PostgreSQL

Session Objectives

- Creating a new class
- Populating a class with instances
- Querying the class
- Altering the structure of the class
- Updation
- Deletion

Classes

The fundamental notion in Postgres is that of a class, which is a named collection of object instances. Each instance has the same collection of named attributes, and each attribute is of a specific type. Furthermore, each instance has a permanent object identifier (**OID**) that is unique throughout the installation. Since SQL syntax refers to tables, we will use the terms **table** and **class** interchangeably. Likewise, an SQL row is an instance and SQL columns are attributes. Classes are grouped into databases, and a collection of databases managed by a single postmaster process constitutes an installation or site.

Before creating a new class, create a database (described in the previous session) named **employee**.

Creating a new Class

A new class can be created by specifying the class name, along with all attribute names and their types. The syntax is

```
create table <tablename> (col1 datatype, col2 datatype, ….. coln
datatype);
```

Both keywords and identifiers are case-insensitive; identifiers can preserve case by surrounding them with double-quotes as allowed by SQL92. Postgres SQL supports the usual SQL types int, float, real, smallint, char(N), varchar(N), date, time etc. Postgres can be customized with an arbitrary number of user-defined data types. The Postgres CREATE command is exactly similar to the command used to create a table in a traditional relational system. However, we will presently see that classes have properties that are extensions of the relational model.

If we want to create a table called emp, the following command can be used inside the database

```
employee=# create table emp(eno int,ename varchar(20),basic
float,hra float);
CREATE
employee=#
```

Tables can also be created from existing tables. The syntax is

```
CREATE TABLE <TABLE NAME2> AS SELECT FIELD NAMES FROM <TABLE NAME1>;
```

A duplicate emp table can be created from emp table as

```
employee=# create table dupemp as select * from emp;
SELECT
employee=# select * from dupemp;
eno |  ename   | basic | hra  | qual
-----+----------+-------+------+------
100 | Peter    |  4000 | 1800 | B.E
101 | James    |  5000 | 2300 | MCA
102 | Williams |  8000 | 4600 | MBA
103 | John     |  7800 | 3170 | B.E
(4 rows)
employee=#
```

To create a table with specific fields the command is:

```
employee=# create table dupemp1 as select eno,ename from emp;
SELECT
employee=# select * from dupemp1;
 eno |  ename
-----+----------
 100 | Peter
 101 | James
 102 | Williams
 103 | John
(4 rows)
employee=#
```

To create only the structure of a table from an existing table, the command is

```
employee=# create table dupemp2 as select * from temp where 1=2;
SELECT
employee=# select * from dupemp2;
eno | ename | basic | hra | qual
-----+-------+-------+-----+------
(0 rows)

employee=#
```

Since the condition specified in the where clause can never be satisfied, no records are retrieved by the select statement and only the structure gets copied.

Populating a Class with Instances

The INSERT statement is used to populate a class with instances. The syntax of the insert statement is

```
insert into <tablename> [<col1,col2,…coln>] values (val1,val2,…valn)
```

For example, consider the following insert statement:

```
employee=# insert into emp values(100,'Peter',4000,1800);
INSERT 19119 1
employee=#
```

Selective insertion

Sometimes it might be necessary to insert values only for a few columns in a table. If we want to insert values only for eno and ename in the emp table, then the insert statement should be modified as shown below:

```
employee=# insert into emp(eno,ename) values(101,'James');
INSERT 19073 1
employee=#
```

The copy command

It is also possible to use the COPY command to load large amounts of data from flat (ASCII) files. This is usually faster because the data is read (or written) as a single atomic transaction directly to or from the target table. An example would be:

```
employee=# copy emp from '/var/lib/pgsql/emp.txt' using delimiters
'|';
COPY
employee=#
```

where the path name for the source file must be available to the backend server machine, not the client, since the backend server reads the file directly.

Querying a Class

A class can be queried with normal relational selection and projection queries. A SQL **SELECT** statement is used to do this. The statement is divided into a target list (the part that lists the attributes to be returned) and a qualification (the part that specifies any restrictions).

