***   EXPERIMENT NO: 09   ***

Authors : Akshat Ganesh Chandak , Shreya Agnihotri

Roll No : 35, 13 [5B]

Group No : 05

Date : 07-November-2020

AIM: To study any one NoSQL database and develop an application that incorporates NoSQL database access

PROBLEM STATEMENT:

Each group will identify and propose an application that will incorporate a NoSQL database(s). Additionally, the group members are free to use and integrate available software applications and state-of-art technologies they are good at.

INTRODUCTION:

When people use the term "NoSQL database", they typically use it to refer to any non-relational database. Some say the term "NoSQL" stands for "non-SQL" while others say it stands for "not only SQL." Either way, most agree that NoSQL databases are databases that store data in a format other than relational tables.

NoSQL data models allow related data to be nested within a single data structure.

Structured Data: Structured data is usually text files, with defined column titles and data in rows. Such data can easily be visualized in form of charts and can be processed using data mining tools.

Unstructured Data: Unstructured data can be anything like video file, image file, PDF, Emails etc. Structured Information can be extracted from unstructured data, but the process is time consuming. And as more and more modern data is unstructured, there was a need to have something to store such data for growing applications, hence setting path for NoSQL.

NoSQL, as many of you may already know, is basically a database used to manage huge sets of unstructured data, where in the data is not stored in tabular relations like relational databases. Most of the currently existing Relational Databases have failed in solving some of the complex modern problems like:

- Continuously changing nature of data - structured, semi-structured, unstructured and polymorphic data.

- Applications now serve millions of users in different geo-locations, in different time zones and have to be up and running all the time, with data integrity maintained
- Applications are becoming more distributed with many moving towards cloud computing.

NoSQL databases emerged in the late 2000s as to address these issues

NoSQL plays a vital role in an enterprise application which needs to access and analyse a massive set of data that is being made available on multiple virtual servers (remote based) in the cloud infrastructure and mainly when the data set is not structured. Hence, the NoSQL database is designed to overcome the Performance, Scalability, Data Modelling and Distribution limitations that are seen in the Relational Databases.

Following are the NoSQL database types:

1. **Document Databases:** In this type, key is paired with a complex data structure called as Document. Example: MongoDB
2. **Graph stores:** This type of database is usually used to store networked data. Wherein we can relate data based on some existing data.
3. **Key-Value stores:** These are the simplest NoSQL databases. In this each is stored with a key to identify it. In some Key-value databases, we can even save the type of the data saved along, like in Redis.
4. **Wide-column stores:** Used to store large data sets (store columns of data together). Example: Cassandra (Used in Facebook), HBase etc.

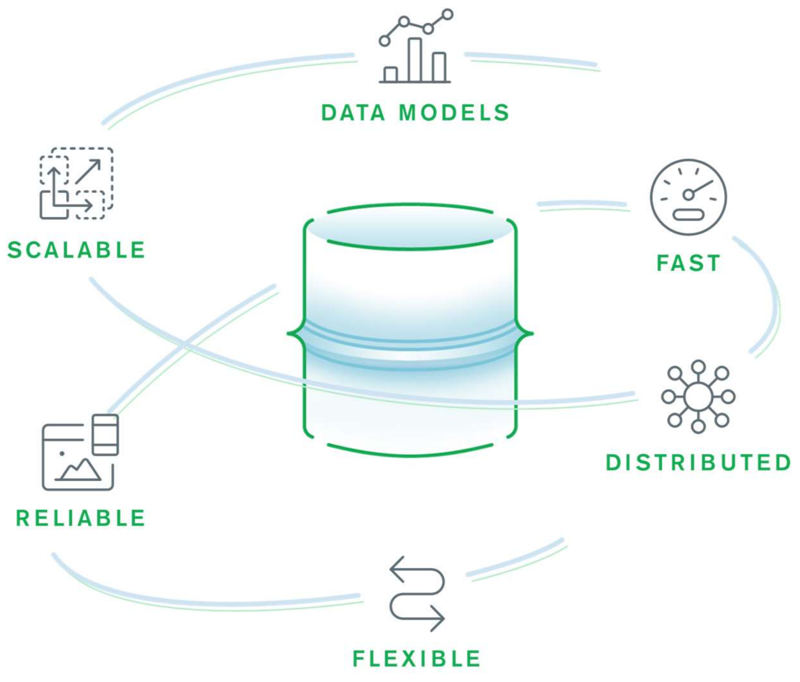


Figure 1 : The benefits of NOSQL Database

In this Experiment we would be using MONGODB which has a document-based structure

MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently. It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.

