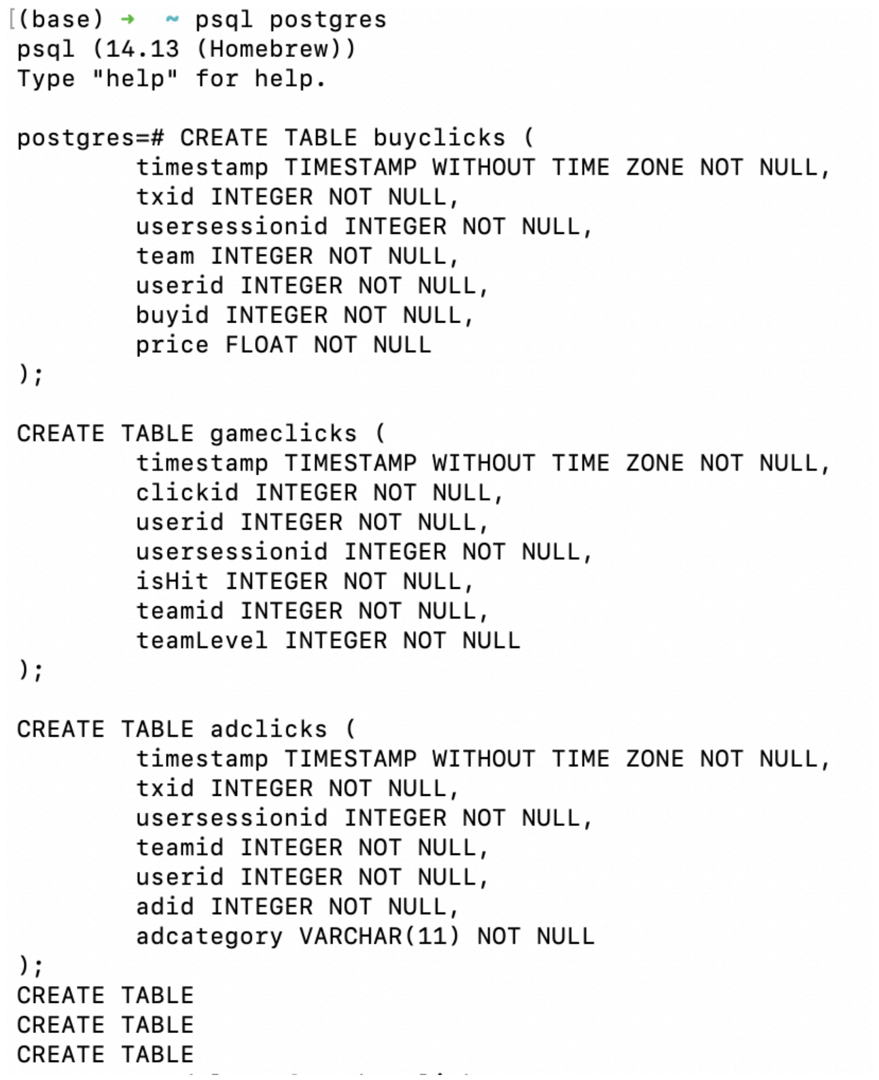
BDA Assignment 4

Arya Sagar
202200673
TY-A
Rn 08

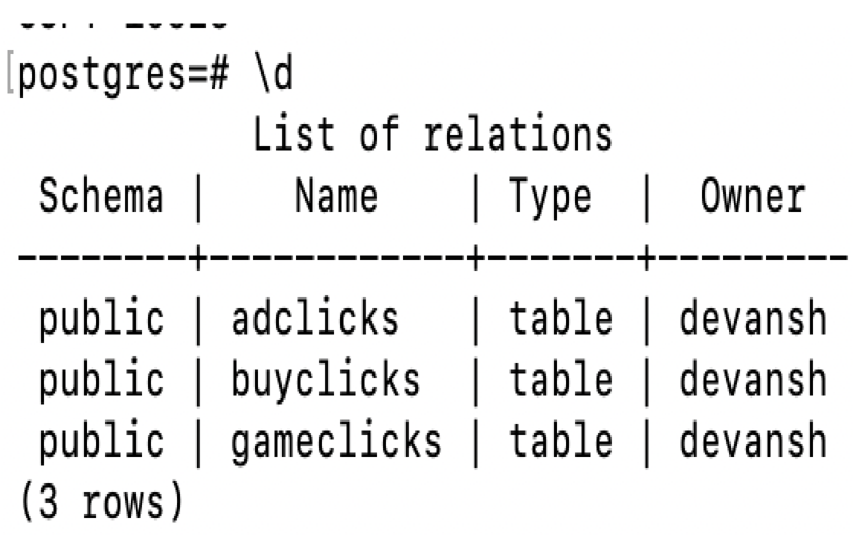
Postgres shell practice queries:

Create Tables;



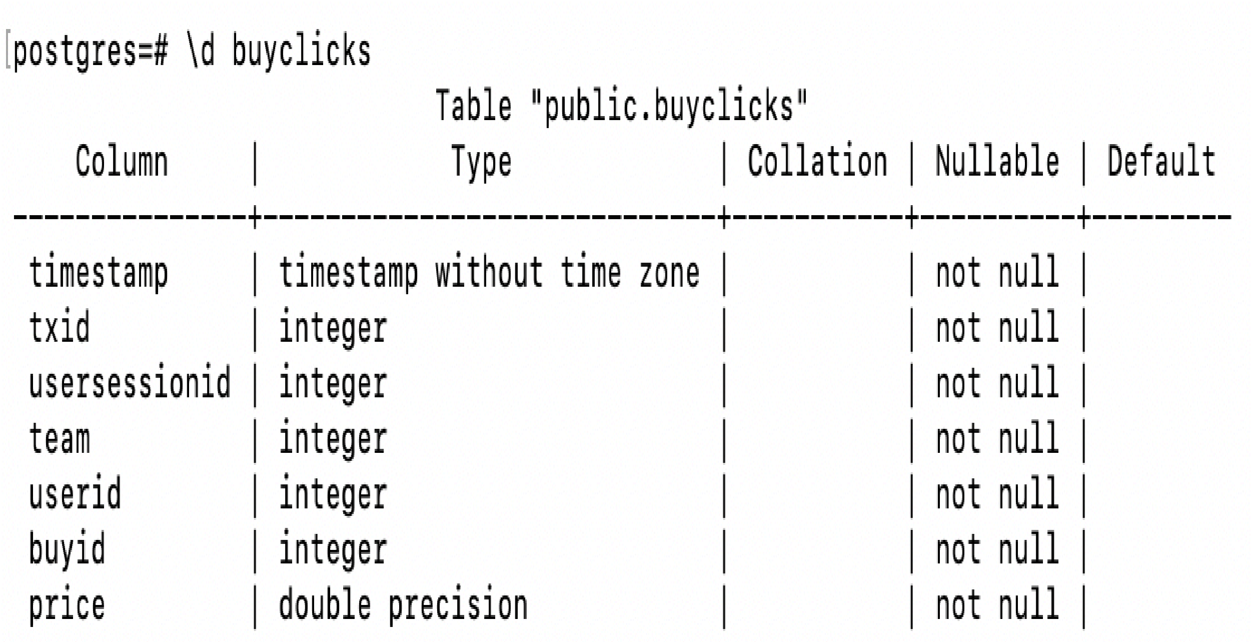
```
[(base) ➜  ~ psql postgres
psql (14.13 (Homebrew))
Type "help" for help.