For example, to retrieve all the rows of the class emp, type:

```
employee=# select * from emp;
 eno |  ename   | basic | hra
-----+----------+-------+------
 100 | Peter    |  4000 | 1800
 101 | James    |  5000 | 2300
 102 | Williams |  8000 | 4600
 103 | John     |  7800 | 3170
(4 rows)

employee =#
```

To retrieve only specific columns, we can specify the column name in the target list. For example,

```
employee=# select eno,ename from emp;
 eno |  ename
-----+---------
 100 | Peter
 101 | James
 102 | Williams
 103 | John
(4 rows)

employee =#
```

It is possible to specify any arbitrary expressions in the target list. For example, if we want to find the average salary of the employees in the organization, we can say:

```
employee =# select avg(basic) AS basic_avg from emp;
 basic_avg
---------
```

```
    6200
(1 row)
employee =#
```

If we want to find the number of employees in the organization, we can say:

```
employee=# select count(eno) from emp;
 count
-------
     5
(1 row)
employee=#
```

If we want to find the total salary of the employees, we can type:

```
employee =# select eno,ename,(basic+hra) as total_sal from emp;
 eno |  ename   | total_sal
-----+----------+-----------
 100 | Peter    |      5800
 101 | James    |      7300
 102 | Williams |     12600
 103 | John     |     10970
(4 rows)

employee =#
```

The like operator

To display details of employees whose names begin with J

```
employee =# select * from emp where ename like 'J%';
 eno | ename | basic | hra
-----+-------+-------+------
 101 | James |  5000 | 2300
 103 | John  |  7800 | 3170
(2 rows)

employee =#
```

The **%** symbol matches one or more character after the alphabet. The **_** symbol matches just a single character.

Assume we have two employees raj and raji, if we use the % symbol to extract employee names beginning with r, both the names would be displayed. In such a case, we can use the _ symbol:

```
select ename from emp where ename like ‘raj_’
ename
--------
raji
```

The between operator

The between operator displays values within a particular range specified in the where clause.

If we want the details of employees whose basic is between 4500 and 7500,

```
employee =# select * from emp where basic between 4500 and 7500;
 eno | ename | basic | hra
-----+-------+-------+------
 101 | James |  5000 | 2300
(1 row)
employee =#
```

Redirecting SELECT Queries

Any SELECT query can be redirected to a new class. Consider the following statement:

```
employee=# select *  into table temp from emp;
SELECT
employee=# select * from temp;
 eno |  ename   | basic | hra  | qual
-----+----------+-------+------+------
 100 | Peter    |  4000 | 1800 | B.E
 101 | James    |  5000 | 2300 | MCA
 102 | Williams |  8000 | 4600 | MBA
 103 | John     |  7800 | 3170 | B.E
(4 rows)

employee=#
```

This forms an implicit CREATE command, creating a new class **temp** with the attribute names and types specified in the target list of the SELECT INTO command. We can then, of course, perform any operations on the resulting class that we can perform on other classes.

Altering the structure of the class

ALTER statements are used to change the existing database objects. If the tables created are to be added with new columns then ALTER TABLE statement can be used.

The syntax is:

```
            alter table <tablename> add (column_name datatype);
```

For example, add a column called qual to the emp table.

```
employee=# alter table emp add qual varchar(4);
ALTER
employee =#

employee =# select * from emp;
 eno |  ename   | basic | hra  | qual
-----+----------+-------+------+------
 100 | Peter    |  4000 | 1800 |
 101 | James    |  5000 | 2300 |
 102 | Williams |  8000 | 4600 |
 103 | John     |  7800 | 3170 |
(4 rows)
employee =#
```

Updating the class

The update command can be used to update existing instances. In the previous case, we have added a column called qual to the emp table. If we want to populate that column, we can go in for the update command.

The syntax for the UPDATE statement is:

```
UPDATE <tablename> SET <column(s)> = <expr> WHERE <condition>
```

To update the qual column in our table,proceed as follows:

```
employee=# update emp set qual='B.E' where eno=100;
UPDATE 1
employee=#
```

If the where condition is not specified, the value specified is set for all the records.

Update the other instances. Now the table contains the values:

```
employee=# select * from emp;
 eno |  ename   | basic | hra  | qual
-----+----------+-------+------+------
 100 | Peter    |  4000 | 1800 | B.E
 101 | James    |  5000 | 2300 | MCA
 102 | Williams |  8000 | 4600 | MBA
 103 | John     |  7800 | 3170 | B.E
(4 rows)

employee=#
```

Querying with boolean operators

Arbitrary Boolean operators (AND, OR and NOT) are allowed in the qualification of any query.

For example, if we want to select all employees who are engineers having basic is more than 5000, we can use the following query:

```
employee=# select * from emp where qual='B.E' and basic>4000;
 eno | ename | basic | hra  | qual
-----+-------+-------+------+------
 103 | John  |  7800 | 3170 | B.E
(1 row)

employee=#
```

To display details of employees whose qualification is either B.E or MBA, we can say

```
employee=# select * from emp where qual='B.E' or qual='MCA';
 eno | ename | basic | hra  | qual
-----+-------+-------+------+------
 100 | Peter |  4000 | 1800 | B.E
 101 | James |  5000 | 2300 | MCA
 103 | John  |  7800 | 3170 | B.E
(3 rows)

employee=#
```

Removing duplicate instances

Duplicate occurences of instances can be removed by specifying the distinct keyword.

