

Experiment No.4

Title: Execution of object relational queries

Batch: SY-IT(B3)

Roll No.: 16010423076

Experiment No.:4

Title: Execution of object relational queries**Resources needed:** PostgreSQL 9.3

Theory

Object types are user-defined types that make it possible to model real-world entities such as customers and purchase orders as objects in the database.

New object types can be created from any built-in database types and any previously created object types, object references, and collection types. Metadata for user-defined types is stored in a schema that is available to SQL, PL/SQL, Java, and other published interfaces.

Row Objects and Column Objects:

Objects that are stored in complete rows in object tables are called row objects.

Objects that are stored as columns of a table in a larger row, or are attributes of other objects, are called column objects

Defining Types:

In PostgreSQL the syntax for creating simple type is as follows,

```
CREATE TYPE name AS  
    ( attribute_name data_type [, ... ] );
```

Example:

A definition of a point type consisting of two numbers in PostgreSQL is as follows,

```
create type PointType as(  
    x int,  
    y int  
);
```

An object type can be used like any other type in further declarations of object-types or table-types.

E.g. a new type with name LineType is created using PointType which is created earlier.

```
CREATE TYPE LineType AS (
    end1 PointType,
    end2 PointType
);
```

Dropping Types :

To drop type for example LineType, command will be :

```
DROP TYPE Linetype;
```

Constructing Object Values:

Like C++, PostgreSQL provides built-in constructors for values of a declared type, and these constructors can be invoked using a parenthesized list of appropriate values.

For example, here is how we would insert into Lines a line with ID 27 that ran from the origin to the point (3,4):

```
INSERT INTO Lines VALUES (27, ((0,0) , (3,4)) , distance(0,0,3,4)) ;
```

Declaring and Defining Methods:

A type declaration can also include methods that are defined on values of that type. The method is declared as shown in example below.

```
CREATE OR REPLACE FUNCTION distance(x1 integer, y1 integer,x2
integer,y2 integer) RETURNS float AS $$
BEGIN
    RETURN sqrt(power((x2-x1),2)+power((y2-y1),2));
END;
$$ LANGUAGE plpgsql;
```

Then you can create tables using these object types and basic datatypes.

Creation on new table Lines is shown below.

```
CREATE TABLE Lines (
    lineID INT,
    line LineType,
    dist float
);
```

Now after the table is created you can add populate table by executing insert queries as explained above.

You can execute different queries on Lines table. For example to display data of Lines table, select specific line from Lines table etc.

Queries to Relations That Involve User-Defined Types:

Values of components of an object are accessed with the dot notation. We actually saw an example of this notation above, as we found the x-component of point end1 by referring to end1.x, and so on. In general, if N refers to some object O of type T , and one of the components (attribute or method) of type T is A , then $N.A$ refers to this component of object O .

For example, the following query finds the x co-ordinates of both endpoints of line.

```
SELECT lineID, ((L.line).end1).x, ((L.line).end2).x
FROM Lines L;
```

- Note that in order to access fields of an object, we have to start with an *alias* of a relation name. While lineID, being a top-level attribute of relation Lines, can be referred to normally, in order to get into the attribute line, we need to give relation Lines an alias (we chose L) and use it to start all paths to the desired subobjects.
- Dropping the ``L" or replacing it by ``Lines." doesn't work.
- Notice also the use of a method in a query. Since line is an attribute of type LineType, one can apply to it the methods of that type, using the dot notation shown.

Here are some other queries about the relation lines.

```
SELECT (L.line).end2 FROM Lines L;
```

Prints the second end of each line, but as a value of type PointType, not as a pair of numbers.

Object Oriented features:

Inheritance:

```
CREATE TABLE point of PointType;

CREATE TABLE axis (
    z int
) inherits (point);

INSERT INTO axis values(2,5,6);

select * from axis;
```

Procedure / Approach /Algorithm / Activity Diagram:

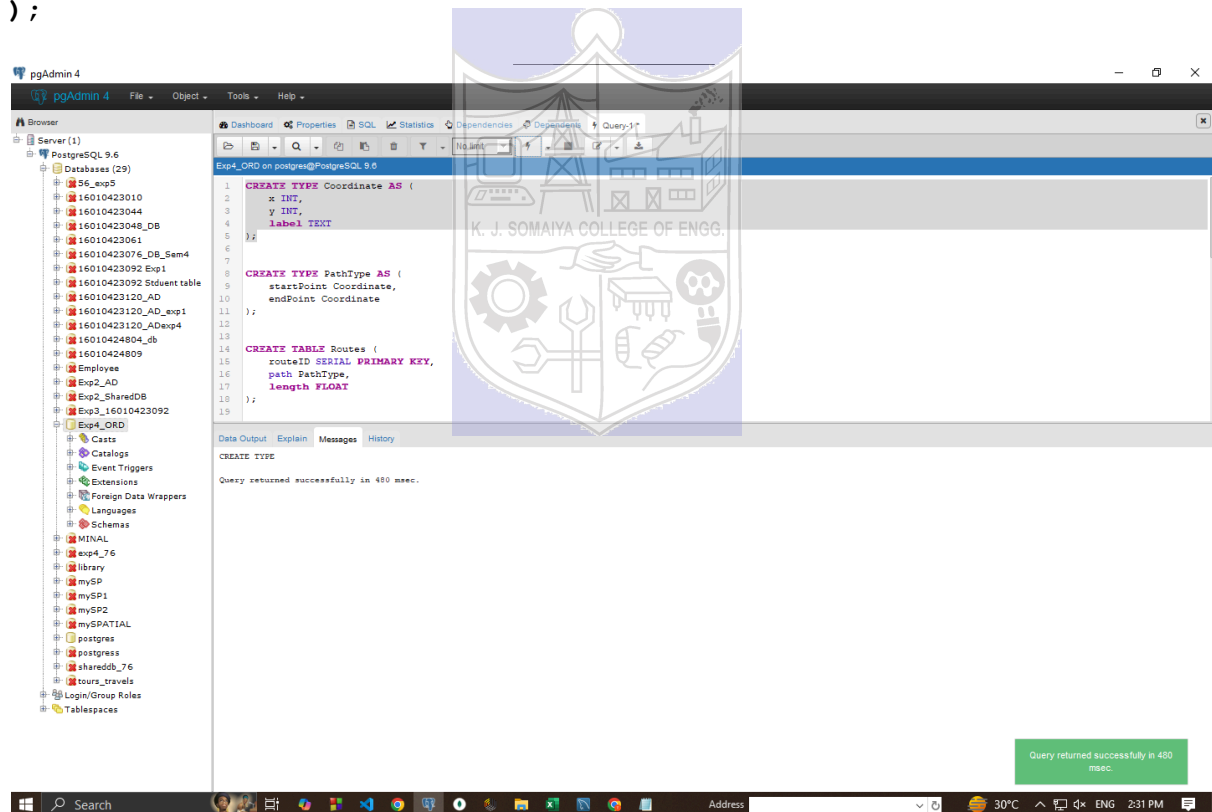
Perform following tasks,

- Create a table using object type field
 - Insert values in that table
 - Retrieve values from the table
 - Implement and use any function associated with the table created
-

Results: (Queries depicting the above said activity performed individually)

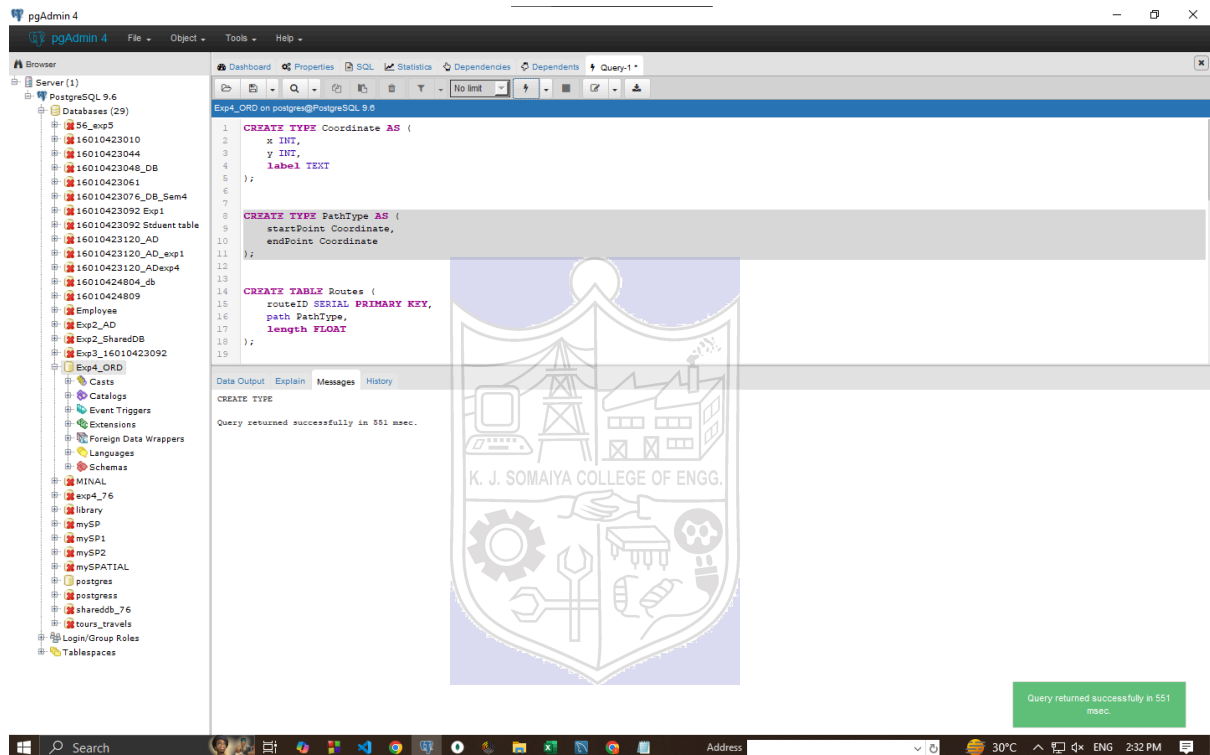
-- Create a type 'Coordinate' to store x, y values and a label

```
CREATE TYPE Coordinate AS (
    x INT,
    y INT,
    label TEXT
);
```



-- Create a type 'PathType' that consists of two coordinates (start and end points)