Now, we will see how actually thing happens behind the scene.

As we know that MongoDB is a database server and the data is stored in these databases. Or in other words, MongoDB environment gives you a server that you can start and then create multiple databases on it using MongoDB. Because of its NoSQL database, the data is stored in the collections and documents. Hence the database, collection, and documents are related to each other as shown below:

- The MongoDB database contains collections just like the MYSQL database contains tables. You are allowed to create multiple databases and multiple collections.
- Now inside of the collection we have documents. These documents contain the data we want to store in the MongoDB database and a single collection can contain multiple documents and you are schema-less meaning it is not necessary that one document is similar to another.
- MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time
- The documents are created using the fields. Fields are key-value pairs in the documents, it is just like columns in the relation database. The value of the fields can be of any BSON data types like double, string, Boolean, etc.
- The document model maps to the objects in your application code, making data easy to work with
- The data stored in the MongoDB is in the format of BSON documents. Here, BSON stands for Binary representation of JSON documents. Or in other words, in the backend, the MongoDB server converts the JSON data into a binary form that is known as BSON and this BSON is stored and queried more efficiently.
- In MongoDB documents, you are allowed to store nested data. This nesting of data allows you to create complex relations between data and store them in the same document which makes the working and fetching of data extremely efficient as compared to SQL.
- MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use

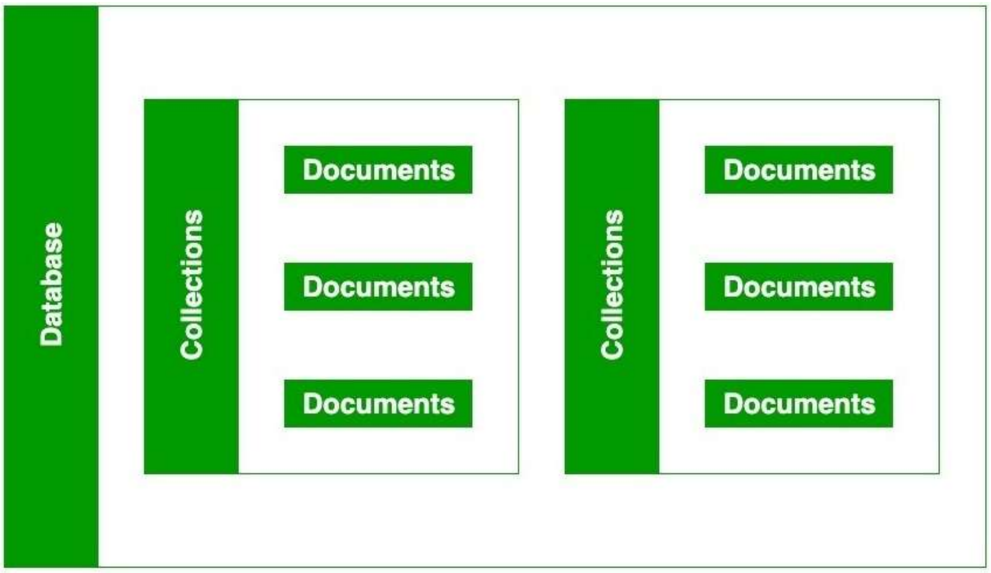


Figure 2 : Structure of MongoDB Database

****

Importing Database

****

The **mongoimport** tool imports content from an Extended JSON, CSV, or TSV

Importing Product Table:

```
mongoimport --db cs535 --collection product --type csv --headerline --file
D:\product.csv
```

Output:

```
2020-10-20T14:46:34.072+0530    connected to: mongodb://localhost/
2020-10-20T14:46:34.134+0530    2030 document(s) imported successfully. 0
document(s) failed to import.
```

Importing Customer Table:

```
mongoimport --db cs535 --collection customer --type csv --headerline --
file D:\customer.csv
```