postgres=# CREATE TABLE buyclicks (
        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 FLOAT NOT NULL
);

CREATE TABLE gameclicks (
        timestamp TIMESTAMP WITHOUT TIME ZONE NOT NULL,
        clickid INTEGER NOT NULL,
        userid INTEGER NOT NULL,
        usersessionid INTEGER NOT NULL,
        isHit INTEGER NOT NULL,
        teamid INTEGER NOT NULL,
        teamLevel INTEGER NOT NULL
);

CREATE TABLE adclicks (
        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 VARCHAR(11) NOT NULL
);
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

View table definitions. : \d



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

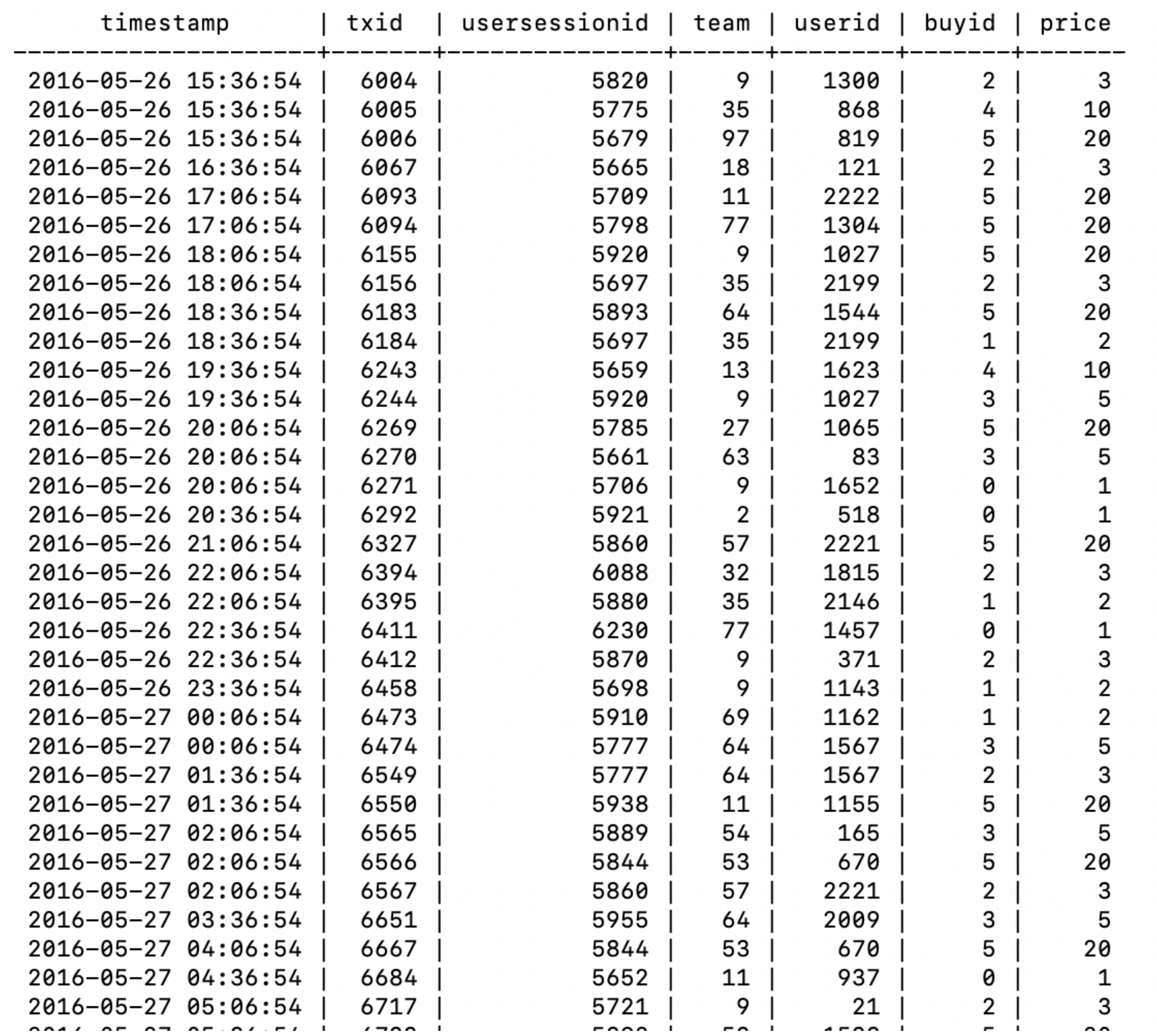
View column definitions : \d buyclicks



```
postgres=# \d buyclicks
                        Table "public.buyclicks"
    Column     |            Type             | Collation | Nullable | Default
---------------+-----------------------------+-----------+----------+---------
 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 |
```

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

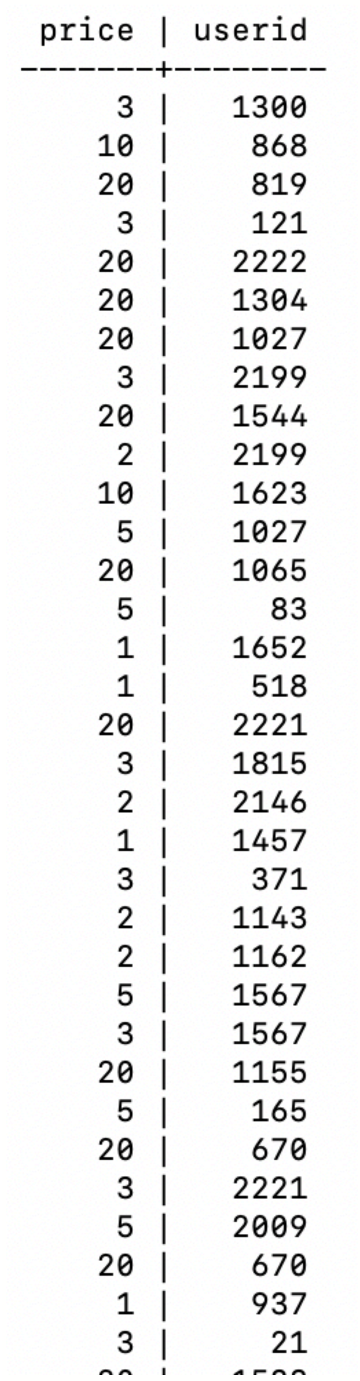
SELECT * FROM buyclicks;



```
      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
 2016-05-26 19:36:54 |  6244 |          5920 |    9 |   1027 |     3 |     5
