DATABASE MANAEMENT SYSTEM

Assignment – 4

LG STORE MANAGEMENT SYSTEM

REPORT

Achyut Jagini – PES2UG19CS013

Ajith Vivekanandan – PES2UG19CS021

Akif Iqbal – PES2UG19CS027

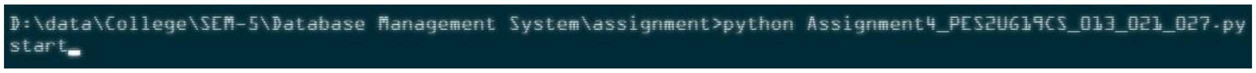
Dependency Installed:

```
import psycopg2
```

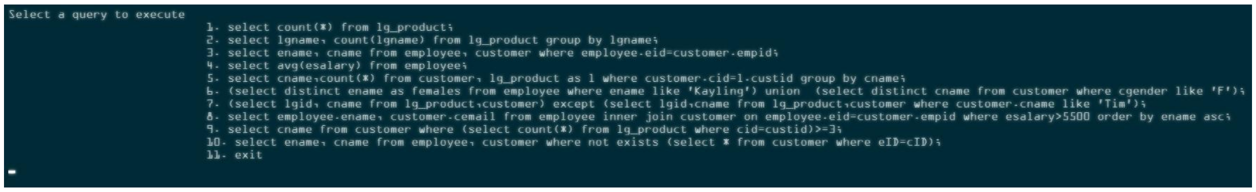
Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection). It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make many concurrent "INSERT" s or "UPDATE" s.

Its documentation can be found here.

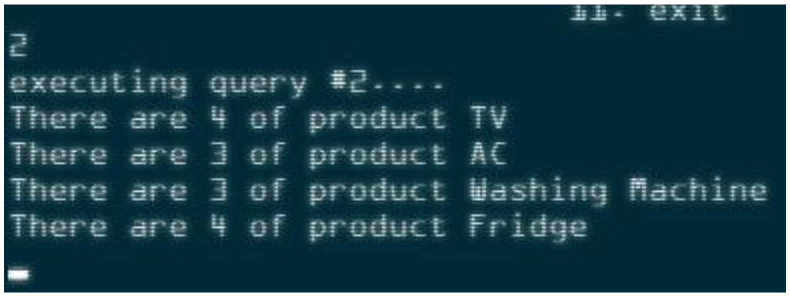
Statements executed from front-end (CLI):



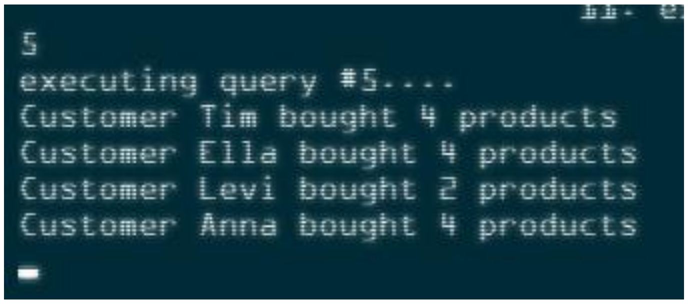
```
D:\data\College\SEM-5\Database Management System\assignment>python Assignment4_PES2UG19CS_013_021_027.py
start_
```



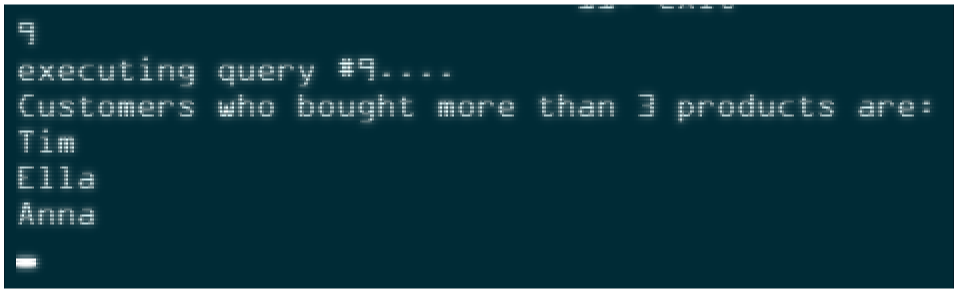
```
Select a query to execute
                        1. select count(*) from lg_product;
                        2. select lgname, count(lgname) from lg_product group by lgname;
                        3. select ename, cname from employee, customer where employee.eid=customer.empid;
                        4. select avg(esalary) from employee;
                        5. select cname,count(*) from customer, lg_product as l where customer.cid=l.custid group by cname;
                        6. (select distinct ename as females from employee where ename like 'Kayling') union  (select distinct cname from customer where cgender like 'F');
                        7. (select lgid, cname from lg_product,customer) except (select lgid,cname from lg_product,customer where customer.cname like 'Tim');
                        8. select employee.ename, customer.cemail from employee inner join customer on employee.eid=customer.empid where esalary>5500 order by ename asc;
                        9. select cname from customer where (select count(*) from lg_product where cid=custid)>=3;
                        10. select ename, cname from employee, customer where not exists (select * from customer where eID=cID);
                        11. exit
-
```