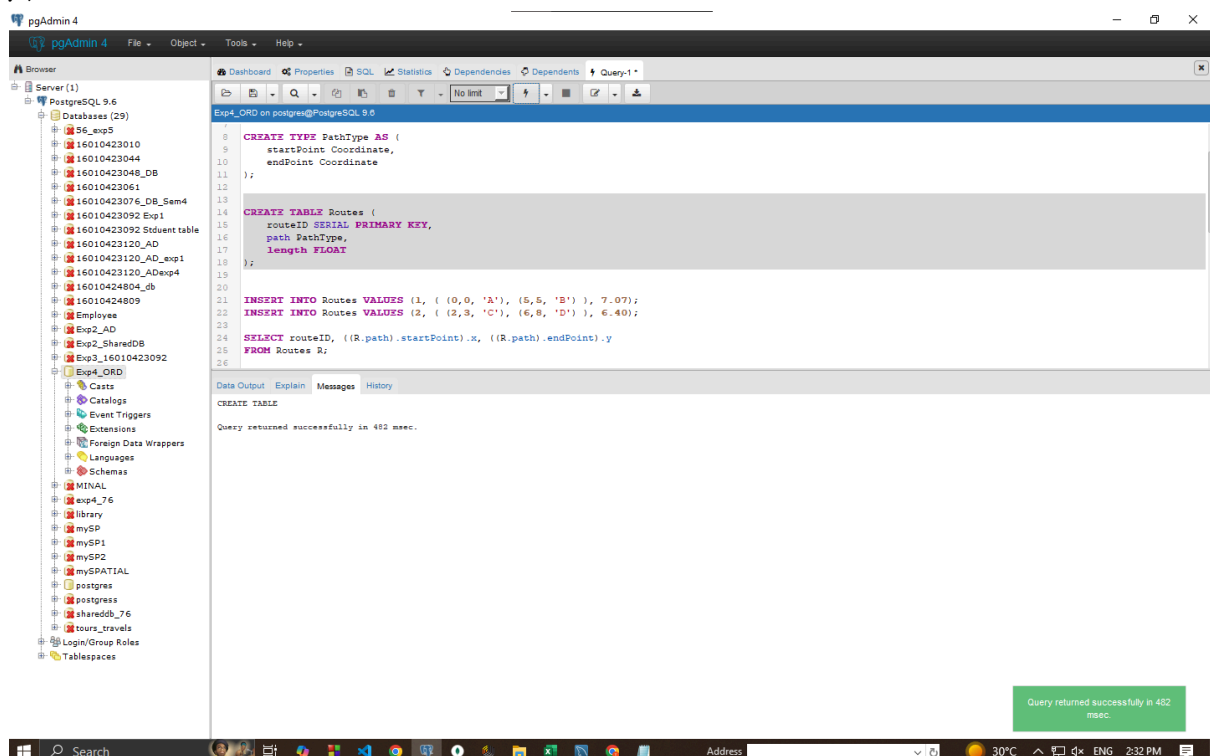
```
CREATE TYPE PathType AS (
    startPoint Coordinate,
    endPoint Coordinate
);
```



```

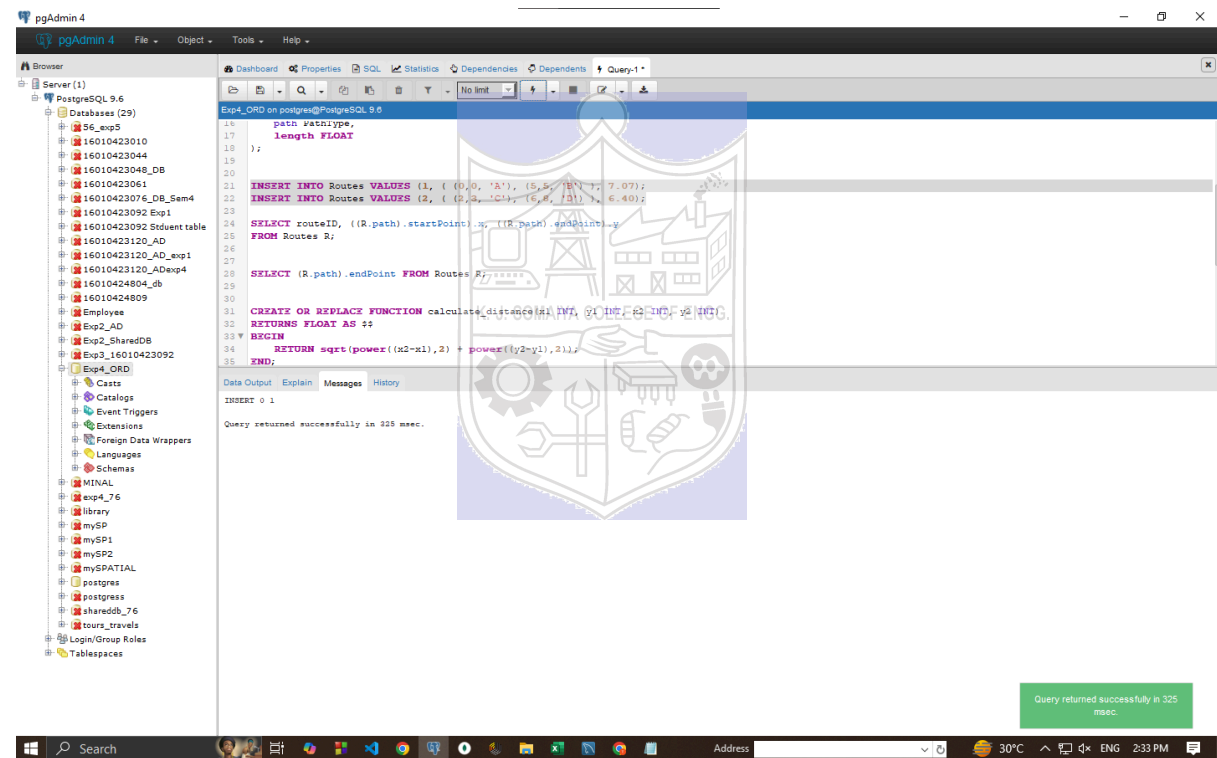
-- Create a table 'Routes' with a unique routeID, a path of type
PathType, and its length
CREATE TABLE Routes (
    routeID SERIAL PRIMARY KEY,
    path PathType,
    length FLOAT
);

