# Experiment No.3

## Title: Execution of object relational queries

**Batch:** **B1** **Roll No.:** **16010420133** **Experiment No.:3**

**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 userdefined 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

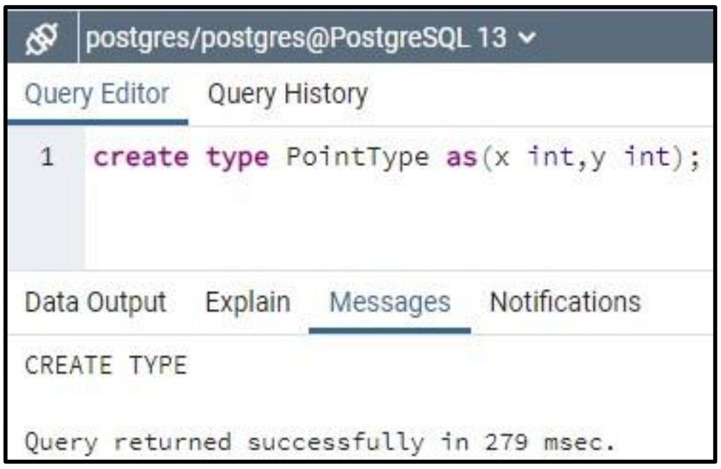
**Results: (Queries depicting the above said activity performed individually)** USE COURIER NEW FONT WITH SIZE =11 FOR QUERY STATEMENTS

Here, we created type as PointType:

**Query:**

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

**Output:**



```
postgres/postgres@PostgreSQL 13
Query Editor   Query History
create type PointType as(x int,y int);
Data Output   Explain   Messages   Notifications
CREATE TYPE

Query returned successfully in 279 msec.
```

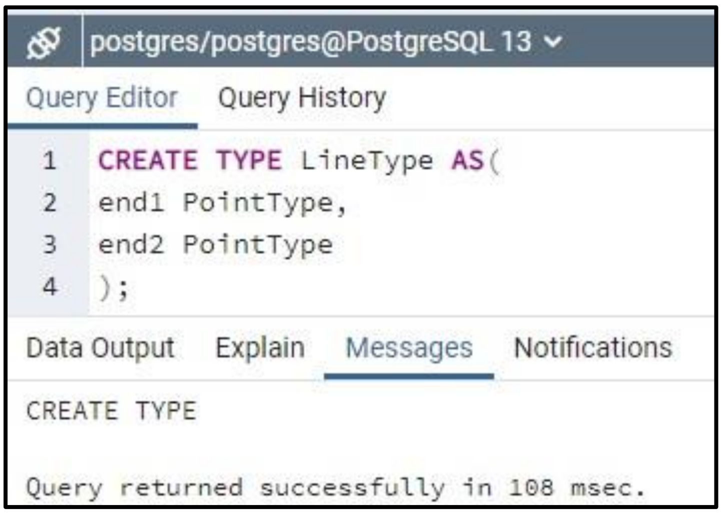
Here, we created LineType with object type as PointType:

**Query:**

*CREATE TYPE LineType AS( end1 PointType, end2 PointType*

*);*

**Output:**



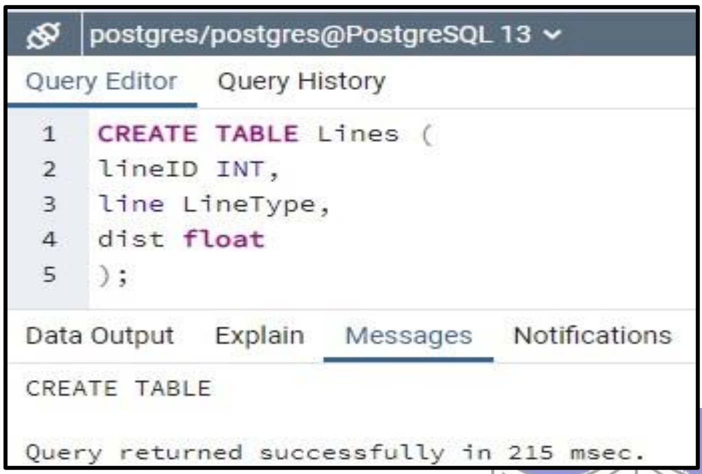
```
postgres/postgres@PostgreSQL 13
Query Editor   Query History
CREATE TYPE LineType AS(
end1 PointType,
end2 PointType
);
Data Output   Explain   Messages   Notifications
CREATE TYPE

Query returned successfully in 108 msec.
```

