals | 0 |
| 1 | iPhone X | 1 | 3 | 5 | 0 | 10 | Sports Gears | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 1 | Music & Entertainment | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 2 | Books | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 3 | Electronics | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 4 | Computers | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 5 | Home Appliances | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 6 | Home Decoration | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 7 | Food & Grocery | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 8 | Beauty & Health | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 9 | Apperals | 0 |
| 2 | iPhone 8 | 2 | 6 | 2 | 0 | 10 | Sports Gears | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 1 | Music & Entertainment | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 2 | Books | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 3 | Electronics | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 4 | Computers | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 5 | Home Appliances | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 6 | Home Decoration | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 7 | Food & Grocery | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 8 | Beauty & Health | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 9 | Apperals | 0 |
| 3 | Apple Watch | 3 | 9 | 1 | 0 | 10 | Sports Gears | 0 |

3. Update Operation:

Update Customer name to 'Vishweshkumar Patel' from 'Vishwesh Patel' :

UPDATE Customer SET Customer_Name = "Vishweshkumar Patel" WHERE
Customer_Name = "Vishwesh Patel";

4. Delete Operation:

Delete Category having Category_Id '9':

```
DELETE * FROM Category WHERE Category_Id = 9;
```

5. Insert Operation:

Insert Product_Details of Apple Mac Book:

```
INSERT INTO "main"."Product_Details"  
("Product_Cost","Product_Description","Product_Quantity","Product_Discount","deleteFlag","Product_Model_No") VALUES (?1,?2,?3,?4,?5,?6)
```

Parameters:

param 1 (integer): 250

param 2 (text): Apple Mac Book

param 3 (integer): 100

param 4 (integer): 25

param 5 (integer): 0

param 6 (text): Mac-5689

Insert Product Apple Mac Book:

```
INSERT INTO "main"."Product"  
("Product_Name","Product_Details_Id","Category_Id","Supplier_Id") VALUES  
(?1,?2,?3,?4)
```

Parameters:

param 1 (text): Apple Mac Book

param 2 (integer): 4

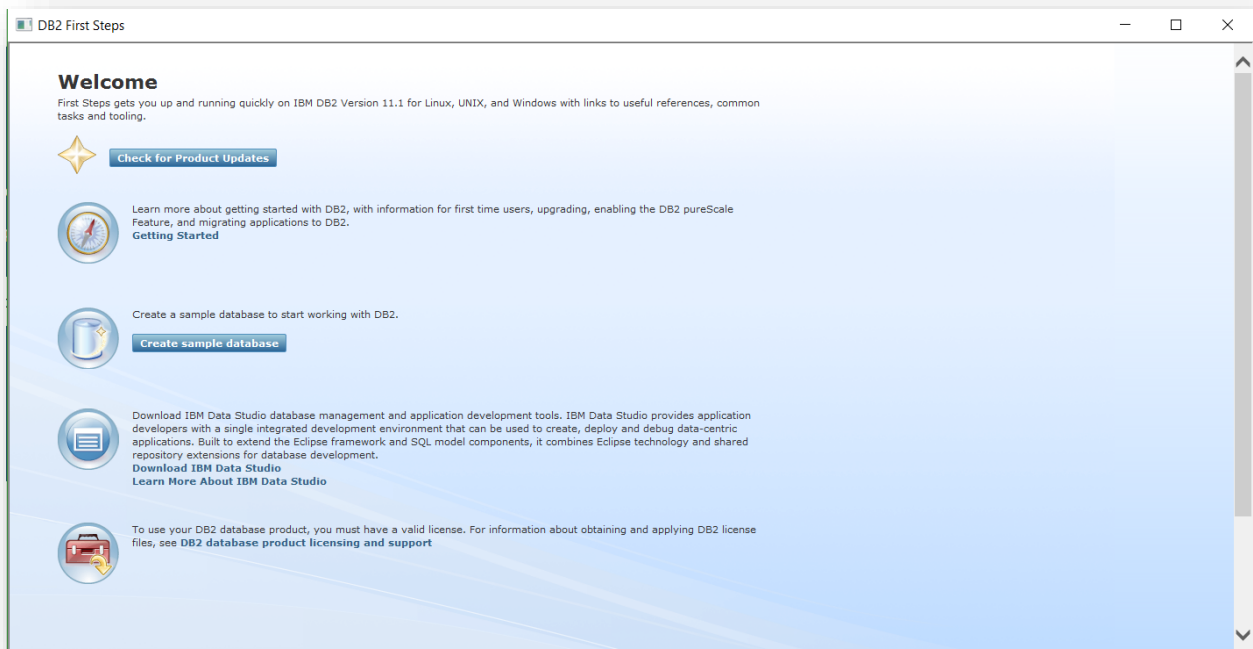
param 3 (integer): 4

param 4 (integer): 3

Db2 Express C

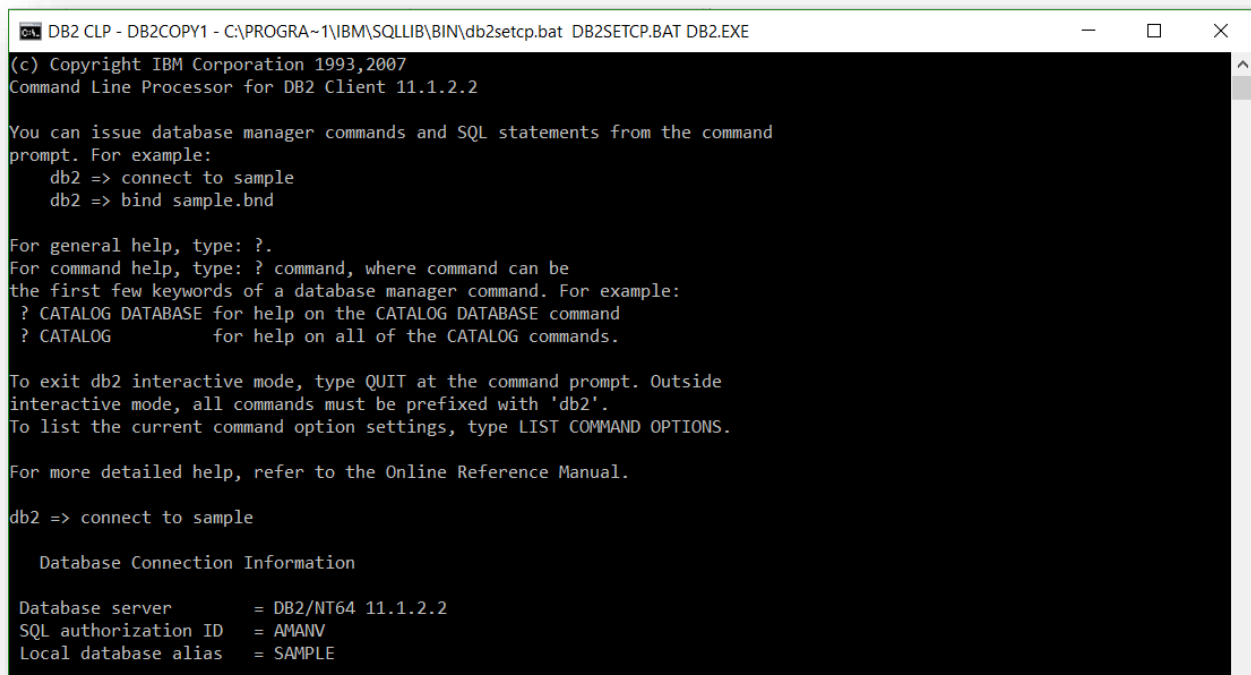
Installation

- Installing Db2 Express C is an easy process.
- Go to <https://www.ibm.com/us-en/marketplace/db2-express-c>
- Sign In using an existing account or Sign Up using your .edu email id.
- A zip file will be downloaded which will take around 5-10 minutes.
- The installation process is self-explanatory.
- You also get an option to setup database user and password.
- First Screen:



Sample Database Creation

- A sample database is created while installing the software.
- Also, we can create our own sample data base using the db2sampl command.
- To access the Sample database, we have to do the following steps:
 - ✓ Go to Start >> IBM Command Line Processor Plus
 - ✓ Type 'connect to sample'



```
DB2 CLP - DB2COPY1 - C:\PROGRA~1\IBM\SQLLIB\BIN\db2setcp.bat DB2SETCP.BAT DB2.EXE
(c) Copyright IBM Corporation 1993,2007
Command Line Processor for DB2 Client 11.1.2.2

You can issue database manager commands and SQL statements from the command
prompt. For example:
  db2 => connect to sample
  db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
  ? CATALOG DATABASE for help on the CATALOG DATABASE command
  ? CATALOG           for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 => connect to sample

Database Connection Information

Database server      = DB2/NT64 11.1.2.2
SQL authorization ID = AMANV
Local database alias = SAMPLE
```

- Running queries is quite similar to other databases.
- A sample query is written and the following output is generated.

