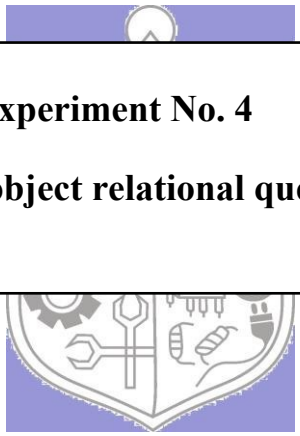

Experiment No. 4

Title: Execution of object relational queries

Batch: A2 **Roll No.: 16010421059** **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.

```
1  create type PointType as(
2      x int,
3      y int
4  );
5
6  CREATE TYPE LineType AS(
7      end1 PointType,
8      end2 PointType
9  );
10
11 CREATE TABLE Lines (
12     lineID INT,
13     line LineType,
14     dist float
15 );
16
17 CREATE OR REPLACE FUNCTION distance(x1 integer, y1 integer,x2 integer,y2 integer) RETURNS float AS $$
18 BEGIN
19 RETURN sqrt(power((x2-x1),2)+power((y2-y1),2));
20 END;
21 $$ LANGUAGE plpgsql;
22
23 INSERT INTO Lines VALUES(27,((0,0),(3,4)),distance(0,0,3,4));
24
25 SELECT * FROM Lines
```

Data Output | Explain | Messages | History

lineid integer	line linetype	dist double pr...
27	("(0,0)","(3,4)")	5

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;
```

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

Data Output | Explain | Messages | History

lineid integer	x integer	x integer
27	0	3

- 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;
```

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

Data Output | Explain | Messages | History

end2 pointtype
(3,4)

Total query runtime: 333 msec.
1 rows retrieved.

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;
```

```
31  CREATE TABLE point of PointType;
32
```

Data Output | Explain | Messages | History

```
CREATE TABLE

Query returned successfully in 345 msec.
```

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

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

 select * from axis;
```

```
33  CREATE TABLE axis (
34  z int
35  ) inherits (point);
36
37  INSERT INTO axis values(2,5,6);
38
39  select * from axis;
```

Data Output | Explain | Messages | History

x integer	y integer	z integer
2	5	6

TASK 2:

```
1  create type numbers as (n1 int, n2 int);
```

Data Output | Explain | Messages | History

```
CREATE TYPE

Query returned successfully in 373 msec.
```

```
create type numbers as (n1 int, n2 int);
create table sum1 (num numbers, sum_of_numbers int);
create or replace function _sum_ (n1 integer, n2 integer) Returns int as $$
Begin
return n1+n2;
end;
$$ language plpgsql;
insert into sum1 values (3, 4, _sum_(3,4));
```

Data Output | Explain | Messages | History

```
CREATE TABLE

Query returned successfully in 454 msec.
```

```
create type numbers as (n1 int, n2 int);
create table sum1 (num numbers, sum_of_numbers int);
create or replace function _sum_ (n1 integer, n2 integer) Returns int as $$
Begin
return n1+n2;
end;
$$ language plpgsql;
insert into sum1 values (3, 4, _sum_(3,4));
```

Data Output | Explain | Messages | History

```
CREATE FUNCTION

Query returned successfully in 382 msec.
```

```
create or replace function _sum_ (n1 integer, n2 integer) Returns int as $$
Begin
return n1+n2;
end;
$$ language plpgsql;
insert into sum1 values ((3, 4), _sum_(3,4));
select * from sum1;
```

Data Output | Explain | Messages | History

```
INSERT 0 1

Query returned successfully in 313 msec.
```

```
return n1+n2;
end;
$$ language plpgsql;
insert into sum1 values ((3, 4), _sum_(3,4));
select * from sum1;
```

Data Output | Explain | Messages | History

num numbers	sum_of_... integer
(3,4)	7

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

Questions:

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

BASIS	RDBMS	OODBMS
Long Form	Stands for Relational Database Management System.	Stands for Object Oriented Database Management System.
Way of storing data	Stores data in Entities, defined as tables hold specific information.	Stores data as Objects.
Data Complexity	Handles comparatively simpler data.	Handles larger and complex data than RDBMS.
Grouping	Entity type refers to the collection of entity that share a common definition.	Class describes a group of objects that have common relationships, behaviors, and also have similar properties.
Data Handling	RDBMS stores only data.	Stores data as well as methods to use it.
Main Objective	Data Independence from application program.	Data Encapsulation.
Key	A Primary key distinctively identifies an object in a table..	An object identifier (OID) is an unambiguous, long-term name for any type of object or entity.