Output:

```
2020-10-20T14:46:34.072+0530    connected to: mongodb://localhost/
```

```
2020-10-20T14:46:34.134+0530    17060 document(s) imported successfully. 0
document(s) failed to import.
```

QUERY SET

==========
****
QUERY-01: Find details about product id 83123A
****

```
> db.product.find(
                   {"_id":"85123A"}
             )
```

Output:

```
{
      "_id" : "85123A",
      "Description" : "WHITE HANGING HEART T-LIGHT HOLDER",
      "Quantity" : 504,
      "Price" : 2.55
}
```

This query selects documents in a collection (product) or view and returns a cursor to the selected documents , As **_id** is the unique element in the product table it returns only one document

<u>Note:</u> In MONGODB the tables are called collections.

****
QUERY-02: Find the schema of customer collection
****

```
> var schema = db.customer.findOne();
> for (var key in schema)
{
      print (key);
}
```

Output:

```
      _id
      product_id
      Date
      Quantity
      Unit Price
      Total Cost
      CustomerID
      Country
```

This query gives the details about the schema of a particular collection mentioned, this is very similar with SQL query **DESC TABLE_NAME;**

****
QUERY-03: Find details about product whose stock quantity is less than 2 units
****

```
> db.product.find(
                {Quantity :{$lt:2}}
          )
```

Output:

```
{
      "_id" : "21166",
      "Description" : "COOK WITH WINE METAL SIGN ",
      "Quantity" : 1,
      "Price" : 1.95
}
{
      "_id" : "84880",
      "Description" : "WHITE WIRE EGG HOLDER",
      "Quantity" : 1,
      "Price" : 4.95
}
{
      "_id" : "22915",
      "Description" : "ASSORTED BOTTLE TOP  MAGNETS ",
      "Quantity" : 0,
      "Price" : 0.42
```

```
}
{
      "_id" : "22968",
      "Description" : "ROSE COTTAGE KEEPSAKE BOX ",
      "Quantity" : 1,
      "Price" : 8.5
}
      .
      .
      .
      .
{
      "_id" : "84030E",
      "Description" : "ENGLISH ROSE HOT WATER BOTTLE",
      "Quantity" : 1,
      "Price" : 3.75
}
{
      "_id" : "21980",
      "Description" : "PACK OF 12 RED RETROSPOT TISSUES ",
      "Quantity" : 1,
      "Price" : 0.29
}
{
      "_id" : "22976",
      "Description" : "CIRCUS PARADE CHILDRENS EGG CUP ",
      "Quantity" : 1,
      "Price" : 2.51
}
{
      "_id" : "35957",
      "Description" : "SMALLFOLKART BAUBLE CHRISTMAS DEC",
      "Quantity" : 1,
      "Price" : 1.66
}
{
      "_id" : "85017B",
      "Description" : "ENVELOPE 50 BLOSSOM IMAGES",
      "Quantity" : 1,
      "Price" : 0.85
}
      .
      .
      .
      .
{
      "_id" : "84581",
      "Description" : "DOG TOY WITH PINK CROCHET SKIRT",
      "Quantity" : 1,
      "Price" : 4.21
}
{
      "_id" : "17003",
      "Description" : "BROCADE RING PURSE ",
      "Quantity" : 1,
      "Price" : 0.43
}
{
      "_id" : "20702",
      "Description" : "PINK PADDED MOBILE",
      "Quantity" : 1,
      "Price" : 4.25
}
{
      "_id" : "21693",
      "Description" : "SMALL HAMMERED SILVER CANDLEPOT ",
      "Quantity" : 1,
      "Price" : 2.95
}
```

This query uses the $lt operator which is a Comparison Query Operator to select documents where the value of the field is less than (i.e. <) the specified value.

```
*************************************************************************
QUERY-05: Find the details of the product sold on the invoice number
*************************************************************************
```

```
> db.customer.aggregate([
            {
                  $lookup :{
                        from :"product",
                        localField:"product_id",
                        foreignField:"_id",
                        as :"product_docs"
                  }
            },
            {
                  $project:{
                        product_id : 1,
                        Date : 1,
                        CustomerID:1,
                        product_docs:1
                  }
            }
])
```