Here, we created a Table with attributes as ID, Linetype And Distance:

**Query:**

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

**Output:**



postgres/postgres@PostgreSQL 13

Query Editor   Query History

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

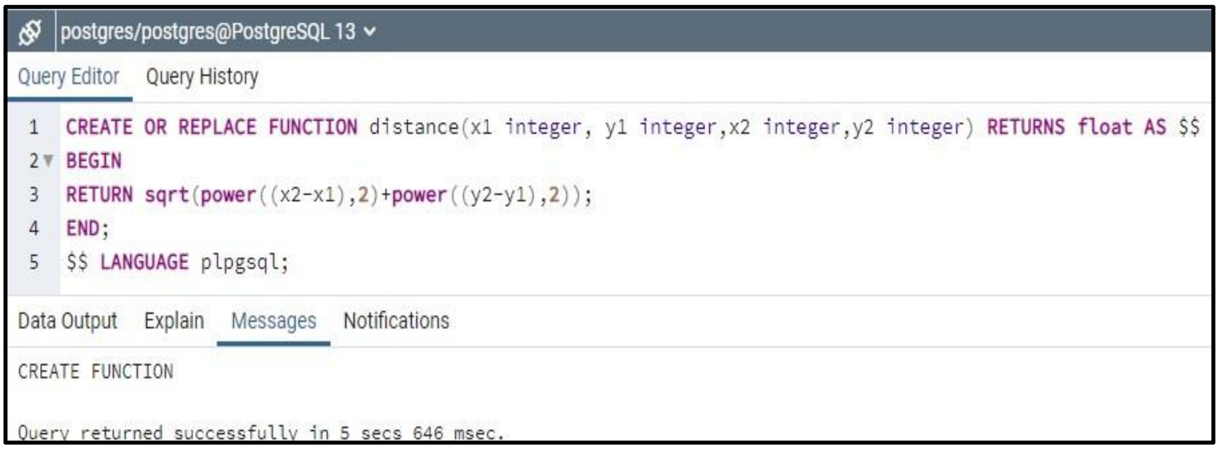
Data Output   Explain   Messages   Notifications

```
CREATE TABLE

Query returned successfully in 215 msec.
```

Here, we created a function 'Distance' and calculated the distance using the defined function:
**Query:**

```
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;
```

**Output:**



postgres/postgres@PostgreSQL 13

Query Editor   Query History

```
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;
```

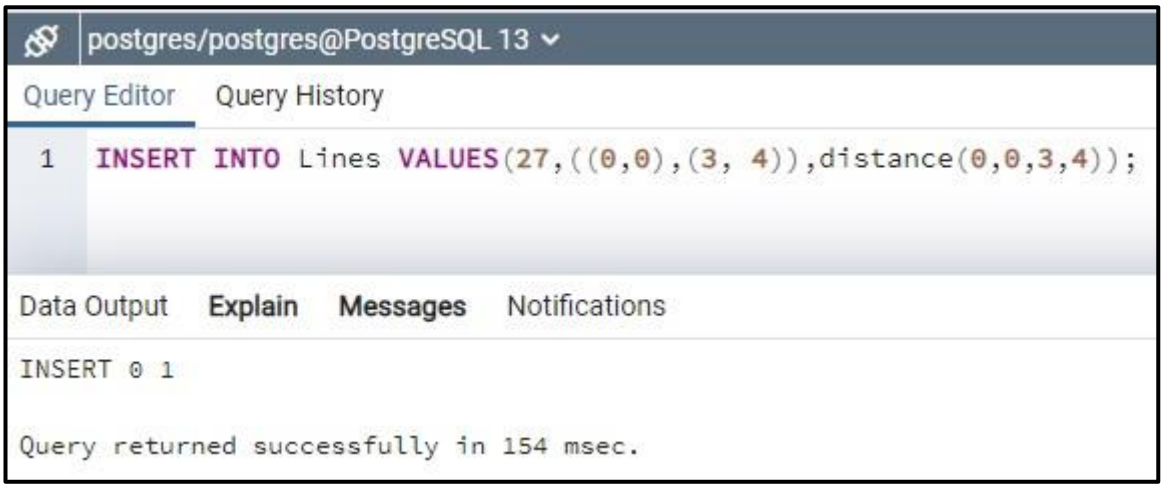
Data Output   Explain   Messages   Notifications

```
CREATE FUNCTION

Query returned successfully in 5 secs 646 msec.
```

