BYU IDAHO CS 313 | Web Engineering II

05 Prepare : Basic Queries

This document covers the basics of retrieving, or querying, data from a Postgres database.

I. Creating Tables

Before we can retrieve data from the database, we need to create some tables and insert data into them. For this example we will create two tables: a note_user table containing an id, a username, and a password; and a note table containing an id, the userId of the note user that created it, and the content of the note. Recall from our previous discussion, that these tables can be created with the following commands.

```
CREATE TABLE note_user (
  id SERIAL,
  username VARCHAR(255),
  password VARCHAR(255),
  PRIMARY KEY (id)
);

CREATE TABLE note (
  id SERIAL,
  userId INT NOT NULL,
  content TEXT,
  PRIMARY KEY (id),
  FOREIGN KEY (userId) REFERENCES note_user (id)
);
```

II. Inserting Data

To insert data into these tables, we use the following syntax.

INSERT INTO tableName (column1, column2, etc.) VALUES (value1, value2, etc.);

Note that the specification of column names is not technically required, but it enables us to specify each specific one for which we are passing data and in what order.

Thus, to insert data into the user table, we can run the following commands.

```
INSERT INTO note_user (username, password) VALUES ('john', 'pass');
INSERT INTO note_user (username, password) VALUES ('jane', 'byui');
```

Note that we do not have to specify the id column for these rows because we have set it to *SERIAL*, so it will automatically be assigned the next integer value.

Similarly, we can insert notes into the notes table as follows. These lines create two notes for the user john, and one for the user jane.

```
INSERT INTO note (userId, content) VALUES (1, 'A note for John');
INSERT INTO note (userId, content) VALUES (1, 'Another note for John');
INSERT INTO note (userId, content) VALUES (2, 'And this is a note for Jane');
```

III. Retrieving Data

At this point, we have two tables in our database, and we have inserted data into them. Now we need to discover how to retrieve the data from these tables. This is done using the SELECT statement as follows.

SELECT columnNames FROM tableName;

To select all rows from the user table, and display every column,

we can use the following command. Note that the * operator will select all columns.

```
SELECT * FROM note_user;

 id | username | password
----+----------+----------
  1 | john     | pass
  2 | jane     | byui
(2 rows)
```

If we only want to receive certain columns, or specify a different order for the columns, we can list specific column names, rather than using the *, as follows.

```
SELECT username FROM note_user;

 username
----------
 john
 jane
(2 rows)
```

```
SELECT password, username FROM note_user;

 password | username
----------+----------
 pass     | john
 byui     | jane
(2 rows)
```

Notice that in each of these cases we are returning all rows from the table. If we want to restrict the rows that are returned, we add a "WHERE clause" at the end of our statement.

SELECT columnNames FROM tableName WHERE conditions;

```
SELECT * FROM note_user WHERE username = 'john';

 id | username | password
----+----------+----------
  1 | john     | pass
(1 row)
```

```
SELECT * FROM note_user WHERE id > 1;

 id | username | password
----+----------+----------
  2 | jane     | byui
(1 row)
```

Another clause that can be added to a query after the optional "where clause" is an "order by clause."

```
SELECT * FROM note_user ORDER BY username;

 id | username | password
----+----------+----------
  2 | jane     | byui
  1 | john     | pass
(2 rows)
```

```
SELECT * FROM note_user ORDER BY username DESC;

 id | username | password
----+----------+----------
  1 | john     | pass
  2 | jane     | byui
(2 rows)
```

Notice in the first example, the rows are ordered by the username, and in the second, they are ordered by username in descending fashion (DESC).

IV. Joining Tables

Using the same format as before, we could select all the notes from the note table.

```
SELECT * FROM note;

 id | userid |           content
----+--------+-----------------------------
  1 |      1 | A note for John
  2 |      1 | Another note for John
  3 |      2 | And this is a note for Jane
(3 rows)
```

But suppose we only wanted the notes for the user 'john.' We could add a where clause to our query like so:

```
SELECT * FROM note where userID = 1;

 id | userid |        content
----+--------+-----------------------
  1 |      1 | A note for John
  2 |      1 | Another note for John
(2 rows)
```

But this requires us to first find out John's userId, if we only know his username. A better approach is to "join" the tables together on the matching column (in this case userId) to temporarily make one big table that has the user information and the note information in it. Then we can use the username in the where clause. The syntax of joining tables is as follows.

```
SELECT columns
  FROM table1
  JOIN table2
  ON columnFromTable1 = columnFromTable2;
```

Then we can of course add other clauses afterward. While this works because these two tables may have columns with the same names, it is useful to give them aliases so that we can say which

table we mean.

```
SELECT columns
  FROM table1 AS t1
  JOIN table2 AS t2
  ON t1.column = t2.column;
```

The aliases t1, and t2 can then be used in other places in the query such as the columns that will be selected. Using the join syntax, we can join the user and note table together and return all rows and all columns from the big temporary table.

```
SELECT * FROM note_user AS u
> JOIN note AS n
> ON u.id = n.userId;

 id | username | password | id | userid |           content
----+----------+----------+----+--------+-----------------------------
  1 | john     | pass     |  1 |      1 | A note for John
  1 | john     | pass     |  2 |      1 | Another note for John
  2 | jane     | byui     |  3 |      2 | And this is a note for Jane
(3 rows)
```

You can see that all columns and rows from both tables are present, and that data such as John's username has been repeated wherever required.

Using this syntax, a more useful query could be to return all of John's notes.

```
SELECT * FROM note_user AS u
> JOIN note AS n
> ON u.id = n.userId
> WHERE u.username = 'john';

 id | username | password | id | userid |        content
----+----------+----------+----+--------+-----------------------
  1 | john     | pass     |  1 |      1 | A note for John
  1 | john     | pass     |  2 |      1 | Another note for John
(2 rows)
```

And, if we only want to see the content of John's notes, we could change the * to n.content, like so:

```
SELECT n.content FROM note_user AS u
> JOIN note AS n
> ON u.id = n.userId
> WHERE u.username = 'john';

       content
----------------------
A note for John
Another note for John
(2 rows)
```

Notice that in this example, we have specified that we only want to see the content column from the note table (n.content), and that we want to restrict the data to only those rows where the username column from the user table (u.username) is equal to 'john'.