```



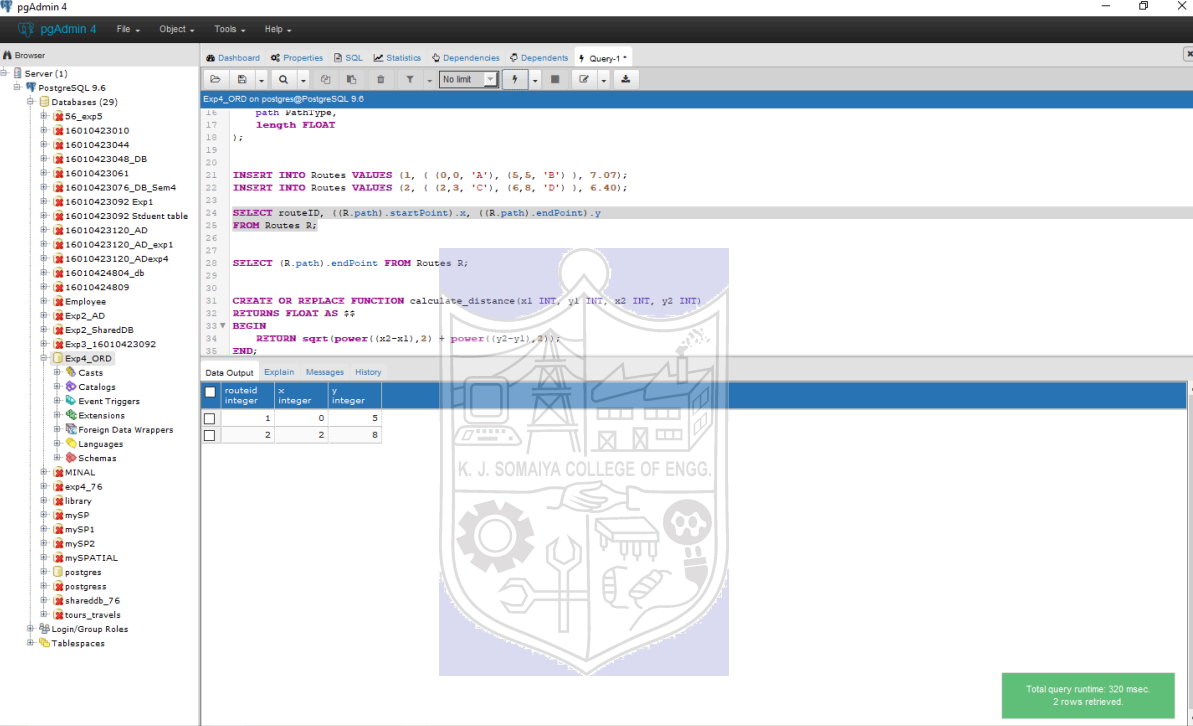
```
-- Insert some routes with start and end coordinates and predefined distances
```

```
INSERT INTO Routes VALUES (1, ( (0,0, 'A'), (5,5, 'B') ), 7.07);
INSERT INTO Routes VALUES (2, ( (2,3, 'C'), (6,8, 'D') ), 6.40);
```



```
-- Retrieve the routeID, x-coordinate of the start point, and y-coordinate of the end point
```

```
SELECT routeID, ((R.path).startPoint).x, ((R.path).endPoint).y
FROM Routes R;
```



pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (29)

- 36_exp5
- 16010423010
- 16010423044
- 16010423048_DB
- 16010423061
- 16010423076_DB_Sem4
- 16010423092_Exp1
- 16010423092_Student table
- 16010423120_AD
- 16010423120_AD_exp1
- 16010423120_ADexp4
- 16010424804_db
- 16010424809
- Employee
- Exp2_AD
- Exp2_SharedDB
- Exp3_16010423092
- Exp4_ORD
- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas
- MINAL
- exp4_76
- library
- mySP
- mySP1
- mySP2
- mySPATIAL
- postgres
- postgres
- shareddb_76
- tours_travels
- Login/Group Roles
- Tablespaces

Query 1

```

11 path varchar,
12 length FLOAT
13 );
14
21 INSERT INTO Routes VALUES (1, ( (0,0, 'A'), (5,5, 'B') ), 7.07);
22 INSERT INTO Routes VALUES (2, ( (2,3, 'C'), (6,8, 'D') ), 6.40);
23
24 SELECT routeID, ((R.path).startPoint).x, ((R.path).endPoint).y
25 FROM Routes R;
26
27 SELECT (R.path).endPoint FROM Routes R;
28
29
30 CREATE OR REPLACE FUNCTION calculate_distance(x1 INT, y1 INT, x2 INT, y2 INT)
31 RETURNS FLOAT AS $$
32 BEGIN
33 RETURN sqrt(power((x2-x1),2) + power((y2-y1),2));
34 END;
35