Select * from staff;

DB2 CLP - DB2COPY1 - C:\PROGRA~1\IBM\SQLLIB\BIN\db2setcp.bat DB2SETCP.BAT DB2.EXE

db2 => select * from staff

| ID | NAME | DEPT | JOB | YEARS | SALARY | COMM |
|-----|-----------|------|-------|-------|----------|---------|
| 10 | Sanders | 20 | Mgr | 7 | 98357.50 | - |
| 20 | Pernal | 20 | Sales | 8 | 78171.25 | 612.45 |
| 30 | Marengi | 38 | Mgr | 5 | 77506.75 | - |
| 40 | O'Brien | 38 | Sales | 6 | 78006.00 | 846.55 |
| 50 | Hanes | 15 | Mgr | 10 | 80659.80 | - |
| 60 | Quigley | 38 | Sales | - | 66808.30 | 650.25 |
| 70 | Rothman | 15 | Sales | 7 | 76502.83 | 1152.00 |
| 80 | James | 20 | Clerk | - | 43504.60 | 128.20 |
| 90 | Koonitz | 42 | Sales | 6 | 38001.75 | 1386.70 |
| 100 | Plotz | 42 | Mgr | 7 | 78352.80 | - |
| 110 | Ngan | 15 | Clerk | 5 | 42508.20 | 206.60 |
| 120 | Naughton | 38 | Clerk | - | 42954.75 | 180.00 |
| 130 | Yamaguchi | 42 | Clerk | 6 | 40505.90 | 75.60 |
| 140 | Fraye | 51 | Mgr | 6 | 91150.00 | - |
| 150 | Williams | 51 | Sales | 6 | 79456.50 | 637.65 |
| 160 | Molinare | 10 | Mgr | 7 | 82959.20 | - |
| 170 | Kermisch | 15 | Clerk | 4 | 42258.50 | 110.10 |
| 180 | Abrahams | 38 | Clerk | 3 | 37009.75 | 236.50 |
| 190 | Sneider | 20 | Clerk | 8 | 34252.75 | 126.50 |
| 200 | Scoutten | 42 | Clerk | - | 41508.60 | 84.20 |
| 210 | Lu | 10 | Mgr | 10 | 90010.00 | - |
| 220 | Smith | 51 | Sales | 7 | 87654.50 | 992.80 |
| 230 | Lundquist | 51 | Clerk | 3 | 83369.80 | 189.65 |
| 240 | Daniels | 10 | Mgr | 5 | 79260.25 | - |
| 250 | Wheeler | 51 | Clerk | 6 | 74460.00 | 513.30 |
| 260 | Jones | 10 | Mgr | 12 | 81234.00 | - |
| 270 | Lea | 66 | Mgr | 9 | 88555.50 | - |
| 280 | Wilson | 66 | Sales | 9 | 78674.50 | 811.50 |
| 290 | Quill | 84 | Mgr | 10 | 89818.00 | - |
| 300 | Davis | 84 | Sales | 5 | 65454.50 | 806.10 |
| 310 | Graham | 66 | Sales | 13 | 71000.00 | 200.30 |
| 320 | Gonzales | 66 | Sales | 4 | 76858.20 | 844.00 |
| 330 | Burke | 66 | Clerk | 1 | 49988.00 | 55.50 |
| 340 | Edwards | 84 | Sales | 7 | 67844.00 | 1285.00 |
| 350 | Gafney | 84 | Clerk | 5 | 43030.50 | 188.00 |

35 record(s) selected.