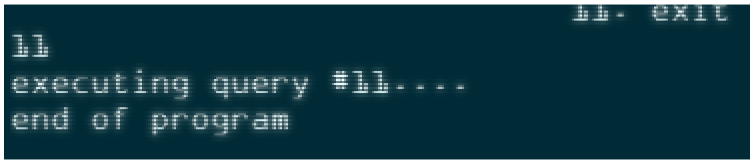
```
                                   11. exit
2
executing query #2....
There are 4 of product TV
There are 3 of product AC
There are 3 of product Washing Machine
There are 4 of product Fridge
-
```



```
5
executing query #5....
Customer Tim bought 4 products
Customer Ella bought 4 products
Customer Levi bought 2 products
Customer Anna bought 4 products
-
```



```
9
executing query #9....
Customers who bought more than 3 products are:
Tim
Ella
Anna
```



```
                                          11. exit
11
executing query #11....
end of program
```

<u>Schema change statements:</u>

```
alter table employee drop column ebank cascade;
drop table archive_employee cascade;
alter table customer add column caddress varchar(50);
alter table customer alter column cgender set default 'M';
drop table customer cascade;
```

<u>Changes in Business/Application changes/expansion - that might lead to:</u>

1. **<u>Schema changes-</u>** When there is a requirement to improve security, or performance in an organizational or business-related database. Schema could be altered to help them further improve on old schemas. Sometimes constant modifications of Schema could let new kinds of data be easily added to the database.
2. **<u>Constraint changes-</u>** Constraint changes would occur when there is a need for flexibility in the organization for adding data. Some data would go from being essential to obsolete during management which would require to remove some constraints on that data. Adding constraints could also be done when there may be some need to prevent altering the newly added data.
3. **<u>DBMS migration-</u>** When there may be a need to shift to a new application entirely for general reasons in the business or

organization there must be a basic idea of how to queries are made. Sometimes ties with other organizations would require shifting from one to another because of some changes. DBMS migration could be done only when there is a general idea of the entire database as both a schema and a relational table

Migration to a No-SQL variety: MongoDB

MongoDB has many advantages, including its native drivers, flexible schema, and query language, built-in autoscaling, and much more.

MongoDB vs. PostgreSQL

MongoDB is a scalable and flexible NoSQL document database. Rather than storing data in tables, MongoDB stores its data in collections of BSON (Binary JSON) documents. MongoDB can manage structured and unstructured data. That allows you to build your application without needing to first define the schema.

PostgreSQL is a relational database management system (RDBMS). In an RDBMS, data is stored in tables, and the schema of the database must be defined on creation. While it is possible to alter tables after their creation as an application's needs change, this process can be complicated and error prone.

Preparing to switch from PostgreSQL to MongoDB

While switching from PostgreSQL to MongoDB is not difficult, the process often involves more than just extracting and migrating data. You'll also need to examine the details of the applications that access your database. For example, if you're using an ORM that does not support both relational and document databases, you’ll need to find a new library that can connect to MongoDB.

MongoDB provides excellent documentation on SQL to MongoDB Mappings.

Once you’ve considered any changes needed to your application, the next step is to migrate the data. The migration for some of your tables might be simple.

However, you might want to restructure your data to fit better within a MongoDB schema design.

How to move data from PostgreSQL to MongoDB

The process for transferring data from PostgreSQL to MongoDB is clear-cut. Ultimately, the ease of your task depends on the complexity of the PostgreSQL database and the structure of document collections needed in the new MongoDB database.

