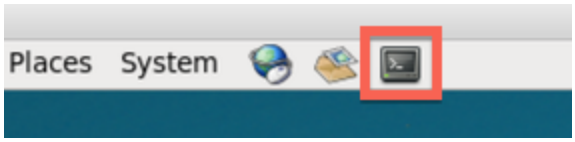
Querying Relational Data with Postgres

By this end of this activity, you will be able to:

1. View table and column definitions, and perform SQL queries in the Postgres shell
2. Query the contents of SQL tables
3. Filter table rows and columns
4. Combine two tables by joining on a column

Step 1. **Open a terminal window and start Postgres shell.** Open a terminal window by clicking on the square black box on the top left of the screen.



Next, start the Postgres shell by running *psql:*

```
[cloudera@quickstart big-data-3]$ psql
psql (8.4.20)
Type "help" for help.

cloudera=#
```

Step 2. **View table and column definitions.** We can list the tables in the database with the *d* command:

```
cloudera=# \d
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | adclicks   | table | cloudera
 public | buyclicks  | table | cloudera
 public | gameclicks | table | cloudera
(3 rows)
```

The database contains three tables: *adclicks, buyclicks,* and *gameclicks.* We can see the column definitions of the *buyclicks* table by running *\d buyclicks:*

```
cloudera=# \d buyclicks
                Table "public.buyclicks"
    Column     |            Type             | Modifiers
---------------+-----------------------------+-----------
 timestamp     | timestamp without time zone | not null
 txid          | integer                     | not null
 usersessionid | integer                     | not null
 team          | integer                     | not null
 userid        | integer                     | not null
 buyid         | integer                     | not null
 price         | double precision            | not null
```

This shows that the *buyclicks* table has seven columns, and what each column name and data type is.

Step 3. **Query table.** We can run the following command to view the contents of the *buyclicks* table:

```
select * from buyclicks;
```

The *select* * means we want to query all the columns, and *from buyclicks* denotes which table to query. Note that all query commands in the Postgres shell must end with a semi-colon.

The result of the query is:

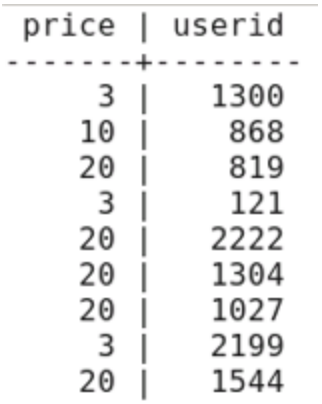
```
     timestamp      | txid  | usersessionid | team | userid | buyid | price
--------------------+-------+---------------+------+--------+-------+-------
 2016-05-26 15:36:54 |  6004 |          5820 |    9 |   1300 |     2 |     3
 2016-05-26 15:36:54 |  6005 |          5775 |   35 |    868 |     4 |    10
 2016-05-26 15:36:54 |  6006 |          5679 |   97 |    819 |     5 |    20
 2016-05-26 16:36:54 |  6067 |          5665 |   18 |    121 |     2 |     3
 2016-05-26 17:06:54 |  6093 |          5709 |   11 |   2222 |     5 |    20
 2016-05-26 17:06:54 |  6094 |          5798 |   77 |   1304 |     5 |    20
 2016-05-26 18:06:54 |  6155 |          5920 |    9 |   1027 |     5 |    20
 2016-05-26 18:06:54 |  6156 |          5697 |   35 |   2199 |     2 |     3
 2016-05-26 18:36:54 |  6183 |          5893 |   64 |   1544 |     5 |    20
 2016-05-26 18:36:54 |  6184 |          5697 |   35 |   2199 |     1 |     2
 2016-05-26 19:36:54 |  6243 |          5659 |   13 |   1623 |     4 |    10
```

You can hit <*space*> to scroll down, and *q* to quit.

Step 4. **Filter rows and columns.** We can query only the *price* and *userid* columns with the following command:

```
select price, userid from buyclicks;
```

The result of this query is:

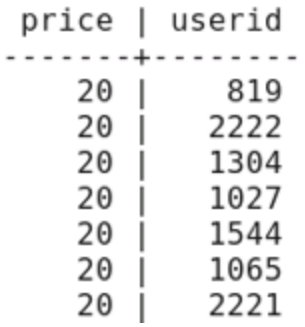


price	userid
3	1300
10	868
20	819
3	121
20	2222
20	1304
20	1027
3	2199
20	1544

We can also query rows that match a specific criteria. For example, the following command queries only rows with a price greater than 10:

```
select price, userid from buyclicks where price > 10;
```

The result is:



price	userid
20	819
20	2222
20	1304
20	1027
20	1544
20	1065
20	2221

Step 5. **Perform aggregate operations.** The SQL language provides many aggregate operations. We can calculate the average price:

```
cloudera=# select avg(price) from buyclicks;
       avg
------------------
 7.26399728537496
(1 row)
```

We can also calculate the total price:

```
cloudera=# select sum(price) from buyclicks;
  sum
-------
 21407
(1 row)
```

The complete list of aggegrate functions for Postgres 8.4 (the version installed on the Cloudera VM) can be found here: https://www.postgresql.org/docs/8.4/static/functions-aggregate.html

Step 6. **Combine two tables.** We combine the contents of two tables by matching or joining on a single column. If we look at the definition of the *adclicks* table:

```
cloudera=# \d adclicks
                Table "public.adclicks"
    Column     |            Type             | Modifiers
---------------+-----------------------------+-----------
 timestamp     | timestamp without time zone | not null
 txid          | integer                     | not null
 usersessionid | integer                     | not null
 teamid        | integer                     | not null
 userid        | integer                     | not null
 adid          | integer                     | not null
 adcategory    | character varying(11)       | not null
```

We see that *adclicks* also has a column named *userid.* The following query combines the *adclicks* and *buyclicks* tables on the *userid* column in both tables:

```
from adclicks join buyclicks on adclicks.userid = buyclicks.userid;

select adid, buyid, adclicks.userid
```

This query shows the columns *adid* and *userid* from the *adclicks* table, and the *buyid* column from the *buyclicks* table. The *from adclicks join buyclicks* denotes that we want to combine these two tables, and *on adclicks.userid = buyclicks.userid* denotes which two columns to use when the tables are combined.

The result of the query is:

```
 adid | buyid | userid
------+-------+--------
    2 |     5 |    611
    2 |     4 |    611
    2 |     4 |    611
    2 |     5 |    611
    2 |     4 |    611
    2 |     1 |    611
   21 |     1 |   1874
   21 |     1 |   1874
   21 |     3 |   1874
   21 |     1 |   1874
   21 |     2 |   1874
```

Enter *q* to quit the Postgres shell.