Output:

```
{
      "_id" : "536365",
      "product_id" : "85123A",
      "Date" : ISODate("2011-01-11T18:30:00Z"),
      "CustomerID" : 17850,
      "product_docs" : [
            {
                  "_id" : "85123A",
                  "Description" : "WHITE HANGING HEART T-LIGHT HOLDER",
                  "Quantity" : 504,
                  "Price" : 2.55
            }
      ]
}
.
.
.
{
        "_id" : "536366",
        "product_id" : "71053",
        "Date" : ISODate("2011-01-11T18:30:00Z"),
        "CustomerID" : 17850,
        "product_docs" : [
                {
                        "_id" : "71053",
                        "Description" : "WHITE METAL LANTERN",
                        "Quantity" : 2375,
                        "Price" : 3.39
                }
        ]
}
.
.
.
{
        "_id" : "536377",
        "product_id" : "22749",
        "Date" : ISODate("2011-01-11T18:30:00Z"),
        "CustomerID" : 17850,
        "product_docs" : [
                {
                        "_id" : "22749",
                        "Description" : "FELTCRAFT PRINCESS CHARLOTTE DOLL",
                        "Quantity" : 3239,
                        "Price" : 3.75
                }
        ]
}
.
.
.
{
        "_id" : "536384",
        "product_id" : "21777",
        "Date" : ISODate("2011-01-11T18:30:00Z"),
        "CustomerID" : 18074,
        "product_docs" : [
            {
                  "_id" : "21777",
                  "Description" : "RECIPE BOX WITH METAL HEART",
                  "Quantity" : 18,
                  "Price" : 7.95
            }
      ]
}
```

This query contains a Aggregation pipeline, which is a framework for data aggregation modelled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results.

This query uses the **$lookup aggregation** which Performs a left outer join to a collection in the same database to filter in documents from the “joined” collection for processing. To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the “joined” collection.

The $lookup stage passes these reshaped documents to the next stage. And then in the next stage we used **$project aggregation** which acts like **SELECT** in SQL and projects only those which are being selected in the query, the specified fields can be existing fields from the input documents or newly computed fields.

****

QUERY-06: Find the total number of quantity sold in each month of the product “85123A”

****

```
> db.customer.aggregate ([
       {
            $match :{product_id :{$eq:"85123A"}}
       },
       {
            $group :{
                  _id:{
                        product_id:"$product_id",
                        pur_date:{
                              $dateToString :{
                                   format :"%m",
                                   date:"$Date"}
                              }
                  },
                  total:{$sum:"$Quantity"}
            }
       }
]);
```

Output:

```
{
      "_id" : { "product_id" : "85123A", "pur_date" : "11" }, "total" : 13
}
{
      "_id" : { "product_id" : "85123A", "pur_date" : "05" }, "total" : 12
}
{
      "_id" : { "product_id" : "85123A", "pur_date" : "06" }, "total" : 2
}
{
      "_id" : { "product_id" : "85123A", "pur_date" : "12" }, "total" : 12
}
{
      "_id" : { "product_id" : "85123A", "pur_date" : "01" }, "total" : 9
}
{
      "_id" : { "product_id" : "85123A", "pur_date" : "04" }, "total" : 1
}
```

This query also contains or is processed using an **Aggregation Pipeline**, where the first **$match (aggregation),** Filters the documents to pass only the documents that match the specified condition to the next pipeline stage. And **$eq** is the equals to operator to match the product_id.

The **$group (aggregation)** groups input documents by the specified _id expression and for each distinct grouping, outputs a document. The _id field of each output document contains the unique group by value. The output documents can also contain computed fields that hold the values of some accumulator expression.

The **$dateToString (aggregation)** Converts a date object to a string according to a user-specified format (here specified only month) i.e. (%m Month, 2 digits, zero padded)

The **$sum (aggregation) is an accumulator operator** calculates the sum of quantities grouped by the $group aggregation.

****

<u>Creating Application</u>

****

Inventory Management System:

This application is intended for the owners of Retail Shop where they need to manage their inventory, Many of the small or even large scale business always have an existing and long lasting problem managing their inventory and their sales, Its found that many retail shop owners have a need of continuous monitoring of their stocks and sales, and they cannot afford to take stock status or update manually as that’s a very tedious job and is not feasible every time.