Sample Query

- As mentioned before, writing queries is very simple in Db2.
- The outputs are quick and takes very little processing time.
- A simple query was run and output is shown below.

```
Ca\ DB2 CLP - DB2COPY1 - C:\PROGRA~1\IBM\SQLLIB\BIN\db2setcp.bat DB2SETCP.BAT DB2.EXE

-----
10 Sanders      20 Mgr      7 98357.50      -
20 Pernal       20 Sales    8 78171.25    612.45
30 Marenghi     38 Mgr      5 77506.75      -
40 O'Brien     38 Sales    6 78006.00    846.55
50 Hanes        15 Mgr     10 80659.80      -
60 Quigley      38 Sales    - 66808.30    650.25
70 Rothman     15 Sales    7 76502.83   1152.00
80 James        20 Clerk    - 43504.60    128.20
90 Koonitz      42 Sales    6 38001.75   1386.70
100 Plotz       42 Mgr      7 78352.80      -
110 Ngan        15 Clerk    5 42508.20    206.60
120 Naughton    38 Clerk    - 42954.75    180.00
130 Yamaguchi   42 Clerk    6 40505.90     75.60
140 Fraye       51 Mgr      6 91150.00      -
150 Williams    51 Sales    6 79456.50    637.65
160 Molinare    10 Mgr      7 82959.20      -
170 Kermisch    15 Clerk    4 42258.50    110.10
180 Abrahams    38 Clerk    3 37009.75    236.50
190 Sneider     20 Clerk    8 34252.75    126.50
200 Scoutten    42 Clerk    - 41508.60     84.20
210 Lu          10 Mgr     10 90010.00      -
220 Smith       51 Sales    7 87654.50    992.80
230 Lundquist   51 Clerk    3 83369.80    189.65
240 Daniels     10 Mgr      5 79260.25      -
250 Wheeler     51 Clerk    6 74460.00    513.30
260 Jones       10 Mgr     12 81234.00      -
270 Lea         66 Mgr      9 88555.50      -
280 Wilson      66 Sales    9 78674.50    811.50
290 Quill       84 Mgr     10 89818.00      -
300 Davis       84 Sales    5 65454.50    806.10
310 Graham      66 Sales   13 71000.00    200.30
320 Gonzales    66 Sales    4 76858.20    844.00
330 Burke       66 Clerk    1 49988.00     55.50
340 Edwards     84 Sales    7 67844.00   1285.00
350 Gafney      84 Clerk    5 43030.50    188.00

35 record(s) selected.

db2 => select count(DEPT), JOB from staff group by JOB

1          JOB
-----
          12 Clerk
          11 Mgr
          12 Sales

3 record(s) selected.
```


Query explain Plan

- To generate the query explain plan, we have to run the 'EXPLAIN.DDL' from the misc directory.

```
C:\Program Files\IBM\SQLLIB\MISC>db2 -tf EXPLAIN.DDL

***** IMPORTANT *****

USAGE: db2 -tf EXPLAIN.DDL

***** IMPORTANT *****
```

- Next, we have to set the current explain mode flag and current explain snapshot flag to 'yes' by using the following commands:

```
db2 set current explain mode yes
db2 set current explain snapshot yes
```

- Next, we need to execute a query for which we need the Query explain plan.
- Finally, we can run the db2exfmt command to generate the plan in an output file.

```
C:\Program Files\IBM\SQLLIB\BIN>db2exfmt
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

Enter Database Name ==> SAMPLE
Connecting to the Database.
Connect to Database Successful.
Binding package - Bind was Successful
Enter up to 26 character Explain timestamp (Default -1) ==>
Enter up to 128 character source name (SOURCE_NAME, Default %) ==>
Enter source schema (SOURCE_SCHEMA, Default %) ==>
Enter section number (0 for all, Default 0) ==>
Enter outfile name. Default is to terminal ==> output.txt
Output is in output.txt.
Executing Connect Reset -- Connect Reset was Successful.

C:\Program Files\IBM\SQLLIB\BIN>
```

IBM Bluemix

The restaurant graph database application service is designed to show relation between entities like Person, Restaurant and Location.

Diagram shows different entities and their attributes.

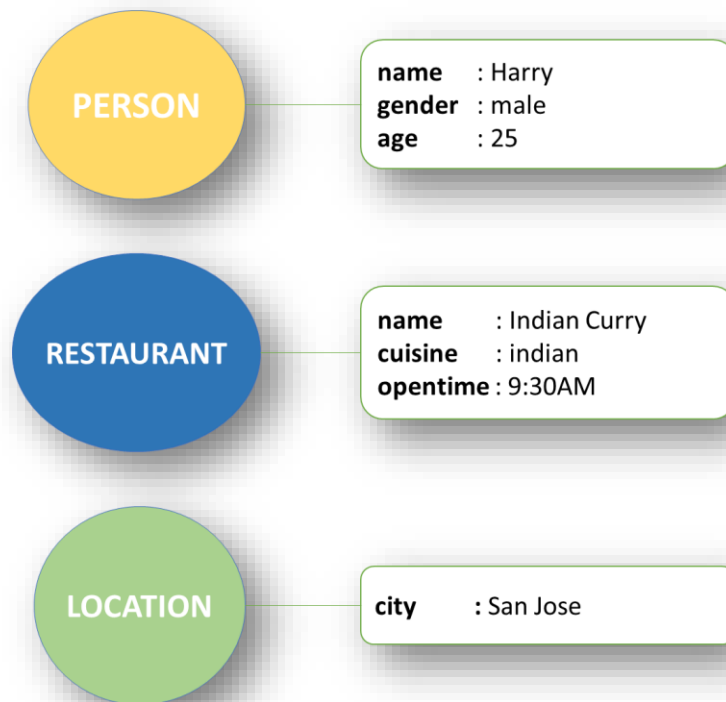


Figure 1: Entities and respective attributes

The dataset model below represents the relationships between different entities.