If we want to display the qualification of all the employees in the organization, we can say

```
employee=# select distinct qual from emp;
 qual
------
 B.E
 MBA
 MCA
(3 rows)

employee=#
```

If the distinct keyword had not been used the output would be as shown below:

```
employee=# select qual from emp;
 qual
------
 B.E
 MCA
 MBA
 B.E
(4 rows)

employee=#
```

Displaying in sorted order

It is possible to order records in ascending or descending order by using the order by clause. One or more columns can be chosen for sorting the records in ASCENDING OR in DESCENDING.

Ascending Order (Default)

- Earliest date is listed first.
- Characters are sorted alphabetically.
- Numeric values have the lowest value first.
- Null values are moved to the end of the result set.

If we want to sort employee records by ename, we can use the following command

```
employee=# select * from emp order by ename;
 eno |  ename   | basic | hra  | qual
-----+----------+-------+------+------
 101 | James    |  5000 | 2300 | MCA
 103 | John     |  7800 | 3170 | B.E
 100 | Peter    |  4000 | 1800 | B.E
 102 | Williams |  8000 | 4600 | MBA
(4 rows)

employee=#
```

To display employee details in descending order of basic salary,

```
employee=# select * from emp order by basic desc;
 eno |  ename   | basic | hra  | qual
 -----+----------+-------+------+------
 102 | Williams |  8000 | 4600 | MBA
 103 | John     |  7800 | 3170 | B.E
 101 | James    |  5000 | 2300 | MCA
 100 | Peter    |  4000 | 1800 | B.E
(4 rows)

employee=#
```

Deleting records

Deletions are performed using the DELETE command.

```
employee=# delete from emp where eno=103;
DELETE 1
employee=# select * from emp;
 eno |  ename   | basic | hra  | qual
-----+----------+-------+------+------
 100 | Peter    |  4000 | 1800 | B.E
 101 | James    |  5000 | 2300 | MCA
 102 | Williams |  8000 | 4600 | MBA
(3 rows)

employee=#
```

All employee details belonging to eno 103 are removed. One should be wary of queries of the form

```
DELETE FROM classname;
```

Without a where clause, DELETE will simply remove all instances of the given class, leaving it empty. The system will not request confirmation before doing this.

Dropping a class

Drop statements are used to remove the database objects from the database. The syntax of the drop command is

```
            DROP DATABASEOBJECT <DATABASEOBJECT NAME>;
```

```
employee=# drop table emp;
DROP
employee=# select * from emp;
ERROR:  Relation 'emp' does not exist
employee=#
```

When a table is dropped, the structure and contents are also dropped.

Truncating a class

To remove only the contents of a table and not the structure, the TRUNCATE statement can be used.

The syntax is:

```
            TRUNCATE TABLE <TABLE NAME>;
```

```
employee=# truncate table emp;
TRUNCATE
employee=# select * from emp;
 eno | ename | basic | hra | qual
-----+-------+-------+-----+------
(0 rows)

employee=#
```

Lab Exercises

1. Create a table SUPPLIER_MASTER with the following structure.

```
SUPP_CODE      CHAR(4)
SUPP_NAME      VARCHAR(15)
ADDRESS1       VARCHAR(20)
ADDRESS2       VARCHAR(20)
CITY           VARCHAR(20)
PHONE          VARCHAR(6)
```

2. Create table ITEM_MASTER with the following structure:

```
ITEM_CODE            CHAR(4)
ITEM_TYPE            CHAR(1)
ITEM_NAME            VARCHAR2(15)
QTY_ON_HAND          NUMBER(4)
REORDER_LEVEL        NUMBER(4)
```

3. Add a column UNIT_PRICE to the ITEM_MASTER table. The data type should be float.

4. Create a table ITEM_MASTER1 with a structure that is similar to ITEM_MASTER.

5. Create a table PURCHASE_ORDER with the following structure.

```
PO_NUMBER           CHAR(4)
PO_DATE              DATE
```

```
SUPP_CODE            CHAR(4)
```

6. Create a table ORDER_DETAILS with the following structure.

```
PO_NUMBER      CHAR(4)
ITEM_CODE      CHAR(4)
QUANTITY NUMBER(4)
TRANS_DATE     DATE
```

Select all the employees who joined after 10/may/1998.

1. What is the maximum salary offered by the organization? (Hint: use the function max())
2. Select all the employees whose salary is greater than 5000 and whose qualification is MCA.