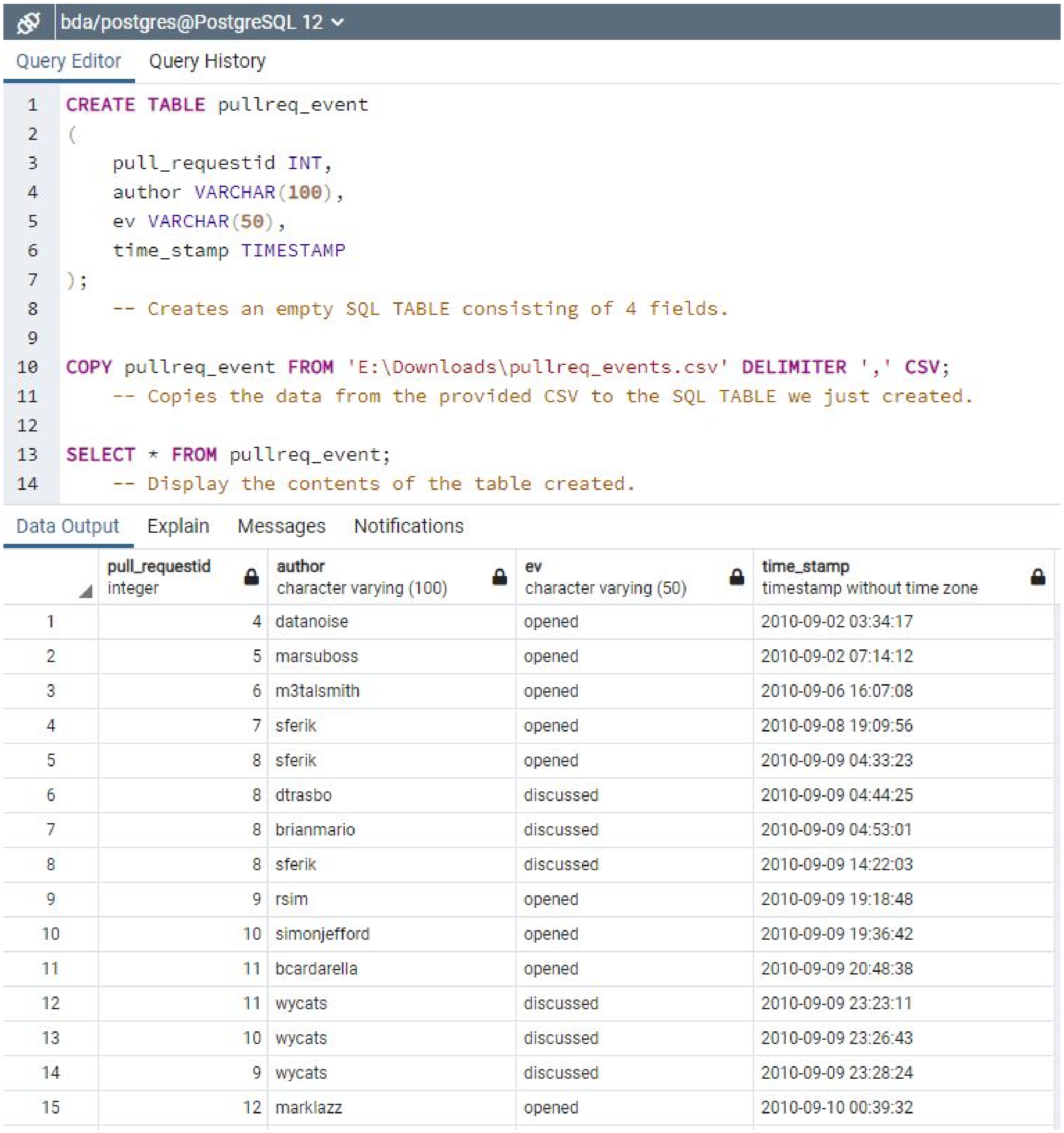
BDA Assignment-1 Report

1. Explain your methodology: approach and reason clearly in the report.

- Consider the following screenshot explaining how the given CSV was imported to an SQL TABLE in a POSTGRESQL RELATIONAL DATABASE.