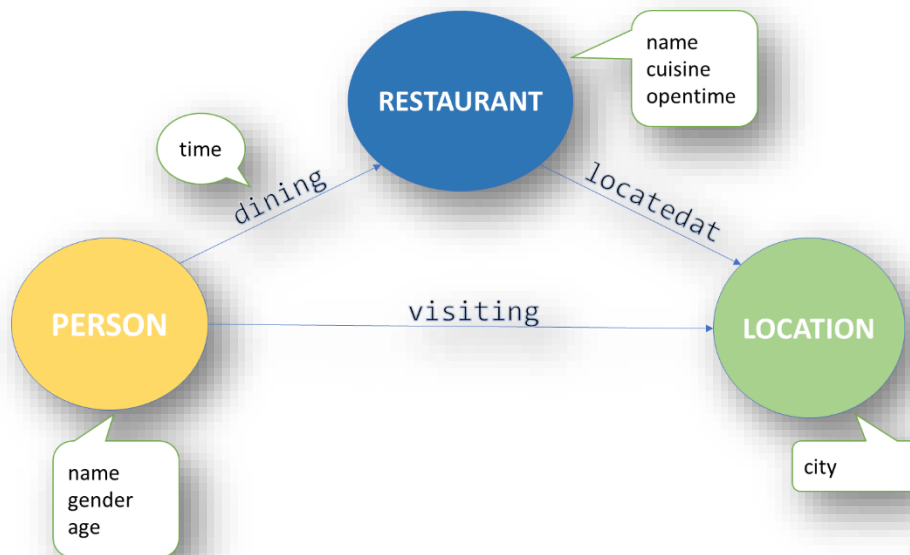


Figure 2: Dataset model

SCHEMA:

```
SCHEMA='
{
  "propertyKeys": [
    {"name": "name", "dataType": "String", "cardinality": "SINGLE"},
    {"name": "cuisine", "dataType": "String", "cardinality": "SINGLE"},
    {"name": "opentime", "dataType": "String", "cardinality": "SINGLE"},
    {"name": "city", "dataType": "String", "cardinality": "SINGLE"},
    {"name": "gender", "dataType": "String", "cardinality": "SINGLE"},
    {"name": "age", "dataType": "Integer", "cardinality": "SINGLE"},
    {"name": "time", "dataType": "String", "cardinality": "SINGLE"}
  ],
  "vertexLabels": [
    {"name": "person"},
    {"name": "restaurant"},
    {"name": "location"}
  ],
  "edgeLabels": [
    {"name": "locatedat", "multiplicity": "MULTI"},
    {"name": "visiting", "multiplicity": "MULTI"},
    {"name": "dinning", "multiplicity": "MULTI"}
  ]
}
```

```
],  
"vertexIndexes": [  
  {"name": "vByName", "propertyKeys": ["name"], "composite": true, "unique": true},  
  {"name": "vByCuisine", "propertyKeys": ["cuisine"], "composite": true, "unique": false},  
  {"name": "vByOpentime", "propertyKeys": ["opentime"], "composite": true, "unique": false},  
  {"name": "vByCity", "propertyKeys": ["city"], "composite": true, "unique": false},  
  {"name": "vBygender", "propertyKeys": ["gender"], "composite": true, "unique": false},  
  {"name": "vByAge", "propertyKeys": ["age"], "composite": true, "unique": false}  
],  
"edgeIndexes": [  
  {"name": "eByTime", "propertyKeys": ["time"], "composite": true, "unique": false}  
]  
}'
```

- We must setup a schema before creating the data which defines the datatypes and indexes for the properties used while data acquisition. Every property, let it be a vertex or an edge property, requires an index. Schemas once created neither be updated or be appended in any case.
- The above schema represents the dataset we have used in our application. Our schema describes a dataset consisting information of some American restaurants with multiple cuisines to offer, their visitors and the city it's located in. Some of the restaurants can have visitors in common, even though situated in two different cities. Using a gremlin query over the IBM Graph, we can easily track frequent visitors and their information. Moreover, restaurants can be classified based on cuisine they had to offer in that locality.
- The entities mentioned above constitutes restaurant, person and location respectively. A typical restaurant entity stores the restaurant's name, it's cuisine and opening time, whereas a person entity stores the individual's name, its gender and age. The location entity is nothing but the city name where the restaurant is located. An edge from person to restaurant signifies the person dinning at a mentioned time, whereas that from a person to location states him/her visiting that city. A link between a restaurant and location can be interpreted as restaurant's address.