Here, we inserted the values into Line Table:
**Query:**

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

**Output:**



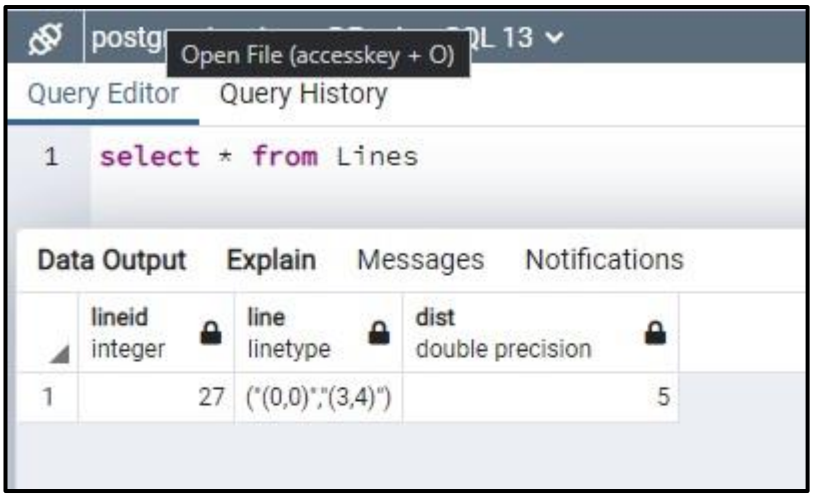
```
INSERT INTO Lines VALUES(27,((0,0),(3, 4)),distance(0,0,3,4));
```

```
INSERT 0 1

Query returned successfully in 154 msec.
```

Here, we displayed the line Table:
**Query:**

*select * from Lines*

**Output:**



```
select * from Lines
```

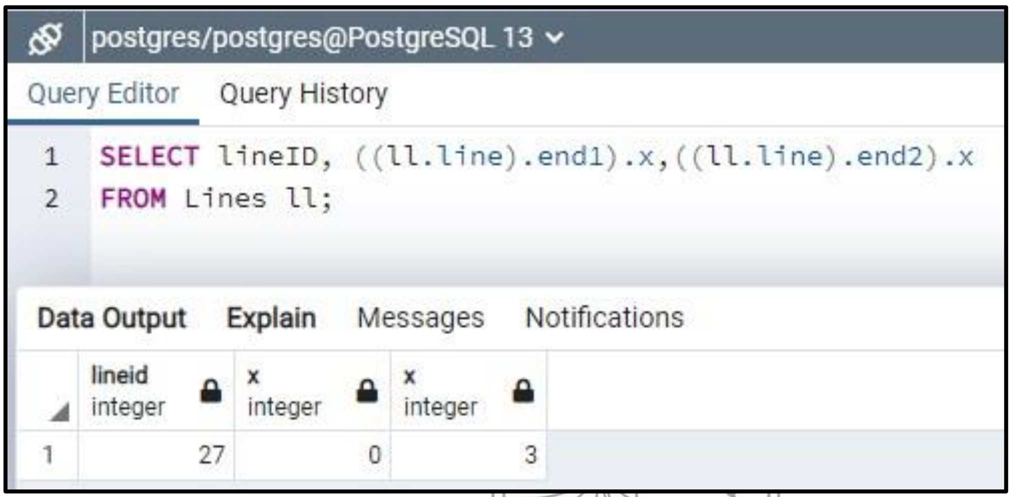
| | lineid integer | line linetype | dist double precision |
|---|---|---|---|
| 1 | 27 | ("(0,0)","(3,4)") | 5 |