 2016-05-26 20:06:54 |  6269 |          5785 |   27 |   1065 |     5 |    20
 2016-05-26 20:06:54 |  6270 |          5661 |   63 |     83 |     3 |     5
 2016-05-26 20:06:54 |  6271 |          5706 |    9 |   1652 |     0 |     1
 2016-05-26 20:36:54 |  6292 |          5921 |    2 |    518 |     0 |     1
 2016-05-26 21:06:54 |  6327 |          5860 |   57 |   2221 |     5 |    20
 2016-05-26 22:06:54 |  6394 |          6088 |   32 |   1815 |     2 |     3
 2016-05-26 22:06:54 |  6395 |          5880 |   35 |   2146 |     1 |     2
 2016-05-26 22:36:54 |  6411 |          6230 |   77 |   1457 |     0 |     1
 2016-05-26 22:36:54 |  6412 |          5870 |    9 |    371 |     2 |     3
 2016-05-26 23:36:54 |  6458 |          5698 |    9 |   1143 |     1 |     2
 2016-05-27 00:06:54 |  6473 |          5910 |   69 |   1162 |     1 |     2
 2016-05-27 00:06:54 |  6474 |          5777 |   64 |   1567 |     3 |     5
 2016-05-27 01:36:54 |  6549 |          5777 |   64 |   1567 |     2 |     3
 2016-05-27 01:36:54 |  6550 |          5938 |   11 |   1155 |     5 |    20
 2016-05-27 02:06:54 |  6565 |          5889 |   54 |    165 |     3 |     5
 2016-05-27 02:06:54 |  6566 |          5844 |   53 |    670 |     5 |    20