```

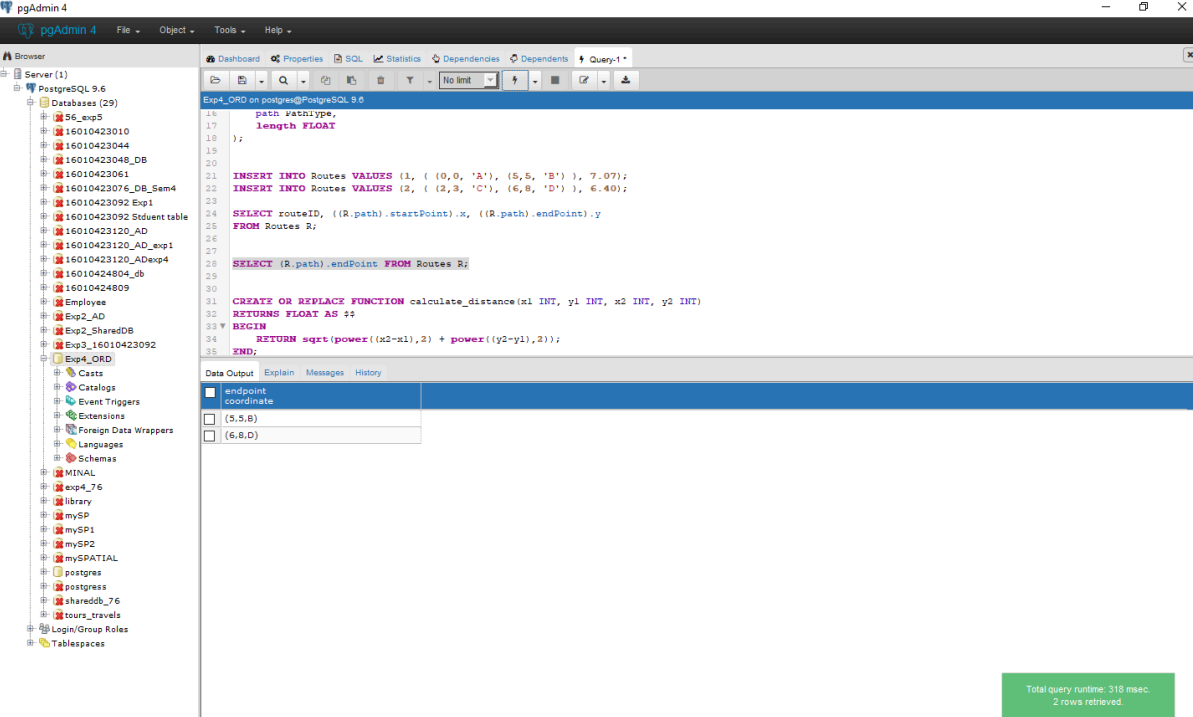
Data Output

routeid	x	y
1	0	5
2	2	8

Total query runtime: 320 msec.
2 rows retrieved.

-- Retrieve the endPoint column (which is of type Coordinate) from all routes

SELECT (R.path).endPoint FROM Routes R;



pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (29)

- 36_exp5
- 16010423010
- 16010423044
- 16010423048_DB
- 16010423061
- 16010423076_DB_Sem4
- 16010423092_Exp1
- 16010423092_Student table
- 16010423120_AD
- 16010423120_AD_exp1
- 16010423120_ADexp4
- 16010424804_db
- 16010424809
- Employee
- Exp2_AD
- Exp2_SharedDB
- Exp3_16010423092
- Exp4_ORD
- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas
- MINAL
- exp4_76
- library
- mySP
- mySP1
- mySP2
- mySPATIAL
- postgres
- postgres
- shareddb_76
- tours_travels
- Login/Group Roles
- Tablespaces