Here, we displayed the x integer of both the points:

**Query:**

*SELECT lineID,*
*((ll.line).end1).x,((ll.line).end2).x*
*FROM Lines ll;*

**Output:**



```
postgres/postgres@PostgreSQL 13
Query Editor   Query History
SELECT lineID, ((ll.line).end1).x,((ll.line).end2).x
FROM Lines ll;
Data Output   Explain   Messages   Notifications
```

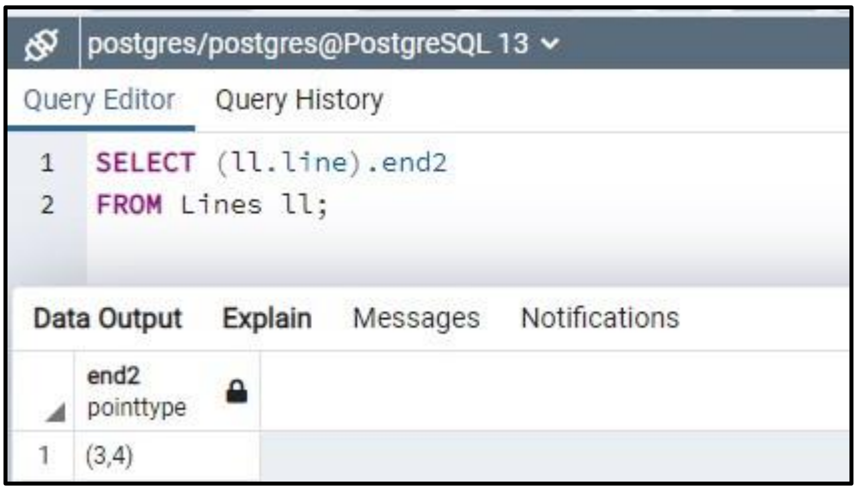
| | lineid integer | x integer | x integer |
|---|---|---|---|
| 1 | 27 | 0 | 3 |

Here, we displayed the point:
**Query:**

*SELECT (ll.line).end2*

*FROM Lines ll;*

**Output:**



```
postgres/postgres@PostgreSQL 13
Query Editor   Query History
SELECT (ll.line).end2
FROM Lines ll;
Data Output   Explain   Messages   Notifications
```

| | end2 pointtype |
|---|---|
| 1 | (3,4) |

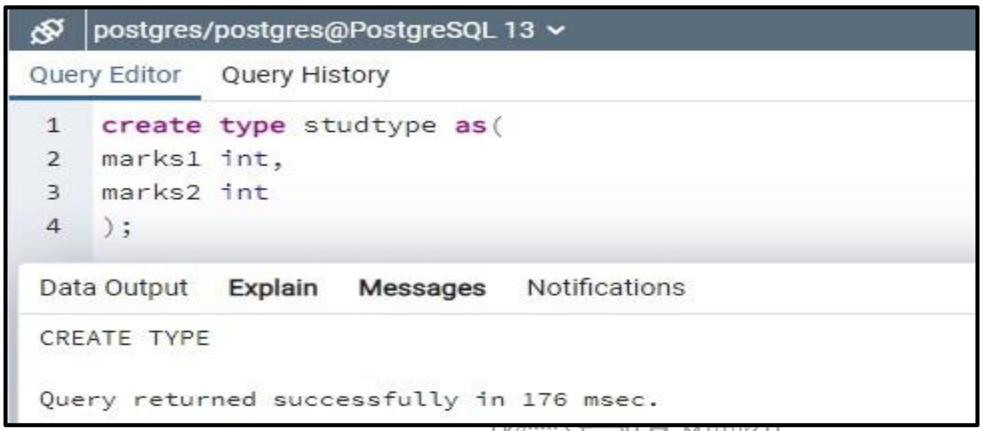
**Procedure / Approach /Algorithm / Activity Diagram:**