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

Differences Between MS SQL Server and Oracle Database

- In general, the Oracle Database is considered to be much more complex than MS SQL Server. That being said, it is meant for larger organizations where a larger database is needed. While the MS SQL Server offers an enterprise version, it is only compatible with Windows and Linux.

- Oracle can be used on any operating system. One of the biggest differences is transaction control, meaning a group of tasks that can be treated as a single unit. So, suppose a set of records must all be updated simultaneously, by default. In that case, SQL Server executes each command individually, and it will be extremely difficult to make changes if any errors are encountered along the way. Oracle, on the other hand, treats each new database connection as a new transaction.

- Next is the organization of these databases. __"MS SQL Server organizes all objects, such as tables, views, and procedures, by database names. Users are assigned to a login, which is granted access to the specific database and its objects. Also, in SQL Server, each database has a private, unshared disk file on the server. In Oracle, all the database objects are grouped by schemas, which are a subset collection of database objects, and all the database objects are shared among all schemas and users. Even though it is all shared, each user can be limited to certain schemas and tables via roles and permissions."

- In terms of functionality (this is a bit technical), MS SQL Server does not offer partitioning, bitmap indexes, reverse key indexes, function-based indexes, or star query optimization, all of which Oracle offers. Both are widely used across the enterprise landscape, but RDBMS is considered superior as a matter of preference and what that particular database is being used for.

- The major distinction between MS SQL and Oracle is the Transaction Control. MS SQL will, by default, perform and commit each job or query separately. Hence, it is not simple or difficult to roll back transactions if any error is encountered in the method. The "Begin Transaction" command is utilized at the start of a transaction for accurate group statements. While the "Commit" statement is employed at the end of the group statement. The modified data is written to the disk and completes the transaction in the Commit statement. In the transaction, any modifications performed within the transaction block are rejected in the Rollback. However, with decent error handling, the rollback command can provide some security against data corruption.

- In Oracle, each new database link is interpreted as a new transaction. Until the transaction is committed, the transaction can be rolled back, and all the modifications are performed on the system memory. Due to that, in the rollback, all the variations in the statement can be unhitched. After the commit is fulfilled, typically, the next command starts a new transaction. This serves to check errors efficiently and provide compliance.

- MS SQL coordinates all the objects like tables, procedures, and views by database titles. Furthermore, MS SQL databases don't share private disk files on the machine. Users are authorized to login and gain privileged access to the chosen database and its objects. But in Oracle, the database objects are classified by using schemas. Schemas are a segment collection of database objects. All the database objects can be distributed to all users. Schemas and table access can be defined or restricted by roles and permissions.

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

Ans.

first, we are going to create a UDT table and then we will modify it as per need. So let’s consider Electricity_bill is a table name.

```
CREATE TYPE Electricity_bill
 (
  Bill_id int,
  Due_date date,
```

```
  Submit_date date
 );
```

Now, let's verify the user-defined type by using the following CQL query given below.

```
DESCRIBE TYPE Electricity_bill;
```

To add a new column in the user-defined type used the following CQL query.

```
ALTER TYPE cluster1.Electricity_bill
ADD name text;
```

To rename the existing field 'RENAME' keyword can be used. Let's have a look.

```
ALTER TYPE cluster1.Electricity_bill
RENAME name TO full_name;
```

Now, let's verify the modifying user-defined type by using the following CQL query given below.

```
DESCRIBE TYPE Electricity_bill;
```

Restriction: In the case of modifying UDTs, there is a restriction that modifying UDTs in the primary key is not supported and also not supported for the index column and changing column type of UDTs is also not supported.

Outcomes:

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

Conclusion:

Executed object relational queries successfully.

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