**Experiment No.3**

**Title: Execution of object relational queries**

**Batch:B2 Roll No.:1914078 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 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\_namedata\_type [, ... ] )

Example:

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

create type PointTypeas(

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 LineTypeAS(

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, ((ll.line).end1).x,((ll.line).end2).x

FROM Lines ll;

* 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 ll) and use it to start all paths to the desired subobjects.
* Dropping the ``ll.'' 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 (ll.line).end2

FROM Lines ll;

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

Perform following tasks,

* Create table using object type field
* Insert values in that table
* Retrieve values from the table
* Refer the link given and implement atleast 2 new feature from it  
  <https://www.postgresql.org/docs/9.5/static/sql-createtype.html>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

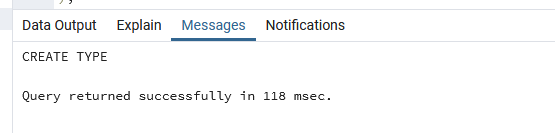
**Results: (Program printout with output / Document printout as per the format)**

**create type PointType AS(**

**x int,**

**y int**

**);**

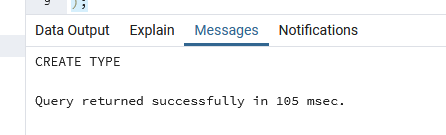


**CREATE TYPE LineType AS(**

**end1 PointType,**

**end2 PointType**

**);**



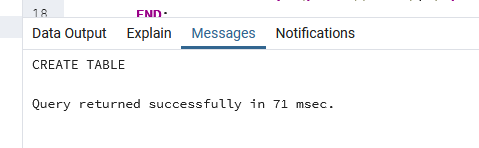
**CREATE TABLE Lines (**

**lineID INT,**

**line LineType,**

**dist float**

**);**



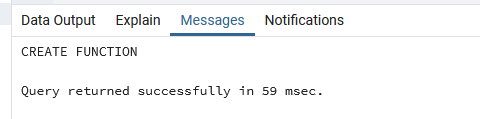
**CREATE OR REPLACE FUNCTION distance(x1 integer,y1 integer,x2 integer,y2 integer) RETURNS float AS $$**

**BEGIN**

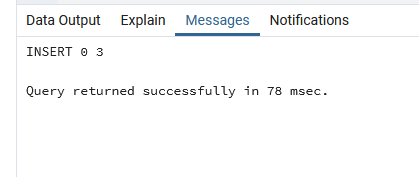
**RETURN sqrt(power((x2-x1),2)+power((y2-y1),2));**

**END;**

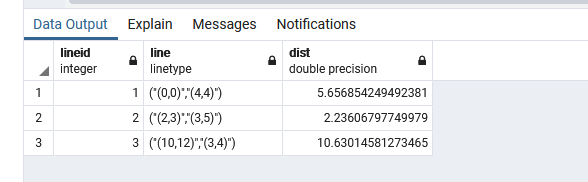
**$$ LANGUAGE plpgsql;**



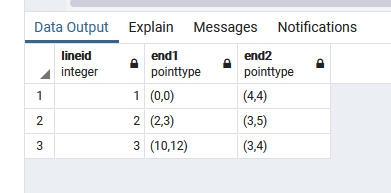
**INSERT INTO Lines Values(1,((0,0),(4,4)),distance(0,0,4,4)),(2,((2,3),(3,5)),distance(2,3,3,5)),(3,((10,12),(3,4)),distance(10,12,3,4));**



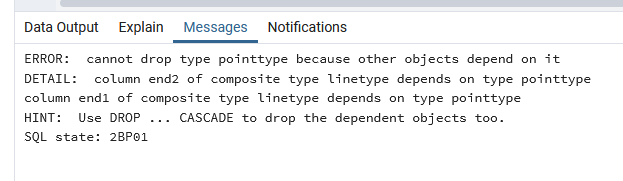
**SELECT \* FROM Lines;**



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

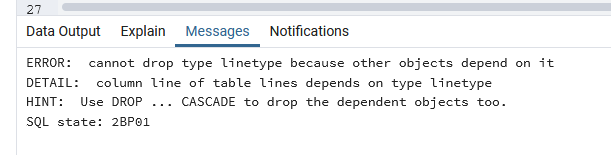


**DROP Type PointType;**



Cannot drop PointType first as Other objects i.e. LineType is dependent on it

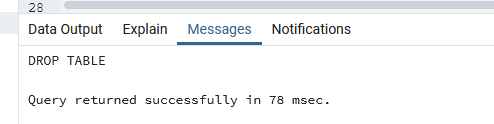
**DROP TYPE LineType;**



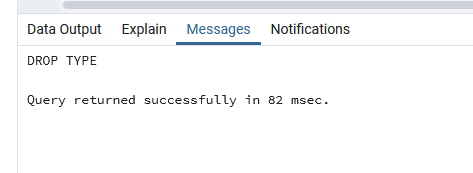
Neither is it possible to drop Linetype as Lines is dependent on it.

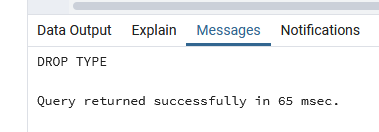
Thus, they need to be dropped in order.

DROP Table Lines;



DROP TYPE LineType;

  
DROP TYPE PointType;



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Questions:**

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

|  |  |
| --- | --- |
| **RDBMS** | **OODMS** |
| Data is stored as entities defined in tabular format. | Data is stored as objects. |
| RDBMS handles simple data | OODBMS handles large and complex data. |
| An entity refers to collection of similar items having same definition. | A class refers to group of objects having common relationships, behaviours and properties. |
| RDBMS handles only data. | |  |  | | --- | --- | |  | OODBMS handles both data and functions operating on that data | |
| A primary key identifies in object in a table uniquely. | Object Id, OID represents an object uniquely in group of objects |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Outcomes:**

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)**

Learned about objects types and object relational queries and executed them in PSQL

**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