 2016-05-27 02:06:54 |  6567 |          5860 |   57 |   2221 |     2 |     3
 2016-05-27 03:36:54 |  6651 |          5955 |   64 |   2009 |     3 |     5
 2016-05-27 04:06:54 |  6667 |          5844 |   53 |    670 |     5 |    20
 2016-05-27 04:36:54 |  6684 |          5652 |   11 |    937 |     0 |     1
 2016-05-27 05:06:54 |  6717 |          5721 |    9 |     21 |     2 |     3
```

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

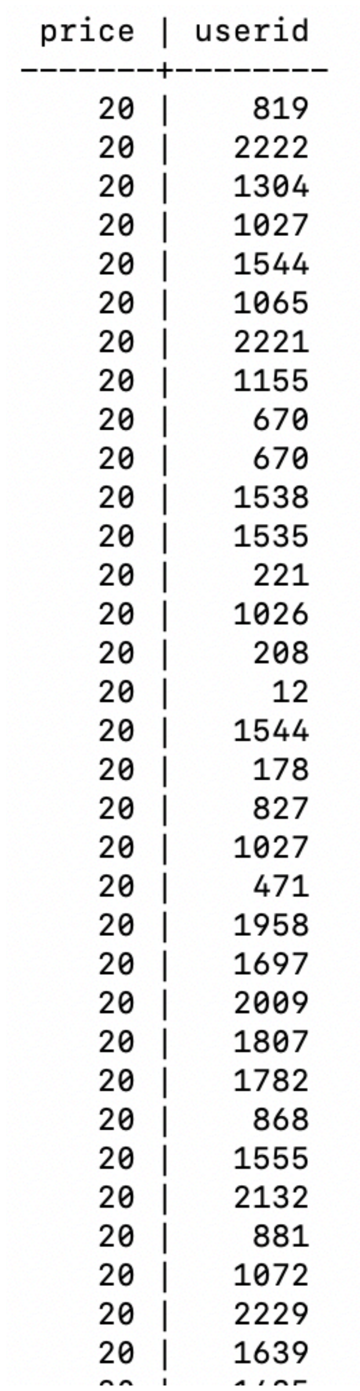
select price, userid from buyclicks;



```
 price | userid
-------+--------
     3 |   1300
    10 |    868
    20 |    819
     3 |    121
    20 |   2222
    20 |   1304
    20 |   1027
     3 |   2199
    20 |   1544
     2 |   2199
    10 |   1623
     5 |   1027
    20 |   1065
     5 |     83
     1 |   1652
     1 |    518
    20 |   2221
     3 |   1815
     2 |   2146
     1 |   1457
     3 |    371
     2 |   1143
     2 |   1162
     5 |   1567
     3 |   1567
    20 |   1155
     5 |    165
    20 |    670
     3 |   2221
     5 |   2009
    20 |    670
     1 |    937
     3 |     21
```

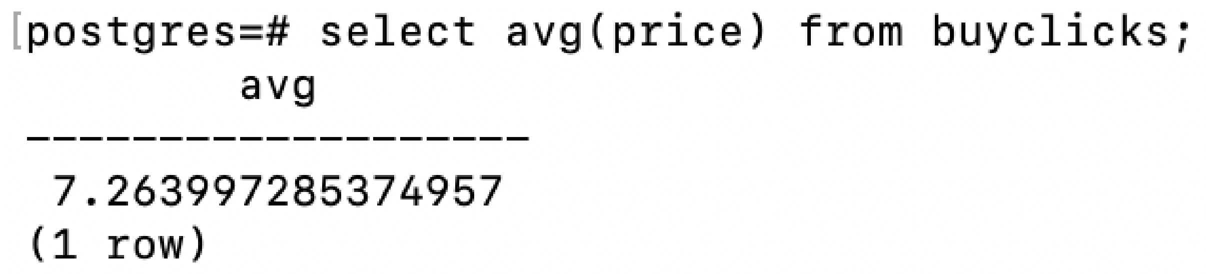
We can also query rows that match a specific criteria. For example, the following command queries only rows with a price greater than 10:

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