Query 1

```

11 path varchar,
12 length FLOAT
13 );
14
21 INSERT INTO Routes VALUES (1, ( (0,0, 'A'), (5,5, 'B') ), 7.07);
22 INSERT INTO Routes VALUES (2, ( (2,3, 'C'), (6,8, 'D') ), 6.40);
23
24 SELECT routeID, ((R.path).startPoint).x, ((R.path).endPoint).y
25 FROM Routes R;
26
27 SELECT (R.path).endPoint FROM Routes R;
28
29
30 CREATE OR REPLACE FUNCTION calculate_distance(x1 INT, y1 INT, x2 INT, y2 INT)
31 RETURNS FLOAT AS $$
32 BEGIN
33 RETURN sqrt(power((x2-x1),2) + power((y2-y1),2));
34 END;
35

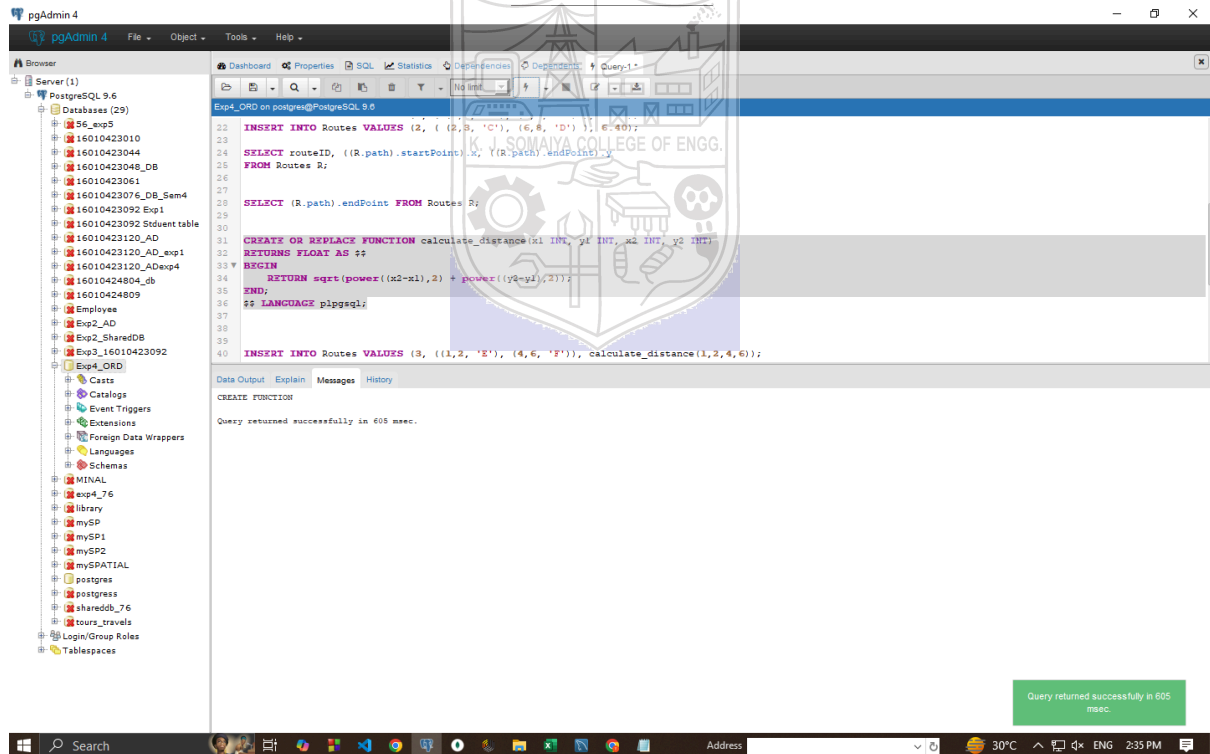
```

Data Output

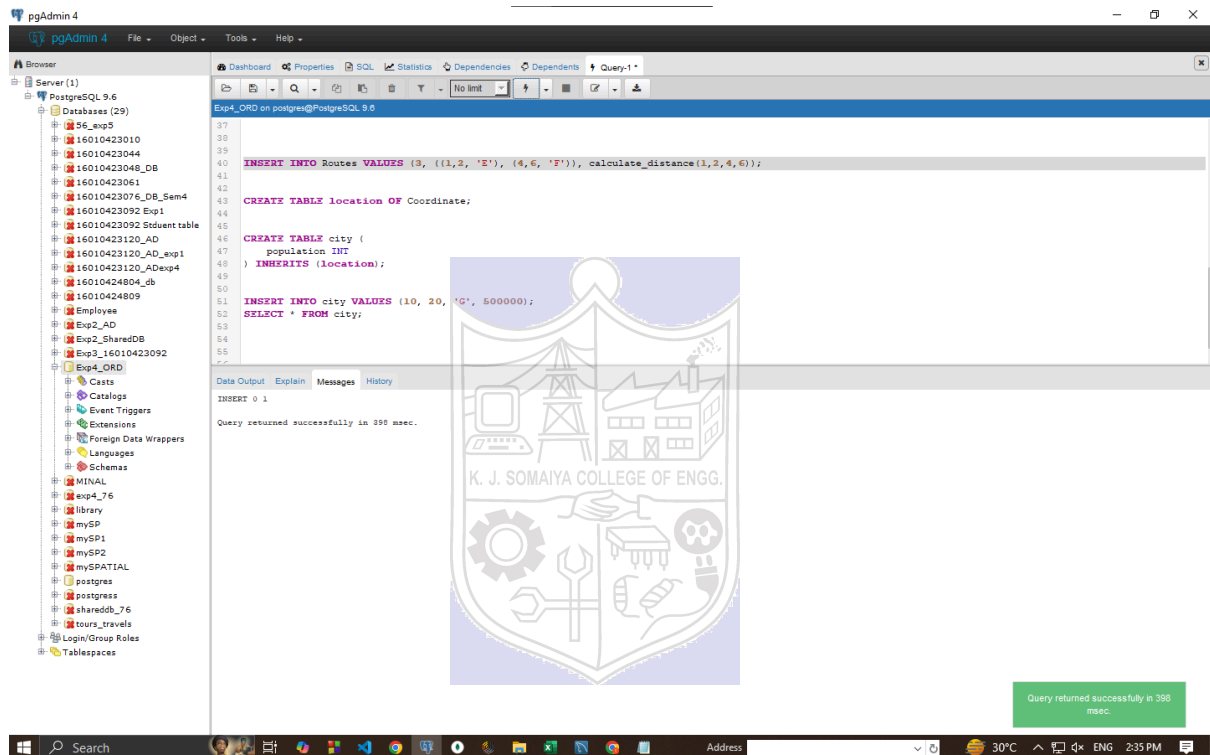
endPoint
(5,5,B)
(6,8,D)