To migrate data, you'll extract it from PostgreSQL and then import it to MongoDB using the mongoimport tool.

Conclusion

A modern, document-based database such as MongoDB can be a great choice over an RDBMS like PostgreSQL. Migrating from PostgreSQL to MongoDB can be a simple process.

Prepare your application for connecting to MongoDB., MongoDB has support for all the major programming languages as well as many popular frameworks.

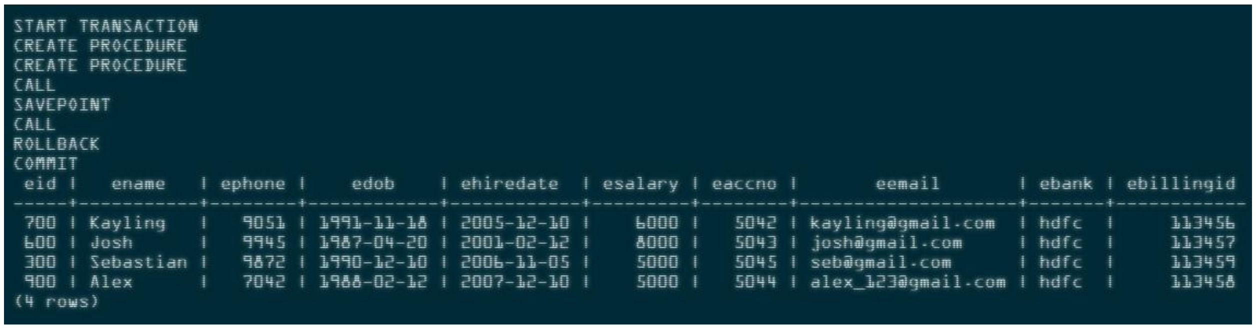
1. Consider the schema changes that would be best for your data, while keeping in mind MongoDB schema best practices and avoiding anti-patterns.
2. Export the data from your PostgreSQL databases by piping the result of an SQL query into a COPY command, outputting the result either as JSON or TSV.
3. Restructure the data to fit your MongoDB schema by using mongoimport.

Switching to MongoDB gives you benefits such as a more flexible schema and easier scalability. And while extracting and migrating a PostgreSQL database to MongoDB does take some thought and planning, the process itself is relatively pain-free.

Assignment-3 (remaining part)

```
two stored procedures with transactions

start transaction;
CREATE OR REPLACE PROCEDURE insert_sale()
LANGUAGE SQL
AS $$
insert into sale values (113910,912300);
$$;
CREATE OR REPLACE PROCEDURE update_emp_email(name varchar(255), mail
varchar(255))
LANGUAGE SQL
AS $$
update employee set eEmail = mail where eName like name;
$$;
call update_emp_email('Alex', 'alex_123@gmail.com');
savepoint s1;
call insert_sale();
rollback to s1;
end transaction;
select * from employee;
```



```
START TRANSACTION
CREATE PROCEDURE
CREATE PROCEDURE
CALL
SAVEPOINT
CALL
ROLLBACK
COMMIT
 eid |   ename   | ephone |    edob    | ehiredate  | esalary | eaccno |       eemail       | ebank | ebillingid
-----+-----------+--------+------------+------------+---------+--------+--------------------+-------+------------
 700 | Kayling   |   9051 | 1991-11-18 | 2005-12-10 |    6000 |   5042 | kayling@gmail.com  | hdfc  |     113456
 600 | Josh      |   9945 | 1987-04-20 | 2001-02-12 |    8000 |   5043 | josh@gmail.com     | hdfc  |     113457
 300 | Sebastian |   9872 | 1990-12-10 | 2006-11-05 |    5000 |   5045 | seb@gmail.com      | hdfc  |     113459
 900 | Alex      |   7042 | 1988-02-12 | 2007-12-10 |    5000 |   5044 | alex_123@gmail.com | hdfc  |     113458
(4 rows)
```

Contributions:

Achyut Jagini – 2 procedures, write up about the database migration to No-SQL variety database – 1hr

Ajith Vivekanandan - Changes in constraints sand schema, Apply the changes to the existing database schema, 5 statement examples for the same – 15mins

Akif Iqbal – Added transactions to 2 procedures, Simple User Interface using Python and CLI, paragraph stating the dependencies installed for database connectivity, compilation of report – 2hrs