bda/postgres@PostgreSQL 12

```
CREATE TABLE pullreq_event
(
    pull_requestid INT,
    author VARCHAR(100),
    ev VARCHAR(50),
    time_stamp TIMESTAMP
);
    -- Creates an empty SQL TABLE consisting of 4 fields.

COPY pullreq_event FROM 'E:\Downloads\pullreq_events.csv' DELIMITER ',' CSV;
    -- Copies the data from the provided CSV to the SQL TABLE we just created.

SELECT * FROM pullreq_event;
    -- Display the contents of the table created.
```

	pull_requestid integer	author character varying (100)	ev character varying (50)	time_stamp timestamp without time zone
1	4	datanoise	opened	2010-09-02 03:34:17
2	5	marsuboss	opened	2010-09-02 07:14:12
3	6	m3talsmith	opened	2010-09-06 16:07:08
4	7	sferik	opened	2010-09-08 19:09:56
5	8	sferik	opened	2010-09-09 04:33:23
6	8	dtrasbo	discussed	2010-09-09 04:44:25
7	8	brianmario	discussed	2010-09-09 04:53:01
8	8	sferik	discussed	2010-09-09 14:22:03
9	9	rsim	opened	2010-09-09 19:18:48
10	10	simonjefford	opened	2010-09-09 19:36:42
11	11	bcardarella	opened	2010-09-09 20:48:38
12	11	wycats	discussed	2010-09-09 23:23:11
13	10	wycats	discussed	2010-09-09 23:26:43
14	9	wycats	discussed	2010-09-09 23:28:24
15	12	marklazz	opened	2010-09-10 00:39:32

- In order to perform operations with the TIMESTAMP related data, we had to perform various operations. Some built-in functions were found to be handy such as EXTRACT() and CAST(). The following resources were referred for the same.
    - https://www.postgresqltutorial.com/postgresql-extract/
    - https://www.postgresql.org/docs/9.2/sql-createcast.html
    - https://www.postgresql.org/docs/9.1/functions-datetime.html

2. Add all data analysis steps which you have performed on the dataset.

- The dataset is a collection of the various events that occur in a platform like GitHub associated with the pull requests along with the author responsible for the event as well as the time stamp.

- The data extends from '2010-09-02 03:34:17' till '2018-09-01 14:03:17', which is almost 8 years of events. The dataset consists of 273089 rows and 4 columns. The four columns were copied to an SQL table to perform the required queries.

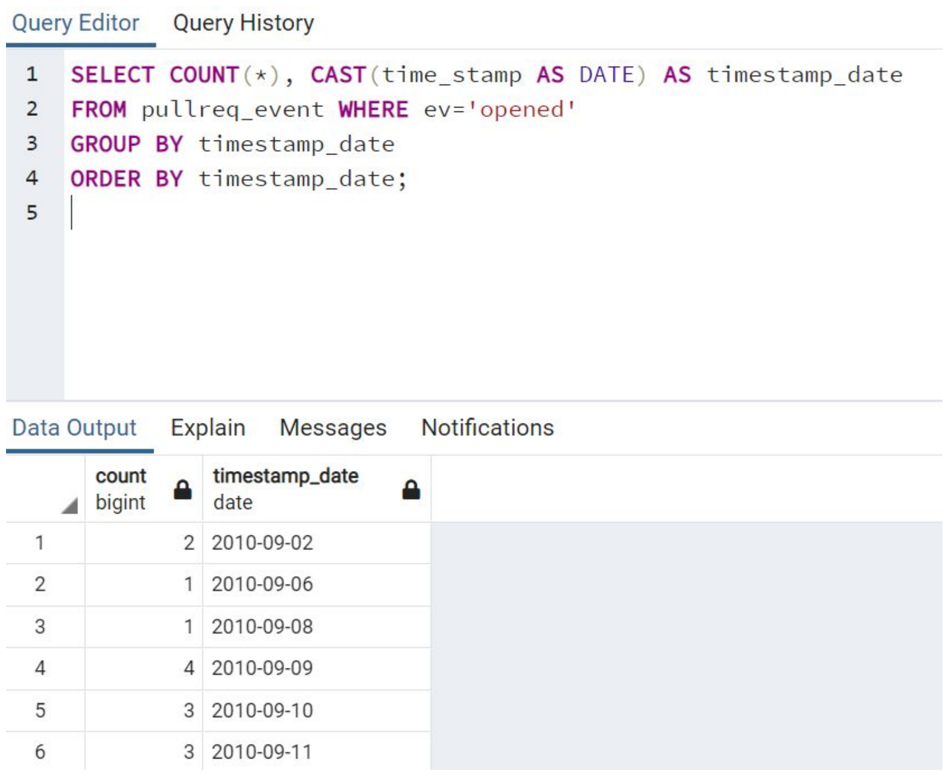
- The SQL TABLE generated consists of 4 fields.
    - Pull_requestid (INTEGER), this is a unique identifier for a pull request.
    - Author (VARCHAR), this is the unique author identifier or the username.
    - Ev (VARCHAR), this represents the event that took place, consists of values out of
        - 'Assigned'
        - 'Closed'
        - 'Discussed'
        - 'Head_ref_deleted'
        - 'Head_ref_restored'
        - 'Mentioned'
        - 'Merged'
        - 'Opened'
        - 'Referenced'
        - 'Reopened'
        - 'Reviewed'
        - 'Subscribed'
        - 'Synchronize'
        - 'Unsubscribed'
    - Time_stamp (TIMESTAMP), this represents the date and time instant that the event was recorded into the database.