Total query runtime: 318 msec.
2 rows retrieved.


```
-- Create a function to calculate the Euclidean distance between two
points
CREATE OR REPLACE FUNCTION calculate_distance(x1 INT, y1 INT, x2
INT, y2 INT)
RETURNS FLOAT AS $$
BEGIN
    RETURN sqrt(power((x2-x1),2) + power((y2-y1),2));
END;
$$ LANGUAGE plpgsql;
```



```
-- Insert a new route using the calculate_distance function
INSERT INTO Routes VALUES (3, ((1,2, 'E'), (4,6, 'F')),
calculate_distance(1,2,4,6));
```



pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (29)

- 36_exp5
- 16010423010
- 16010423044
- 16010423048_DB
- 16010423061
- 16010423076_DB_Sem4
- 16010423092_Exp1
- 16010423092_Student table
- 16010423120_AD
- 16010423120_AD_exp1
- 16010423120_ADexp4
- 16010424804_db
- 16010424809
- Employee
- Exp2_AD
- Exp2_SharedDB
- Exp3_16010423092
- Exp4_ORD
- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas
- MINAL
- exp4_76
- library
- mySP
- mySP1
- mySP2
- mySPATIAL
- postgres
- postgres
- shareddb_76
- tours_travels
- Login/Group Roles
- Tablespaces

Query 1

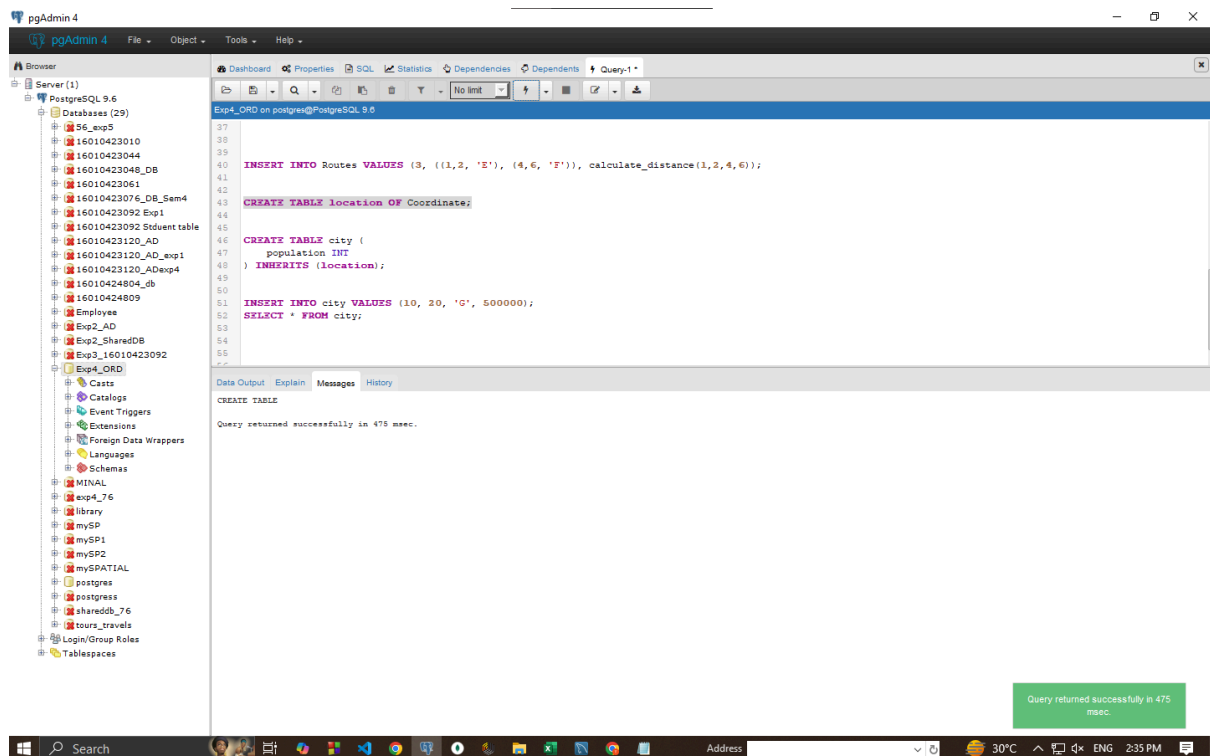
```

INSERT INTO Routes VALUES (3, ((1,2, 'E'), (4,6, 'F')), calculate_distance(1,2,4,6));

```

Query returned successfully in 398 msec.

-- Create a table 'location' using the Coordinate type
CREATE TABLE location OF Coordinate;



pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (29)

- 36_exp5
- 16010423010
- 16010423044
- 16010423048_DB
- 16010423061
- 16010423076_DB_Sem4
- 16010423092_Exp1
- 16010423092_Student table
- 16010423120_AD
- 16010423120_AD_exp1
- 16010423120_ADexp4
- 16010424804_db
- 16010424809
- Employee
- Exp2_AD
- Exp2_SharedDB
- Exp3_16010423092
- Exp4_ORD
- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas
- MINAL
- exp4_76
- library
- mySP
- mySP1
- mySP2
- mySPATIAL
- postgres
- postgres
- shareddb_76
- tours_travels
- Login/Group Roles
- Tablespaces

Query 1

```

CREATE TABLE location OF Coordinate;

```

Query returned successfully in 475 msec.