Following tasks were performed:

Here, we created type as Student:
**Query:**

```
create type studtype as(
marks1 int,
marks2 int
);
```

**Output:**



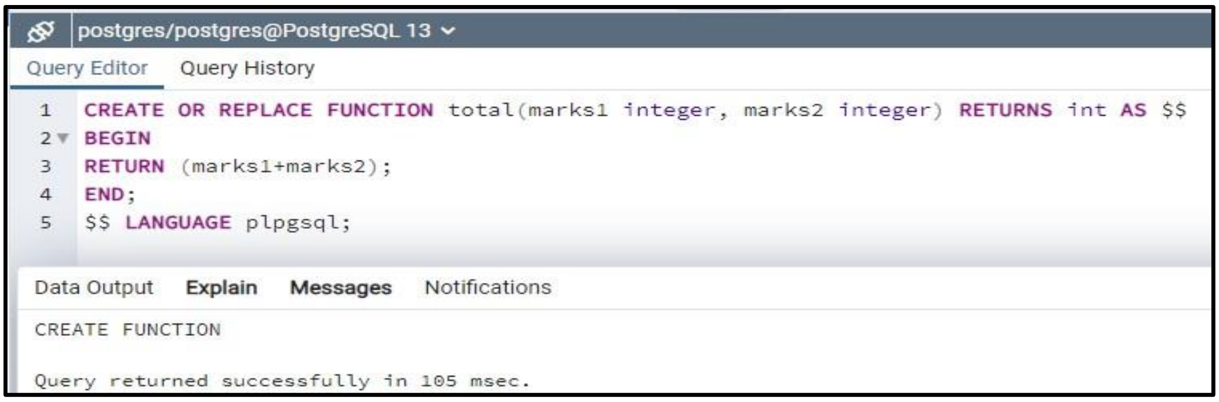
```
create type studtype as(
marks1 int,
marks2 int
);
```

```
CREATE TYPE

Query returned successfully in 176 msec.
```

Here, we Created a function ‘Total’ and calculated the total marks using the defined function:
**Query:**

```
CREATE OR REPLACE FUNCTION total(marks1 integer, marks2 integer)
RETURNS int AS $$
BEGIN
RETURN (marks1+marks2);
END;
$$ LANGUAGE plpgsql;
```

**Output:**



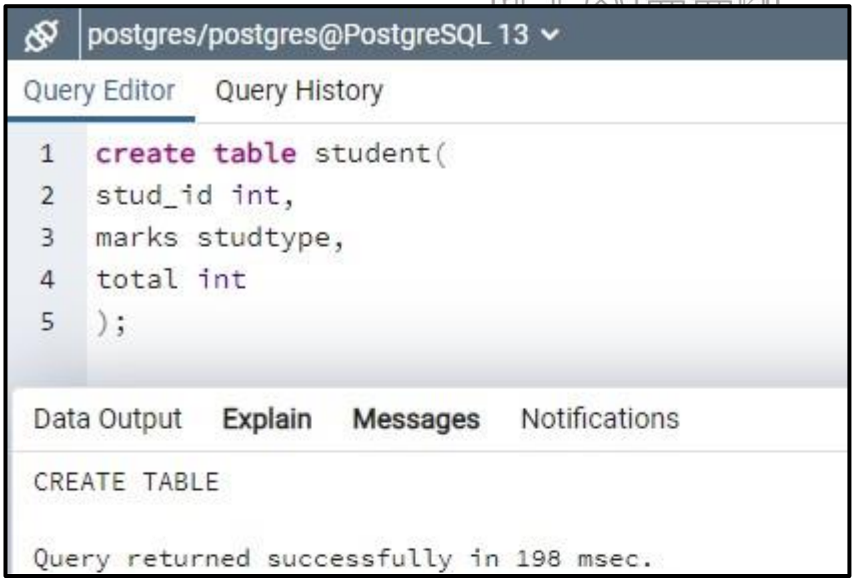
```
CREATE OR REPLACE FUNCTION total(marks1 integer, marks2 integer) RETURNS int AS $$
BEGIN
RETURN (marks1+marks2);
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION

Query returned successfully in 105 msec.
```