3. Report the SQL query and its corresponding output screenshot.

1. Report the number of pull requests

a. “opened” per day.

```
SELECT COUNT(*), CAST(time_stamp AS DATE) AS timestamp_date
FROM pullreq_event WHERE ev='opened'
GROUP BY timestamp_date
ORDER BY timestamp_date
```



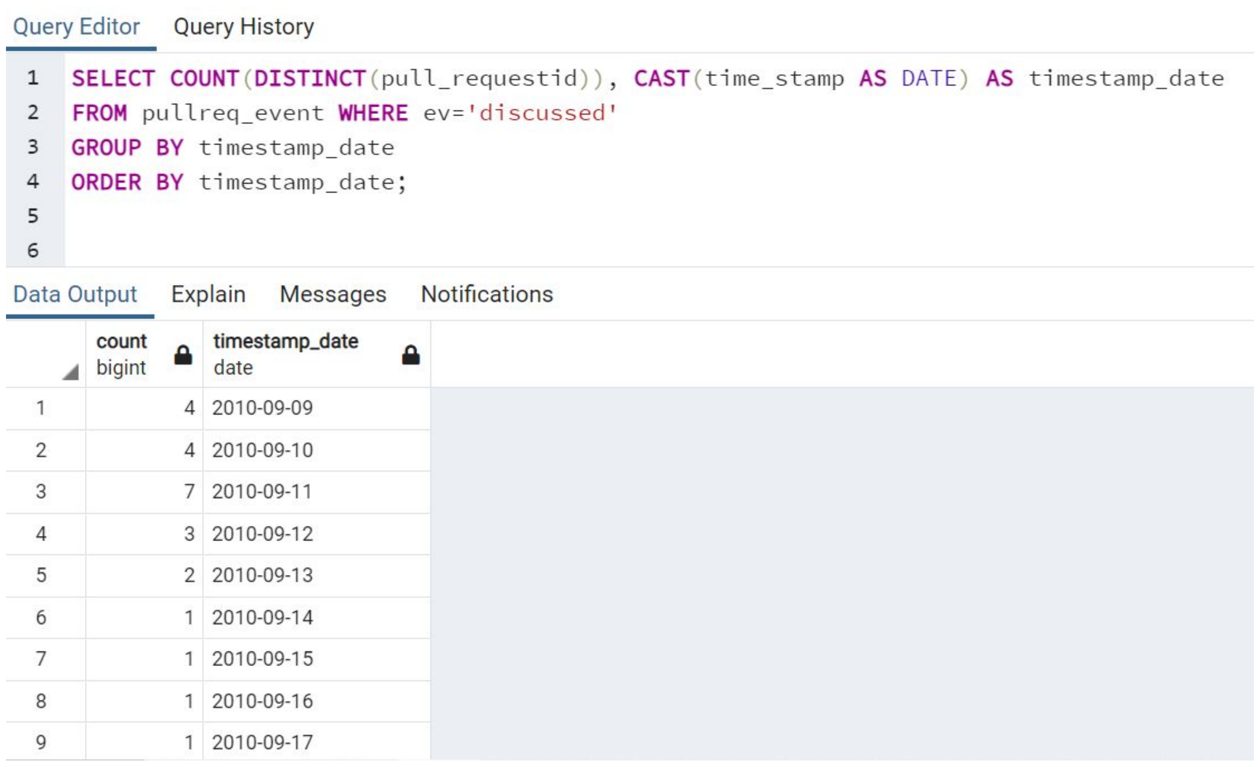
```
SELECT COUNT(*), CAST(time_stamp AS DATE) AS timestamp_date
FROM pullreq_event WHERE ev='opened'
GROUP BY timestamp_date
ORDER BY timestamp_date;
```

	count bigint	timestamp_date date
1	2	2010-09-02
2	1	2010-09-06
3	1	2010-09-08
4	4	2010-09-09
5	3	2010-09-10
6	3	2010-09-11

Explanation: The rows were filtered to collect only ‘opened’ type events using the WHERE clause. Then, the rows were GROUPed BY the date extracted from the timestamp column. Thus, COUNT(*) gives the per day count of opened pull requests.

