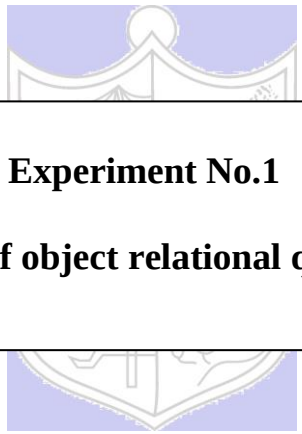


Experiment No.1

Title: Execution of object relational queries

Batch: **Roll No.:** **Experiment No.:1**

Aim: To execute 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:

Oracle allows defining types similar to the types of SQL. The syntax is

```
CREATE TYPE t AS OBJECT (
   list of attributes and methods
);
```

In PostgreSQL the syntax is as follows,

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

Example:

A definition of a point type consisting of two numbers in Oracle is:

```
CREATE TYPE PointType AS OBJECT (
   x NUMBER,
   y NUMBER
);
```

In PostgrSQL it is as follows,

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

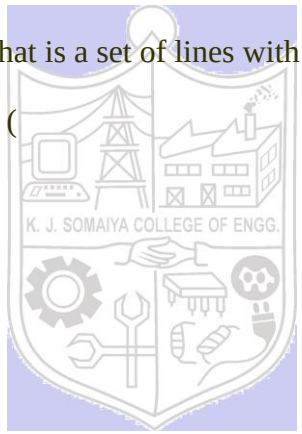
An object type can be used like any other type in further declarations of object-types or table-types.

Define a line type by:

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

Then, creating a relation that is a set of lines with ``line ID's'' as:

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



Dropping Types:

To drop type for example LineType, command will be :

```
DROP TYPE Linetype;
```

Constructing Object Values:

Like C++, Oracle provides built-in constructors for values of a declared type, and these constructors bear the name of the type. Thus, a value of type PointType is formed by the word PointType and 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, LineType(
            PointType(0.0, 0.0),
            PointType(3.0, 4.0)
```

```
        )
);
```

Declaring and Defining Methods:

A type declaration can also include methods that are defined on values of that type. The method is declared by MEMBER FUNCTION or MEMBER PROCEDURE in the CREATE TYPE statement, and the code for the function itself (the definition of the method) is in a separate CREATE TYPE BODY statement.

Methods have available a special tuple variable SELF, which refers to the ``current'' tuple. If SELF is used in the definition of the method, then the context must be such that a particular tuple is referred to.

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 lengths of all the lines in relation Lines, using scale factor 2 (i.e., it actually produces twice these lengths).

```
SELECT lineID, ll.line.length(2.0)
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.end1.x, ll.line.end1.y
FROM Lines ll;
```

prints the x and y coordinates of the first end of each line.

```
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. For instance, one line of output would be PointType(3,4). Notice that type constructors are used for output as well as for input.

Procedure:

Perform following tasks:

- Create table using object type field
- Insert values in that table
- Retrieve values from the table

Results: (Program printout with output)

Questions:

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

Outcomes:

Conclusion: (Conclusion to be based on outcomes achieved)

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