So, let us define the vertices and edges of our database graph.

```
at << ENDGREMLIN >gremlin.json # write everything until ENDGREMLIN into gremlin.json
{
  "gremlin": "
def harry = graph.addVertex('name','Harry',label,'person','gender','male','age',25);
def michael = graph.addVertex('name','Michael',label,'person','gender','male','age',30);
def ron = graph.addVertex('name','Ron',label,'person','gender','male','age',28);
def kate = graph.addVertex('name','Kate',label,'person','gender','male','age',20);

def indiancurry = graph.addVertex('name','Indian Curry', label, 'restaurant', 'cuisine', 'indian',
'opentime', '9:30');
def tandoori = graph.addVertex('name','Tandoori Hut', label, 'restaurant', 'cuisine', 'indian',
'opentime', '10:30');
def punjab = graph.addVertex('name','Punjab Cafe', label, 'restaurant', 'cuisine', 'indian', 'opentime',
'8:30');
def panda = graph.addVertex('name','Panda Express', label, 'restaurant', 'cuisine', 'chinese',
'opentime', '10:30');
def chipotle = graph.addVertex('name','Chipotle', label, 'restaurant', 'cuisine', 'mexican', 'opentime',
'11:30');

def sanjose = graph.addVertex('city','San Jose',label,'location');
def sf = graph.addVertex('city','San Francisco',label,'location');
def sc = graph.addVertex('city','Santa Clara',label,'location');
indiancurry.addEdge('locatedat',sanjose);

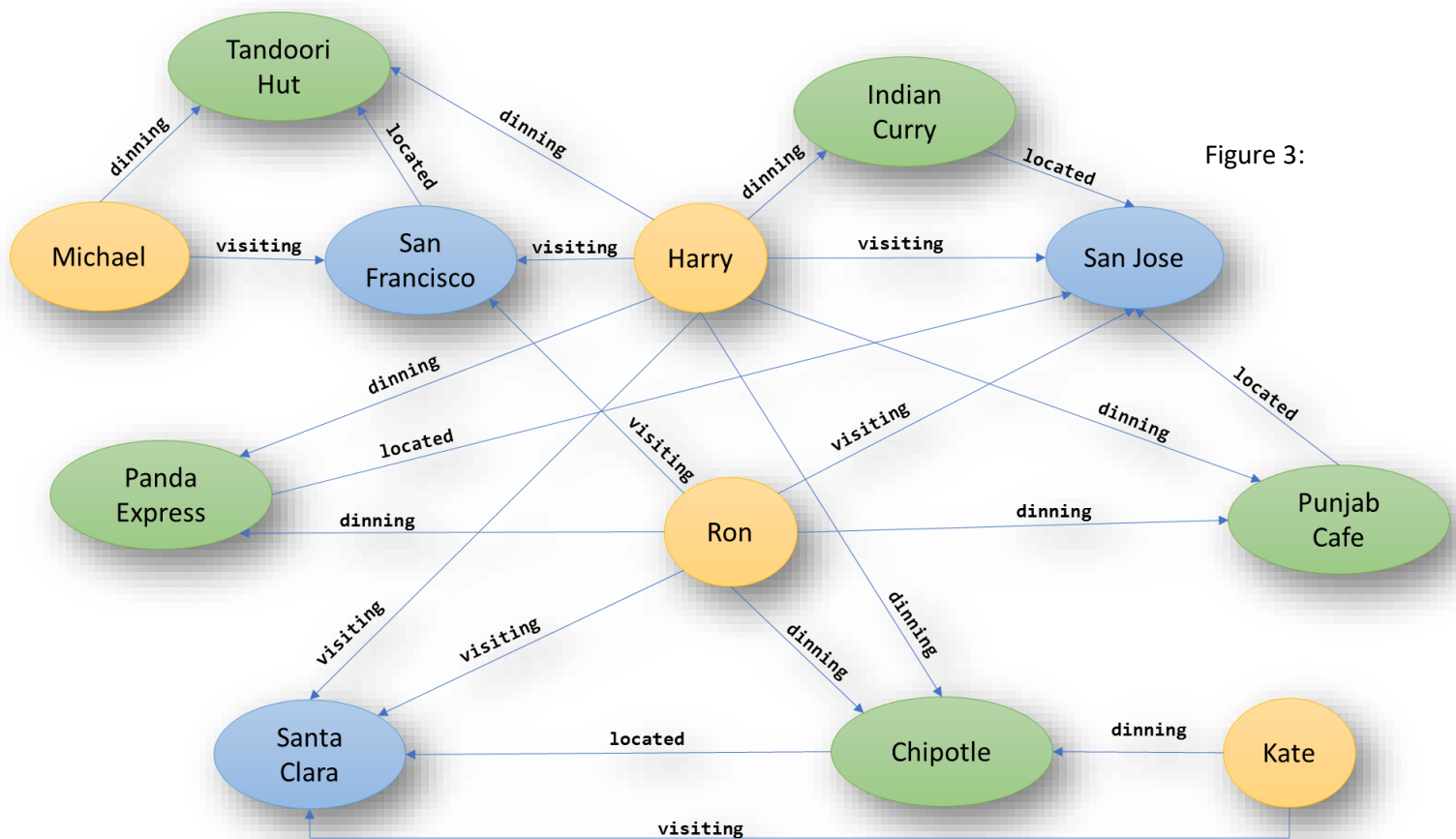
punjab.addEdge('locatedat',sanjose);
tandoori.addEdge('locatedat',sf);
panda.addEdge('locatedat',sanjose);
chipotle.addEdge('locatedat',sc);

harry.addEdge('visiting',sanjose);
harry.addEdge('visiting',sf);
harry.addEdge('visiting',sc);
michael.addEdge('visiting',sf);
kate.addEdge('visiting',sc);
ron.addEdge('visiting',sanjose);
ron.addEdge('visiting',sf);
ron.addEdge('visiting',sc);

harry.addEdge('dinning',indiancurry,'time', '8:30PM');
harry.addEdge('dinning',tandoori, 'time', '9:30PM');
harry.addEdge('dinning',punjab, 'time', '9:00PM');
harry.addEdge('dinning',panda, 'time', '10:00PM');
harry.addEdge('dinning',chipotle, 'time', '08:00PM');
michael.addEdge('dinning',tandoori, 'time', '08:30PM');
ron.addEdge('dinning',punjab, 'time', '9:30PM');
```

```
ron.addEdge('dinning',panda, 'time', '9:30PM');  
ron.addEdge('dinning',chipotle, 'time', '08:30PM');  
kate.addEdge('dinning',chipotle, 'time', '08:30PM');  
"  
}  
ENDGREMLIN
```