b. “Discussed” per day.

```
SELECT COUNT(DISTINCT(pull_requestid)), CAST(time_stamp AS DATE) AS timestamp_date
FROM pullreq_event WHERE ev='discussed'
GROUP BY timestamp_date
ORDER BY timestamp_date;
```


Query Editor Query History

```
SELECT COUNT(DISTINCT(pull_requestid)), CAST(time_stamp AS DATE) AS timestamp_date
FROM pullreq_event WHERE ev='discussed'
GROUP BY timestamp_date
ORDER BY timestamp_date;
```

Data Output Explain Messages Notifications

	count bigint	timestamp_date date
1	4	2010-09-09
2	4	2010-09-10
3	7	2010-09-11
4	3	2010-09-12
5	2	2010-09-13
6	1	2010-09-14
7	1	2010-09-15
8	1	2010-09-16
9	1	2010-09-17

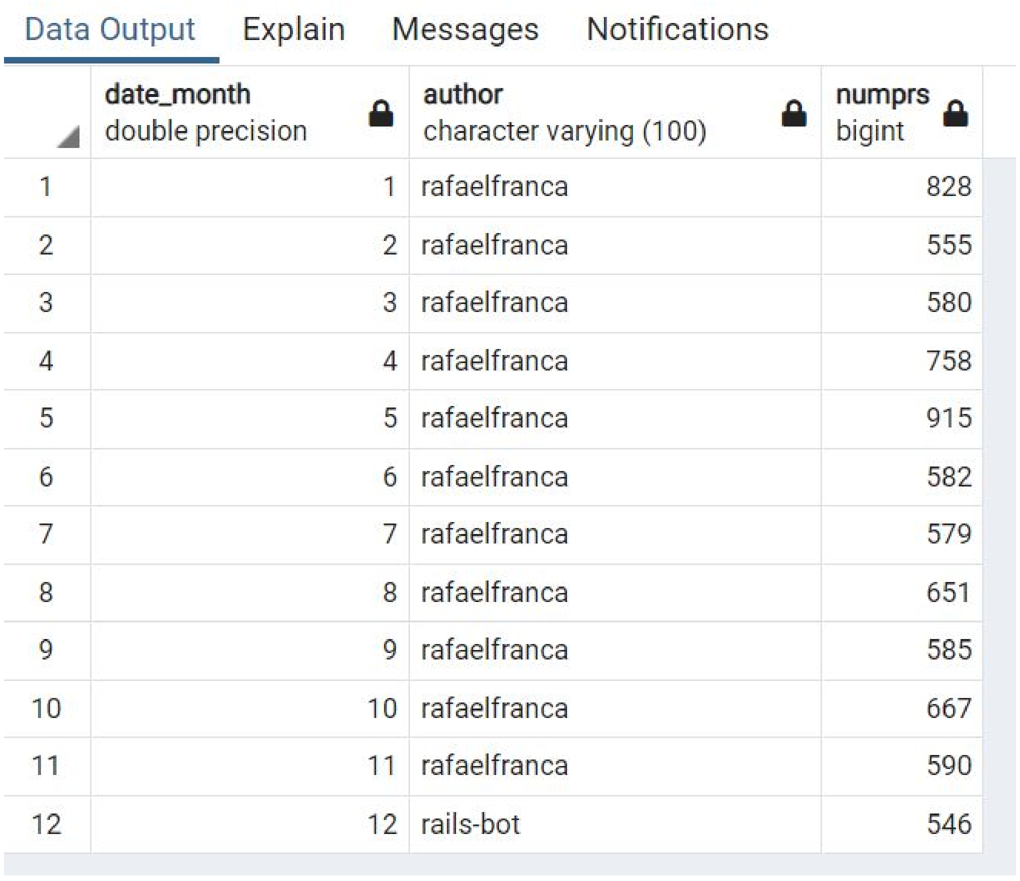
Explanation: In this case, since a pull request could be discussed more than once within the same day, we had to use COUNT(DISTINCT(pull_requestid)). This time the rows were filtered using the WHERE clause to contain only 'discussed' type events. Then, same as 1.a, the rows were GROUPed BY the date extracted from the timestamp data. Thus the COUNT(DISTINCT(pull_requestid)) gives the per day count of unique pull requests discussed.