Here, we Created the table Student with attributes ID, marks and total marks:
**Query:**

*create table student(* *stud_id*
*int,* *marks studtype,* *total*
*int*
*);*

**Output:**



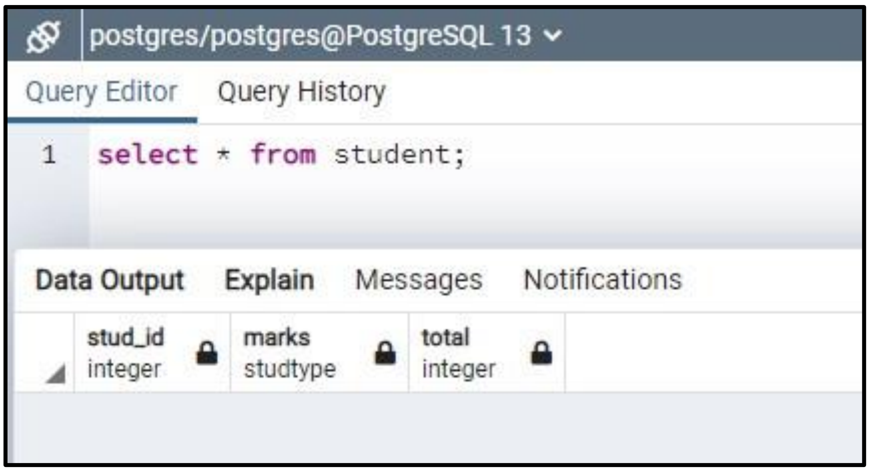
```
create table student(
stud_id int,
marks studtype,
total int
);
```

```
CREATE TABLE

Query returned successfully in 198 msec.
```

Here, we Displayed the student table:
**Query:**

*select * from student;*

**Output:**



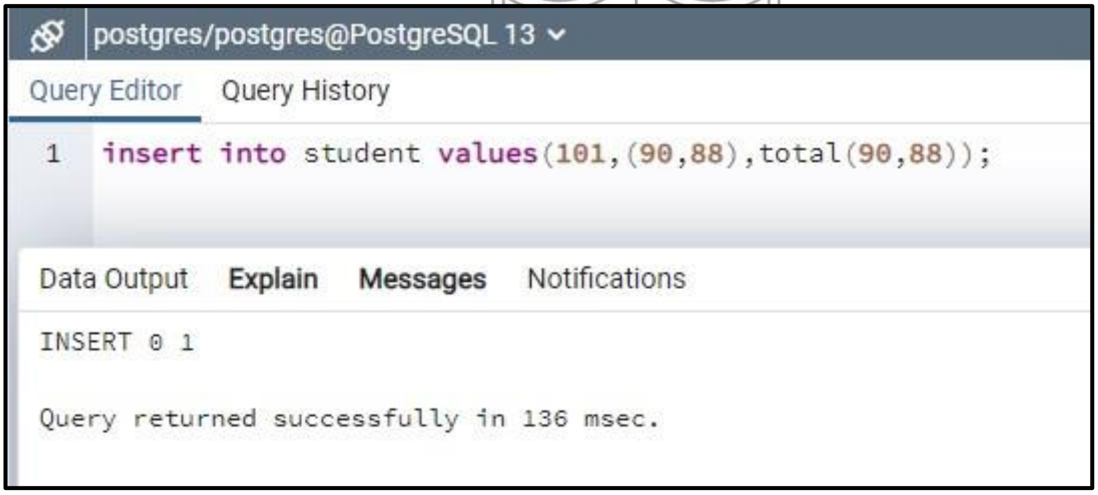
```
select * from student;
```

```
stud_id   marks      total
integer   studtype   integer
```

