Relational Algebra Queries on PostgreSQL

This document explains how to execute relational algebra (RA) queries on SoC's PostgreSQL server named **postgres01-1**.

To learn how to login to the Linux server **stu** and use psql to connect to the PostgreSQL server **postgres01-1**, please refer to the document named **postgresql-soc.pdf** available on LumiNUS → Files → Project.

1. Syntax

The following table explains the syntax of each relational operator, where R and S denote some relational algebra expressions.

SYNTAX	MEANING
select{c}(R)	selection operator on R with predicate c
project{l}(R)	projection operator on R with attribute list l
rename{a1:b1,,..,an:bn}(R)	rename attributes {a1,...,an} of R to {b1,...,bn}
R * S	cross product of R and S
R ~ S	natural join of R and S
R ~> S	natural left join of R and S
R <~ S	natural right join of R and S
R <~> S	natural full join of R and S
R ~{c} S	inner join of R and S with join predicate c
R ~>{c} S	left join of R and S with join predicate c
R <~{c} S	right join of R and S with join predicate c
R <~>{c} S	full join of R and S with join predicate c
R & S	set intersection of R and S
R | S	set union of R and S
R - S	set difference of R and S
T = R	define T to denote a relational expression R

- Keywords/relation/attribute names are case-insensitive
- String values must be double quoted in selection predicates (e.g., cname = "Moe") and must not contain the character } or '

2. Examples

This section illustrates examples of how to execute relational algebra (RA) queries on an example database.

First, login to the stu.comp.nus.edu.sg server. Next, copy and unzip the file **ra.zip** to your current directory as follows:

```
$ cp  /home/course/cs2102/public_html/ra/ra.zip .
$ unzip ra.zip
```

Alternatively, the file could also be obtained using the wget command:

```
$ wget https://www.comp.nus.edu.sg/~cs2102/ra/ra.zip
```

The unzipped contents consist of two files: **pizza.sql** and **example.sql**.

The file **pizza.sql** consists of SQL commands (to be covered in week 3) to create a small database for the pizza schema discussed in class. The commands in pizza.sql could be executed as follows:

```
$ psql -f pizza.sql
```

The above usage of psql assumes that the PostgreSQL environment variables (PGHOST, PGPORT, PGDATABASE, PGUSER) have been appropriately configured as discussed in the postgres-soc.pdf document.

The -f option in psql allows commands to be read from a file instead of typing the commands interactively.

Alternatively, you could also execute the commands from within psql using the **\i** meta-command, which enables input to be read from a file:

```
$ psql

psql (10.15 (Ubuntu 10.15-0ubuntu0.18.04.1), server 12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))
WARNING: psql major version 10, server major version 12.
         Some psql features might not work.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

alice=> \i pizza.sql
```

To execute RA queries on PostgreSQL, we use a user-defined function named **ra** which takes a string argument (enclosed in single quotes) as input for a RA query. The function is invoked using SQL's select command as illustrated by the following example to find all pizzas:

```
alice=> select ra('Pizzas');
```

Note that the above SQL's select construct (to be covered later in class) is not to be confused with the selection operator in relational algebra. Each SQL statement must terminate with a semicolon.

The above RA query outputs the contents of the Pizzas relation:

```
      ra
--------------
  PIZZA
 Diavola
 Funghi
 Hawaiian
 Margherita
 Marinanra
 Sciliana
(7 rows)
```

Note that the number of rows indicated is actually one more than the actual number of tuples in the Pizzas relation as it is also counting the output table header "PIZZA".

The following are additional examples (available in **example.sql**) illustrating the syntax of RA queries.

- **Find all restaurants that are located in the North or South area.**

```
select ra('
project{rname}(
    select{area = "North" or area = "South"}(Restaurants)
)
');
```

- **Find pizzas that contain both cheese and chilli.**

```
select ra('
project{pizza}(
    select{ingredient = "cheese"}(Contains)
)
&
project{pizza}(
    select{ingredient = "chilli"}(Contains)
)
');
```

- **Find customers and the pizzas they like; include also customers who don't like any pizza.**

```
select ra('
project{cname,pizza}(
    Customers ~> Likes
)
');
```

It is also possible to express a complex RA query in multiple steps with each intermediate RA expression being assigned to a variable as illustrated below:

```
select ra('
R1 = project{pizza}(select{cname = "Maggie"}(Likes));

R2 = project{rname}(Sells) * R1;

R3 = project{rname}(
    R2 - project{rname,pizza}(Sells)
);

R4 = project{rname}(Sells) - R3;

R5 = rename{pizza:pizza5}(select{cname = "Ralph"}(Likes));

R6 = project{rname}(select{pizza5 = pizza}(Sells * R5));

R4 - R6
');
```

In the above, two consecutive steps must be separated by a semicolon. The output computed by the final step will be displayed. The above query could also be expressed as a single step as follows:

```
select ra('
project{rname}(Sells)
-
project{rname}(
    project{rname}(Sells) * project{pizza}(select{cname = "Maggie"}(Likes))
    -
    project{rname,pizza}(Sells)
)
-
project{rname}(select{pizza5 = pizza}(
    Sells * rename{pizza:pizza5}(select{cname = "Ralph"}(Likes))
))
');
```

3. Editing Queries

In psql, any SQL command that has been typed but has not yet been sent for execution is stored in a memory buffer called **query buffer**. The contents of this buffer can be edited by invoking a configurable text editor within psql. The default editor is vi and this can be configured using the EDITOR environment variable. For example, to configure the editor to nano, add the following line to your ~/.bashrc file:

```
export EDITOR=nano
```

To invoke the editor, use the psql meta-command **\e**.

The following table shows some of the additional useful psql's meta-commands.

META-COMMAND	MEANING
\q	Quit psql.
\d	List all tables in the database.
\d foo	List information on the relation named foo.
\p	Display the contents of the query buffer; if the current query buffer is empty, display the most recently executed query.
\w foo	Write the contents of the query buffer to the file named foo; if the current query buffer is empty, write the most recently executed query to foo.
\r	Clear the query buffer.
\e	Invoke the text editor to edit the contents of the query buffer; if the query buffer is empty, edit the mostly recently executed query.
\e foo	Invoke the text editor to edit the contents of a file named foo. The contents of the edited file will be copied to the query buffer at the end of the edit session.
\o foo	Enable future query results to be saved to the file named foo.
\g	Send the contents of the current query buffer to the database server for execution; if the current query buffer is empty, the most recently sent query is re-executed.
\i foo	Reads the contents from the file named foo and sent their contents to the database server for execution.
\!	Escapes from the psql session to a sub-shell. The psql session resumes when the sub-shell is exited.

To learn more about psql, you can read its manual online (https://www.postgresql.org/docs/current/app-psql.html) or on the stu server using the **man** command (i.e., man psql).

Last updated 2021-01-22 08:13:27 +08