2. Output the person who has the highest number of comments per month. Assume a month has 30 days. The comments refer to events of the type "discussed". (Points:10)

```
WITH
        tmp as (SELECT EXTRACT(MONTH FROM time_stamp) as date_month,
                author, count(*) as numPRs
                FROM pullreq_event
                WHERE ev = 'discussed'
                GROUP BY date_month, author)

SELECT * FROM tmp
WHERE (date_month, numPRs) IN
(
        SELECT date_month, MAX(numPRs)
        FROM tmp
        GROUP BY date_month
```

);



Data Output | Explain | Messages | Notifications

	date_month double precision	author character varying (100)	numprs bigint
1	1	rafaelfranca	828
2	2	rafaelfranca	555
3	3	rafaelfranca	580
4	4	rafaelfranca	758
5	5	rafaelfranca	915
6	6	rafaelfranca	582
7	7	rafaelfranca	579
8	8	rafaelfranca	651
9	9	rafaelfranca	585
10	10	rafaelfranca	667
11	11	rafaelfranca	590
12	12	rails-bot	546

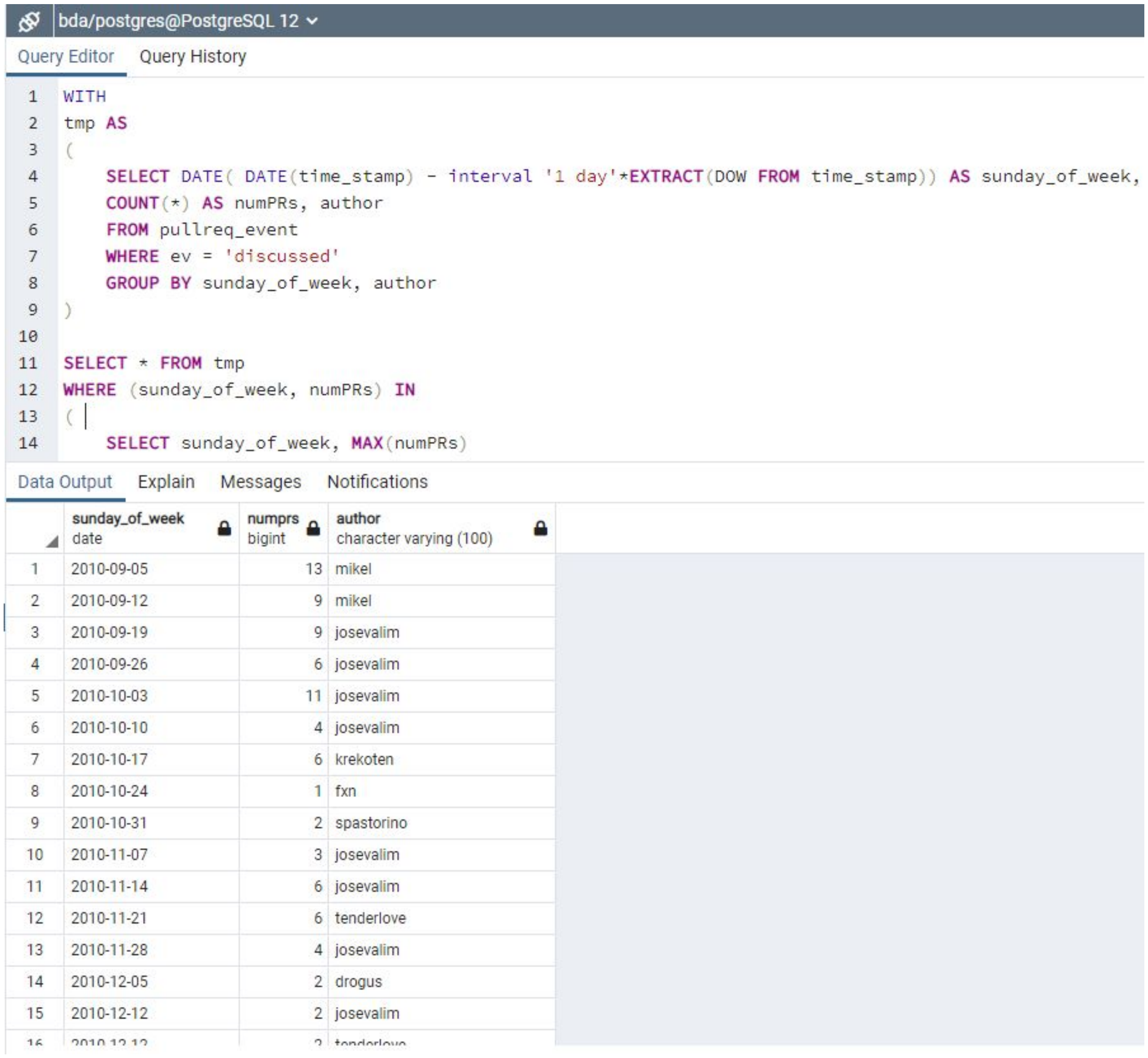
Explanation: First of all we use WITH to create an alias or a temporary table for querying. Using month and author we count the number of comments per month per author. After storing above in a table tmp, we select the maximum number of comments per month, and thus the above result is achieved.