To address these issues, we used NOSQL (Databases or Database Management System) MONGO DB which is a document based NOSQL database.

Here an Inventory Management Application is been designed using Python and its various libraries for making the application interactive and suitable for GUI based interaction collaborating with NOSQL MongoDB Database

We used libraries mainly, Tkinter, PyMongo, matplotlib

1) **Tkinter:** Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
   The GUI based application uses Tkinter to create and design the application’s GUI (Graphical User Interface) by creating an interactive frontend for the user interacting with the database

2) **PyMongo:** The PyMongo library allows interaction with the MongoDB database through Python; it is a native Python driver for MongoDB.
   It contains tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.
   The GUI based application uses PyMongo to establish connection between the database and the application that was created, and then executing queries and aggregation pipelines on the database through PyMongo.

3) **Matplotlib:** Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as Tkinter
   matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

We designed and implemented the application through these steps of implementation:

1) First of all, A dataset was selected from **data.world.com** which is a popularly known site for datasets, then after selecting the raw data set, A set of sequenced steps of data filtration ran through the dataset thoroughly to achieve filtered dataset. (in Subm_9B.pdf)

2) After proper cleaning and filtering the data set, an inbuilt function of mongo dB i.e. mongoimport was used to import the filtered dataset into collections namely product and customer into the database cs535

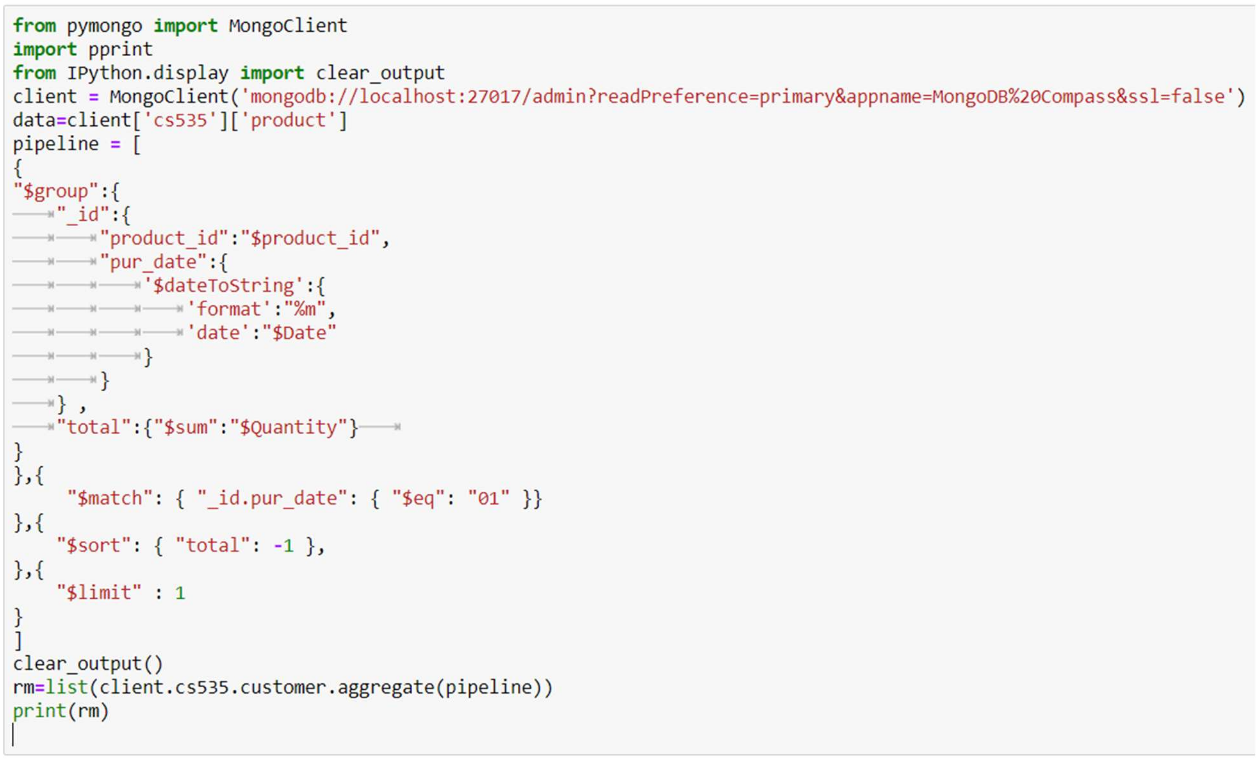
3) Now, starting to create the GUI Application using python (used platform: jupyter notebooks), The GUI was created by importing Tkinter by using which The GUI based application was designed consisting of labels, text boxes and buttons and aligning them properly and to trigger and implement the queries.