Below graph gives you a basic idea of what the structure would be like, upon successful loading of data.



Database graph - Interaction diagram

The sample query set performed on the graph database are as follows:

1. All the people who visited San Francisco

```
def gt = graph.traversal();  
gt.V().hasLabel('location').has('city', 'San Francisco').inE('visiting').outV().path();
```

The screenshot displays a graph database query interface. At the top, a query is entered in a text area:

```
1 def gt = graph.traversal();  
2 gt.V().hasLabel('location').has('city', 'San Francisco').inE('visiting').outV().path();
```

Below the query area, the results are shown in a JSON format:

```
def gt = graph.traversal();gt.V().hasLabel('location').has('city', 'San Francisco').inE('visit
```

The JSON output is expanded to show the following structure:

```
{  
  "labels": [  
    [],  
    [],  
    []  
  ],  
  "objects": [  
    {  
      "id": 4208,  
      "label": "location",  
      "type": "vertex",  
      "properties": {  
        "city": [  
          "San Francisco"  
        ]  
      }  
    }  
  ]  
}
```

At the bottom left, there are filter buttons: "Filter:", "Label", "Type", and "Properties". At the bottom right, it says "Vertices: 4".

On the right side of the interface, a graph visualization is shown. It consists of four nodes: a yellow node labeled "person", a green node labeled "locat.", a purple node labeled "person", and a blue node labeled "person". The yellow "person" node is connected to the green "locat." node. The green "locat." node is connected to the purple "person" node. The blue "person" node is connected to the green "locat." node.

```
def gt = graph.traversal();
gt.V().hasLabel('location').has('city', 'San
Jose').inE('locatedat').outV().has("cuisine", "indian").inE("dinning").outV().path();
```



Vertices: **5**

3. List all the restaurants where Indian cuisine is provided

```
def gt = graph.traversal();  
gt.V().hasLabel("restaurant").has("cuisine","indian");
```

The screenshot displays a graph query interface. At the top, a code editor contains the following Cypher query:

```
1 def gt = graph.traversal();  
2 gt.V().hasLabel("restaurant").has("cuisine","indian");
```

Below the code editor, the query is executed, and the results are shown in a JSON format on the left and a visual graph on the right.

JSON Results:

```
[  
  {  
    "id": 4192,  
    "label": "restaurant",  
    "type": "vertex",  
    "properties": {  
      "name": [  
        {  
          "id": "170-38g-s1",  
          "value": "Indian Curry"  
        }  
      ],  
      "cuisine": [  
        {  
          "id": "118-38g-111",  
          "value": "Indian Curry"  
        }  
      ]  
    }  
  }  
]
```

Visual Graph:

The visual graph shows three vertices, each labeled "resta.". One vertex is purple, one is blue, and one is green. They are arranged in a triangular pattern.

Filter: Label Type Properties

Vertices: 3

4. List all the restaurants where Indian cuisine is provided - path

```
def gt = graph.traversal();  
gt.V().hasLabel("restaurant").has("cuisine","indian").outE('locatedat').inV().path();
```

The screenshot displays a graph query interface. On the left, a code editor shows the following Cypher query:

```
1  def gt = graph.traversal();  
2  gt.V().hasLabel("restaurant").has("cuisine","indian").outE('locatedat').inV().path();
```

Below the code editor, a JSON-like structure represents the query results for a single vertex:

```
1  {  
2    "labels": [  
3      "restaurant"  
4    ],  
5    "objects": [  
6      {  
7        "id": 4192,  
8        "label": "restaurant",  
9        "type": "vertex",  
10       "properties": {  
11         "name": [  
12           "11", "17A 30E 41"  
13         ]  
14       }  
15     }  
16   ]  
17 }
```

At the bottom left, there is a filter section with three buttons: "Label", "Type", and "Properties".

On the right side of the interface, a graph visualization shows five vertices and their connections. The vertices are labeled "resta." and "locat.". The connections are represented by lines between the vertices.

At the bottom right, it says "Vertices: 5".

5. People dining at restaurants offering Indian cuisine

```
def gt = graph.traversal();  
gt.V().hasLabel("restaurant").has("cuisine","indian").inE('dinning').outV().path();
```

```
1  def gt = graph.traversal();  
2  gt.V().hasLabel("restaurant").has("cuisine","indian").inE('dinning').outV().path();  
def gt = graph.traversal();gt.V().hasLabel("restaurant").has("cuisine","indian").inE('dinning').outV().path();
```

1 [

2 {

3 "labels": [

4 [],

5 [],

6 []

7],

8 "objects": [

9 {

10 "id": 4192,

11 "label": "restaurant",

12 "type": "vertex",

13 "properties": {

14 "name": [

15 {

16 "id": "170-20-41"

Filter:

Vertices: 6

6. Get all places where Ron has visited – vertices

```
def gt = graph.traversal();  
gt.V().hasLabel("person").has("name", "Ron").outE("visiting").inV().hasLabel("location").path();
```

```
1  def gt = graph.traversal();  
2  gt.V().hasLabel("person").has("name", "Ron").outE("visiting").inV().hasLabel("location").path();
```

```
def gt = graph.traversal();gt.V().hasLabel("person").has("name", "Ron").outE("visiting").inV()
```

```
1  [  
2  {  
3    "labels": [  
4      [],  
5      [],  
6      []  
7    ],  
8    "objects": [  
9      {  
10       "id": 4176,  
11       "label": "person",  
12       "type": "vertex",  
13       "properties": {  
14         "gender": [  
15           {  
16             "id": "116-288-2-2"
```

Filter:



Vertices: 4

7. Get all places where Ron has visited - values

```
def gt = graph.traversal();  
gt.V().hasLabel("person").has("name", "Ron").out("visiting").values('city');
```

```
1  def gt = graph.traversal();  
2  gt.V().hasLabel("person").has("name", "Ron").out("visiting").values('city');
```



Graph: restaurant-application1506302925

```
def gt = graph.traversal();gt.V().hasLabel("person").has("name", "Ron").out("visiting").values
```



```
1  [  
2    "San Jose",  
3    "San Francisco",  
4    "Santa Clara"  
5  ]
```