3. Output the person who has the highest number of comments per week. Assume a week has 7 days. The comments refer to events of the type "discussed". (Points: 10)

```
WITH
tmp AS
(
        SELECT DATE( DATE(time_stamp) - interval '1 day'*EXTRACT(DOW FROM
time_stamp)) AS sunday_of_week,
        COUNT(*) AS numPRs, author
        FROM pullreq_event
   WHERE ev = 'discussed'
   GROUP BY sunday_of_week, author
```

)

SELECT * FROM tmp
WHERE (sunday_of_week, numPRs) IN
(
	SELECT sunday_of_week, MAX(numPRs)
	FROM tmp
	GROUP BY sunday_of_week
);



```
WITH
tmp AS
(
    SELECT DATE( DATE(time_stamp) - interval '1 day'*EXTRACT(DOW FROM time_stamp)) AS sunday_of_week,
    COUNT(*) AS numPRs, author
    FROM pullreq_event
    WHERE ev = 'discussed'
    GROUP BY sunday_of_week, author
)

SELECT * FROM tmp
WHERE (sunday_of_week, numPRs) IN
(
    SELECT sunday_of_week, MAX(numPRs)
```

	sunday_of_week date	numprs bigint	author character varying (100)
1	2010-09-05	13	mikel
2	2010-09-12	9	mikel
3	2010-09-19	9	josevalim
4	2010-09-26	6	josevalim
5	2010-10-03	11	josevalim
6	2010-10-10	4	josevalim
7	2010-10-17	6	krekoten
8	2010-10-24	1	fxn
9	2010-10-31	2	spastorino
10	2010-11-07	3	josevalim
11	2010-11-14	6	josevalim
12	2010-11-21	6	tenderlove
13	2010-11-28	4	josevalim
14	2010-12-05	2	drogus
15	2010-12-12	2	josevalim
16	2010-12-12	2	tenderlove

Explanation:

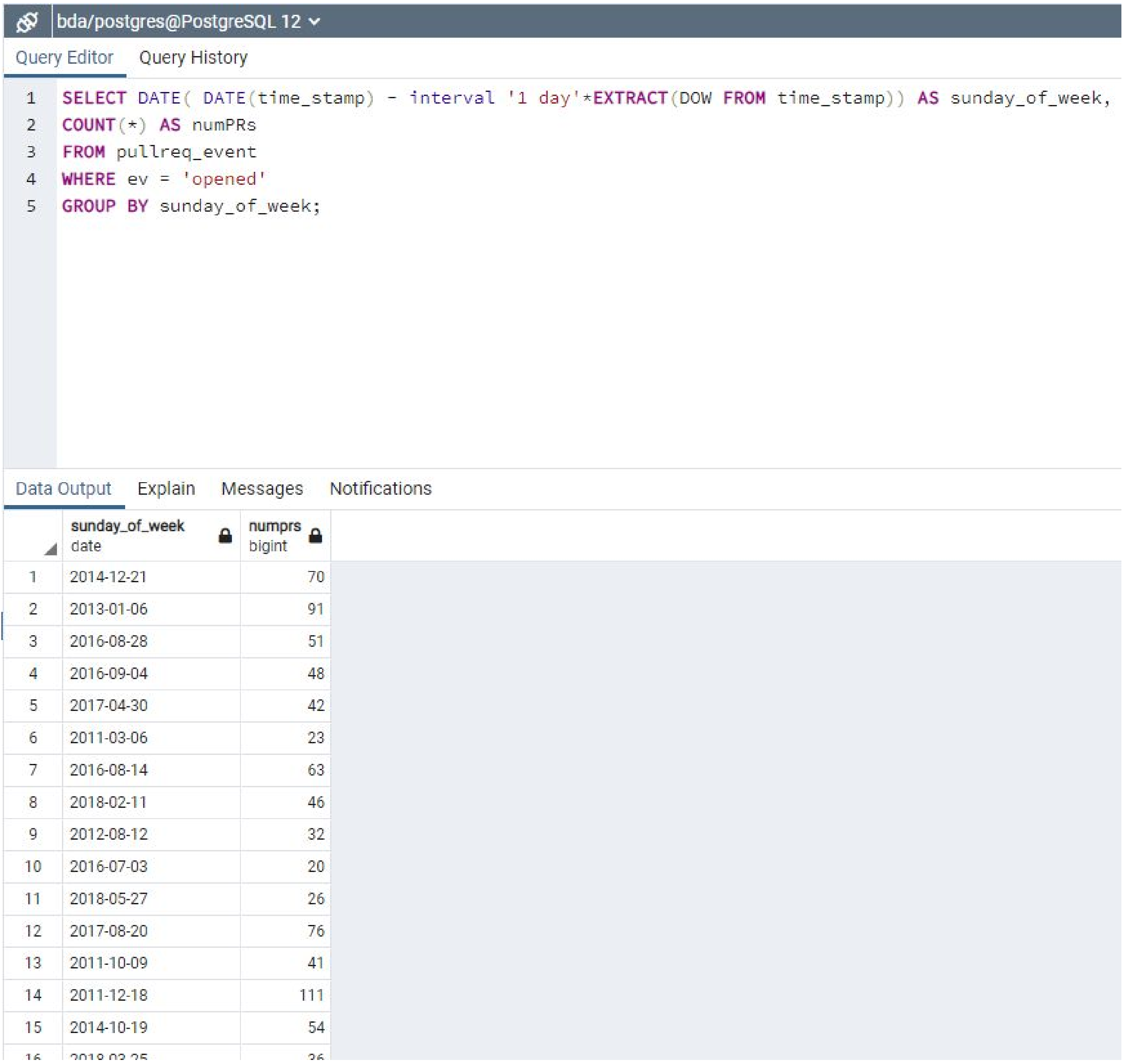
- DATE(DATE(time_stamp) - interval '1 day'*EXTRACT(DOW FROM time_stamp)) AS sunday_of_week