4) After then, from PyMongo library the MongoClient file was imported which establishes a connection between localhost database and python console or application
   Once the connection is successfully established then any sort of queries could be performed on our database.

5) Then, Our application was intended to show an embedded graph as in to see trends of the entered product_id , and for that database was queried upon triggering the button, then data was extracted from the database, which required projecting only the required terms and converted them to arrays and plotted the graph using matplotlib.pyplot and FigureCanvasTkAgg

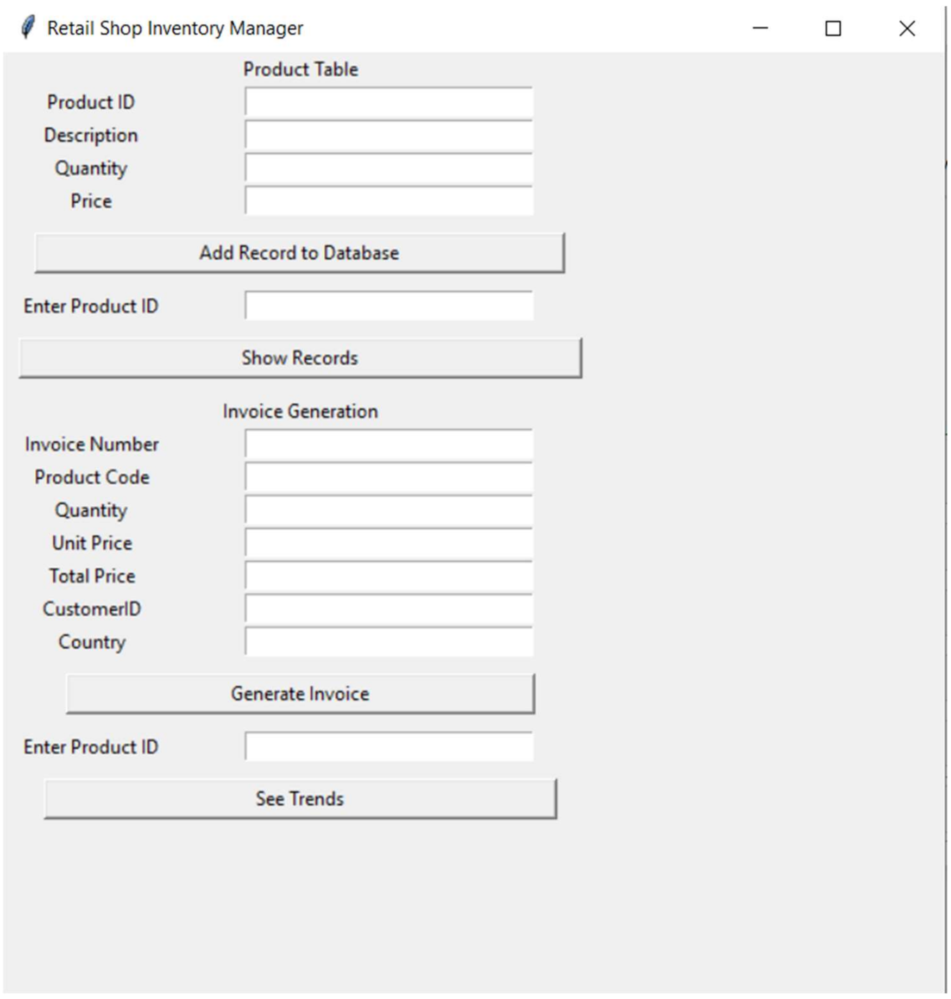
6) Hurray! now the application is perfectly ready for use.