-- Create a table 'city' that inherits from 'location' and adds a population column
CREATE TABLE city (

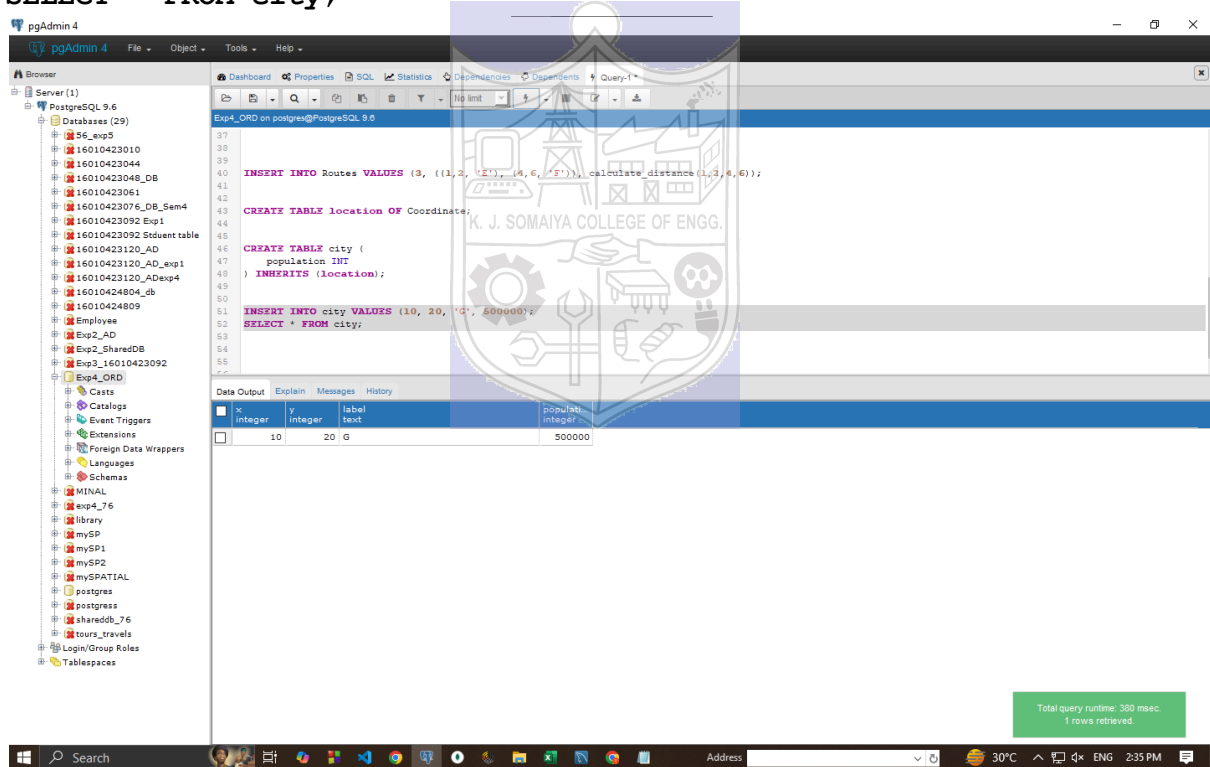
```

    population INT
) INHERITS (location);

-- Insert data into 'city' (includes x, y, label, and population)
INSERT INTO city VALUES (10, 20, 'G', 500000);

-- Retrieve all records from the 'city' table
SELECT * FROM city;

```



Questions:

1. What is the difference between object relational and object oriented databases?

Feature	Object-Relational Database	Object-Oriented Database
Structure	Extends relational model with object features	Uses a fully object-oriented approach
Querying	SQL-based with support for objects	Object query languages (OQL)
Data Representation	Uses tables but allows complex data types	Stores data as objects without tables

Best Use Case	Suitable for applications needing structured and object data	Works well with applications relying on object relationships
----------------------	--	--

2. Give comparison of any two database systems providing object relational database features.

Feature	PostgreSQL	Oracle
Object Features	Offers table inheritance, user-defined types (UDTs), and functions	Supports object types, collections, and XML-based storage
Performance	Excellent for handling complex relationships	Highly optimized for large-scale transactions
Customization	Allows users to define custom operators and indexing	Provides robust object-relational support but with more restrictions
Common Usage	Research, startups, and scalable web applications	Large enterprises requiring complex data structures

3. Explore how the user defined types can be modified with queries.

Altering a **User-Defined Type (UDT)** in PostgreSQL:

Adding a new field:

```
ALTER TYPE employee ADD ATTRIBUTE salary NUMERIC;
```

Changing a field name:

```
ALTER TYPE employee RENAME ATTRIBUTE salary TO income;
```

Removing an attribute:

```
ALTER TYPE employee DROP ATTRIBUTE income;
```

Outcomes:

CO1 : Design advanced database systems using Parallel, Distributed, Object-Relational Databases and its implementation.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

From this experiment, I learned how to create and use object-relational databases in PostgreSQL. I understood how to define custom data types, use them in tables, and perform queries on nested objects using dot notation. I also explored how to write functions to calculate values dynamically and how table inheritance works. This experiment helped me see how object-oriented concepts can be applied in databases to store and manage complex data efficiently.

Grade: AA / AB / BB / BC / CC / CD / DD

Signature of faculty in-charge with date

References:

1. Elmasri and Navathe, "Fundamentals of Database Systems", Pearson Education
2. Raghu Ramakrishnan and Johannes Gehrke, "Database Management Systems" 3rd Edition, McGraw Hill, 2002
3. Korth, Silberchatz, Sudarshan, "Database System Concepts" McGraw Hill