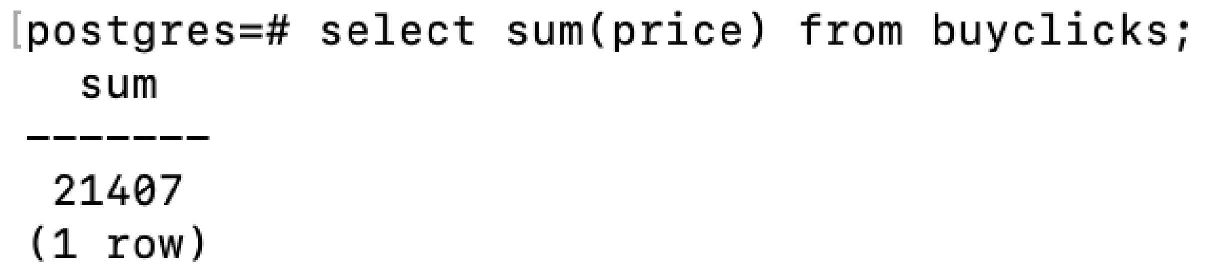
```
 price | userid
-------+--------
    20 |    819
    20 |   2222
    20 |   1304
    20 |   1027
    20 |   1544
    20 |   1065
    20 |   2221
    20 |   1155
    20 |    670
    20 |    670
    20 |   1538
    20 |   1535
    20 |    221
    20 |   1026
    20 |    208
    20 |     12
    20 |   1544
    20 |    178
    20 |    827
    20 |   1027
    20 |    471
    20 |   1958
    20 |   1697
    20 |   2009
    20 |   1807
    20 |   1782
    20 |    868
    20 |   1555
    20 |   2132
    20 |    881
    20 |   1072
    20 |   2229
    20 |   1639
```

Perform aggregate operations. The SQL language provides many aggregate operations. We can calculate the average price



```
[postgres=# select avg(price) from buyclicks;
        avg
-------------------
 7.263997285374957
(1 row)
```

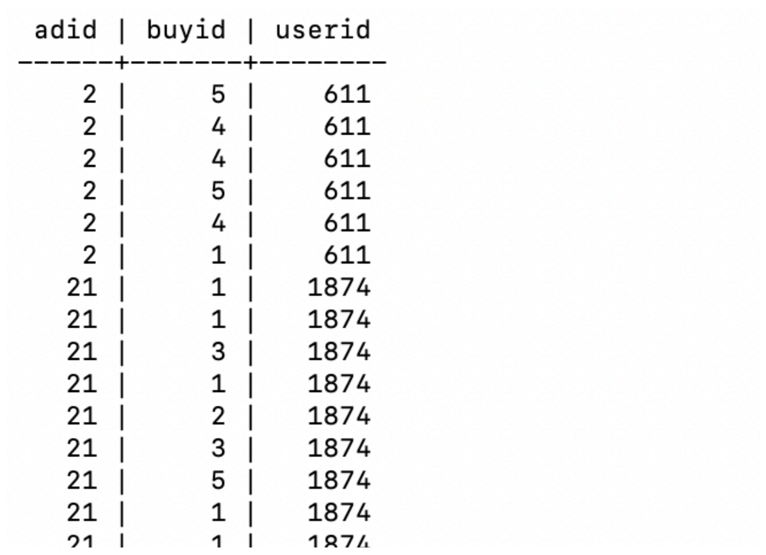
Sum price:



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

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:

select adid, buyid, adclicks.userid
from adclicks join buyclicks on adclicks.userid = buyclicks.userid;



```
 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
   21 |     3 |   1874
   21 |     5 |   1874
   21 |     1 |   1874
   21 |     1 |   1874
```