Here, we Inserted the values into student table:
**Query:**

*insert into student values(101,(90,88),total(90,88));*

**Output:**



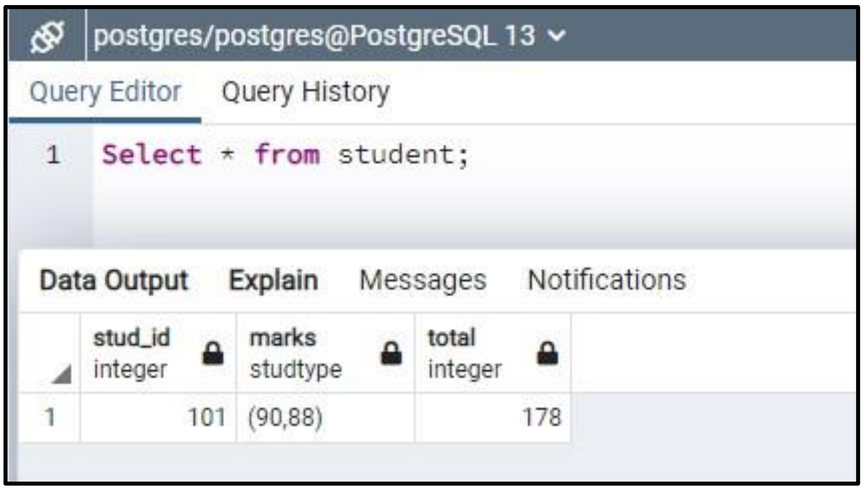
```
insert into student values(101,(90,88),total(90,88));
```

```
INSERT 0 1

Query returned successfully in 136 msec.
```

Here, we Displayed the student table:
**Query:**

*Select * from student;*

**Output:**



```
Select * from student;
```

```
  stud_id   marks      total
  integer   studtype   integer
1     101   (90,88)        178
```

---

**Questions:**

1. **What is the difference between object relational and object oriented databases?**
   **Ans:** a)Relational Database:

•A relational database is a database that stores data in tables that consist of rows and columns.
•Each row has a primary key and each column has a unique name.
•A file processing environment uses the terms file, record, and field to represent data b)Object-oriented database:
•An object-oriented database (OODB) stores data in objects.
•An object is an item that contains data, as well as the actions that read or process the data.
•A Student object, for example, might contain data about a student such as Student ID, First Name, Last Name, Address, and so on.

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

**Ans:** MySQL
This is one of the most popular relational database systems. Originally an open-source solution, MySQL now is owned by Oracle Corporation. Today, MySQL is a pillar of LAMP application software. That means it's a part of Linux, Apache, MySQL, and Perl/PHP/Python stack. Having C and C++ under the hood, MySQL works well with such system platforms as Windows, Linux, MacOS, IRIX, and others.

Oracle
Oracle is a relational database management system created and run by the Oracle Corporation. Currently, it supports multiple data models like document, graph, relational, and key-value within the single database. In its latest releases, it refocused on cloud computing. Oracle database engine licensing is fully proprietary, with both free and paid options available.

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

**Ans:** SQL UPDATE syntax

The UPDATE statement changes existing data in one or more rows in a table. The following illustrates the syntax of the UPDATE statement:

To update data in a table, you need to:

First, specify the table name that you want to change data in the UPDATE clause.
Second, assign a new value for the column that you want to update. In case you want to update data in multiple columns, each column = value pair is separated by a comma (,). Third, specify which rows you want to update in the WHERE clause. The WHERE clause is optional. If you omit the WHERE clause, all rows in the table will be updated.
The database engine issues a message specifying the number of affected rows after you execute the statement.

---

**Outcomes:**

**CO2:** Design advanced database systems using Object Relational, Spatial and NOSQL Databases and its implementation.

---

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

Thus, we have studied Object Oriented Database Management System and Object Relational Queries and executed the Object Relational Queries.

---

**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