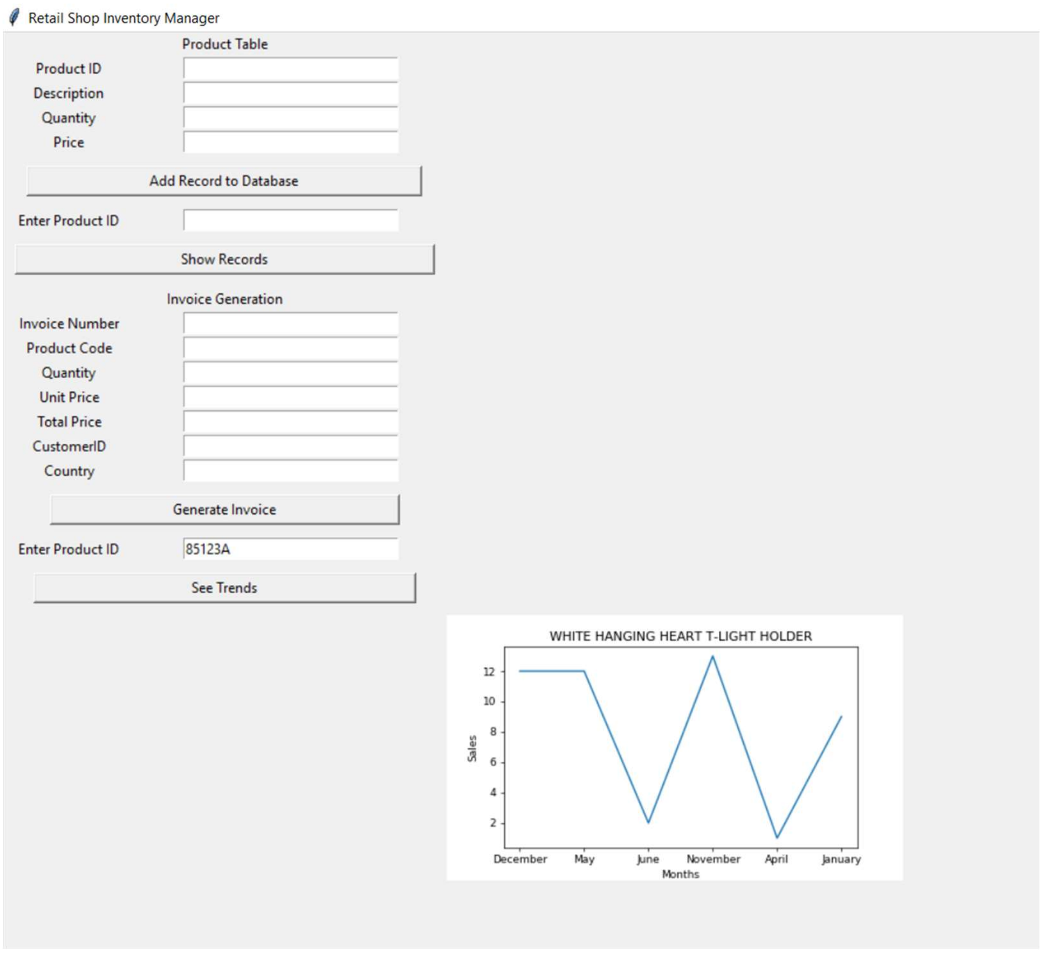
Here’s a sample query using Pymongo



```
from pymongo import MongoClient
import pprint
from IPython.display import clear_output
client = MongoClient('mongodb://localhost:27017/admin?readPreference=primary&appname=MongoDB%20Compass&ssl=false')
data=client['cs535']['product']
pipeline = [
{
"$group":{
	"_id":{
		"product_id":"$product_id",
		"pur_date":{
			'$dateToString':{
				'format':"%m",
				'date':"$Date"
			}
		}
	} ,
	"total":{"$sum":"$Quantity"}	
}
},{
     "$match": { "_id.pur_date": { "$eq": "01" }}
},{
    "$sort": { "total": -1 },
},{
    "$limit" : 1
}
]
clear_output()
rm=list(client.cs535.customer.aggregate(pipeline))
print(rm)
```

Few Screen shots of Application:




Retail Shop Inventory Manager

Product Table

Product ID	
Description	
Quantity	
Price	

Add Record to Database

Enter Product ID	

Show Records

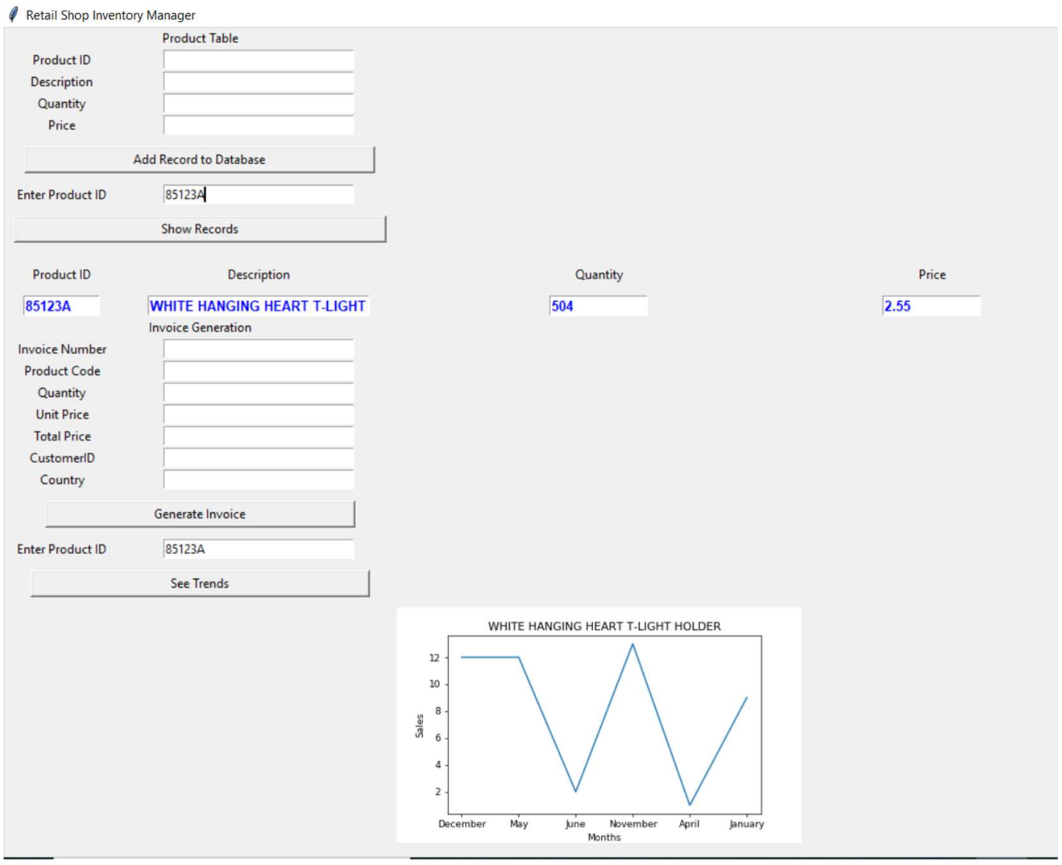
Invoice Generation

Invoice Number	
Product Code	
Quantity	
Unit Price	
Total Price	
CustomerID	
Country	

Generate Invoice

Enter Product ID	85123A

See Trends


Retail Shop Inventory Manager

Product Table

Product ID	
Description	
Quantity	
Price	

Add Record to Database

Enter Product ID	85123A

Show Records

Product ID	Description	Quantity	Price
85123A	WHITE HANGING HEART T-LIGHT	504	2.55

Invoice Generation

Invoice Number	
Product Code	
Quantity	
Unit Price	
Total Price	
CustomerID	
Country	

Generate Invoice

Enter Product ID	85123A

See Trends

****

INFERENCES

=========

1.We studied and implemented queries in MongoDB which is a document based NOSQL database

2. We also learnt how to create a successful connection between Python console and MongoDB using Pymongo

3. We also learnt how to design a GUI using Tkinter which is a Python library and designed an application and connected it to database

4. We also got to learn many more python libraries like matplotlib, FigureCanvasTkAgg and many more.

********************************** E N D **********************************