- This line calculates the date for the sunday of the corresponding week of the date from time_stamp field of the tuples. Basically, we subtract an interval of k days from the time stamp, where k ranges from 0 to 6 depending on the day of the week. 0 represents the Sunday while 6 represents the Saturday.

After filtering the rows, using WHERE clause, to reflect only the entries of the type ‘discussed’, we GROUP them BY author name and the newly created field ‘sunday_of_week’. We save this table as a temporary relation TMP. Next, we output all the tuples from the relation TMP which have the maximum count of the ‘discussed’ event for the week GROUPed BY the sunday_of_week. Additionally, we output the author name from the tuple.

4. Output the number of Pull Requests (PRs) opened per week. (Points:10)

```
SELECT DATE( DATE(time_stamp) - interval '1 day'*EXTRACT(DOW FROM time_stamp))
AS sunday_of_week,
COUNT(*) AS numPRs
FROM pullreq_event
WHERE ev = 'opened'
GROUP BY sunday_of_week;
```



```sql
SELECT DATE( DATE(time_stamp) - interval '1 day'*EXTRACT(DOW FROM time_stamp)) AS sunday_of_week,
COUNT(*) AS numPRs
FROM pullreq_event
WHERE ev = 'opened'
GROUP BY sunday_of_week;
```

	sunday_of_week date	numprs bigint
1	2014-12-21	70
2	2013-01-06	91
3	2016-08-28	51
4	2016-09-04	48
5	2017-04-30	42
6	2011-03-06	23
7	2016-08-14	63
8	2018-02-11	46
9	2012-08-12	32
10	2016-07-03	20
11	2018-05-27	26
12	2017-08-20	76
13	2011-10-09	41
14	2011-12-18	111
15	2014-10-19	54
16	2018-03-25	36

<u>Explanation</u>:

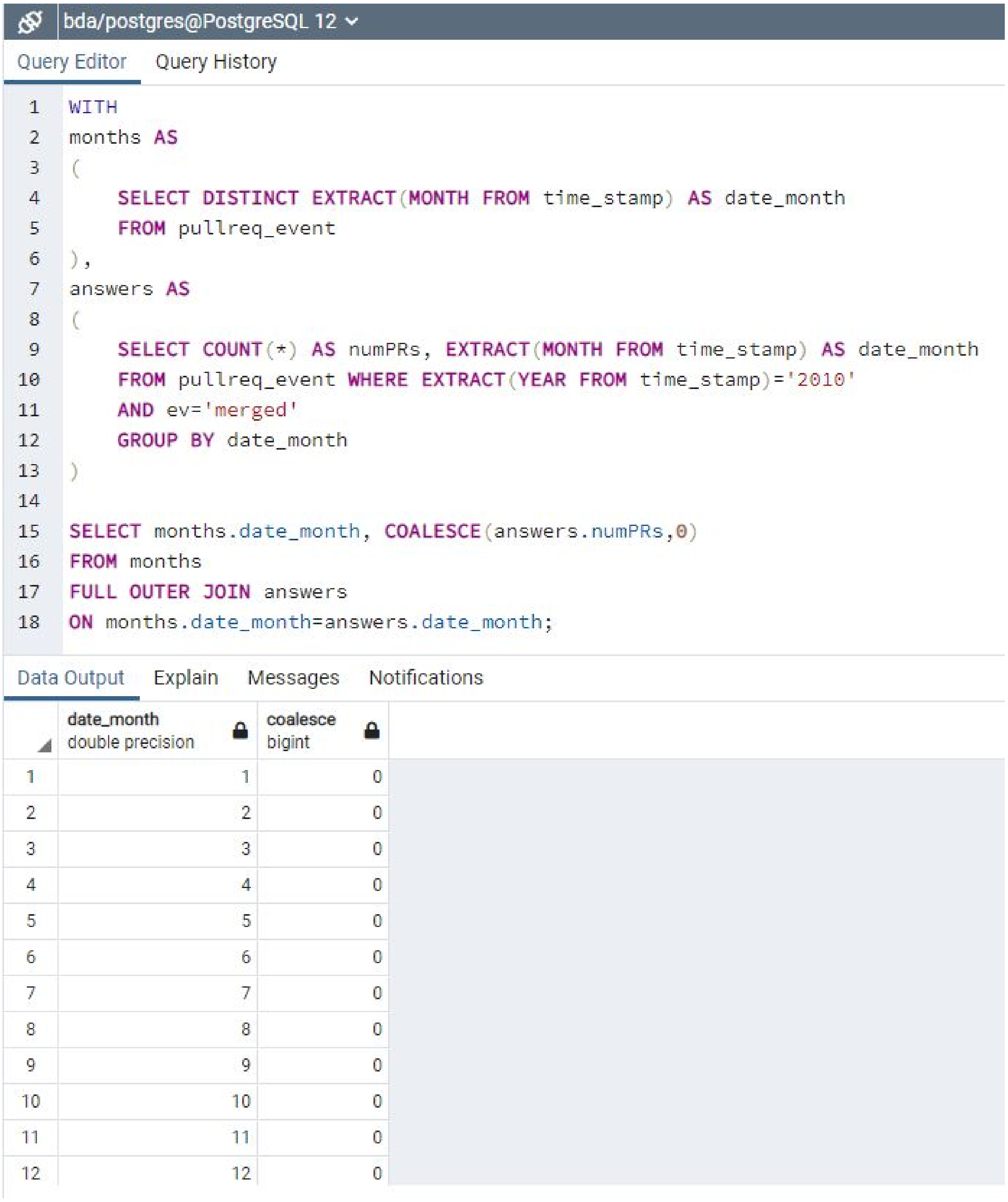
- DATE(DATE(time_stamp) - interval '1 day'*EXTRACT(DOW FROM time_stamp)) AS sunday_of_week
- This line calculates the date for the sunday of the corresponding week of the date from time_stamp field of the tuples. Basically, we subtract an interval of k days from the time stamp, where k ranges from 0 to 6 depending on the day of the week. 0 represents the Sunday while 6 represents the Saturday.

After filtering the tuples to contain only 'opened' type events using WHERE, we GROUP the tuples BY the newly created field 'sunday_of_week', and thus COUNT(*) gives us the total count of pull requests opened per week.

5. Count the number of Pull Requests merged per month in the year 2010. (Points:10)

```
WITH
months AS
(
	SELECT DISTINCT EXTRACT(MONTH FROM time_stamp) AS date_month
	FROM pullreq_event
),
answers AS
(
	SELECT COUNT(*) AS numPRs, EXTRACT(MONTH FROM time_stamp) AS date_month
	FROM pullreq_event WHERE EXTRACT(YEAR FROM time_stamp)='2010'
	AND ev='merged'
	GROUP BY date_month
)

SELECT months.date_month, COALESCE(answers.numPRs,0)
FROM months
FULL OUTER JOIN answers
ON months.date_month=answers.date_month;
```



bda/postgres@PostgreSQL 12

Query Editor   Query History

```
WITH
months AS
(
    SELECT DISTINCT EXTRACT(MONTH FROM time_stamp) AS date_month
    FROM pullreq_event
),
answers AS
(
    SELECT COUNT(*) AS numPRs, EXTRACT(MONTH FROM time_stamp) AS date_month
    FROM pullreq_event WHERE EXTRACT(YEAR FROM time_stamp)='2010'
    AND ev='merged'
    GROUP BY date_month
)

SELECT months.date_month, COALESCE(answers.numPRs,0)
FROM months
FULL OUTER JOIN answers
ON months.date_month=answers.date_month;
```

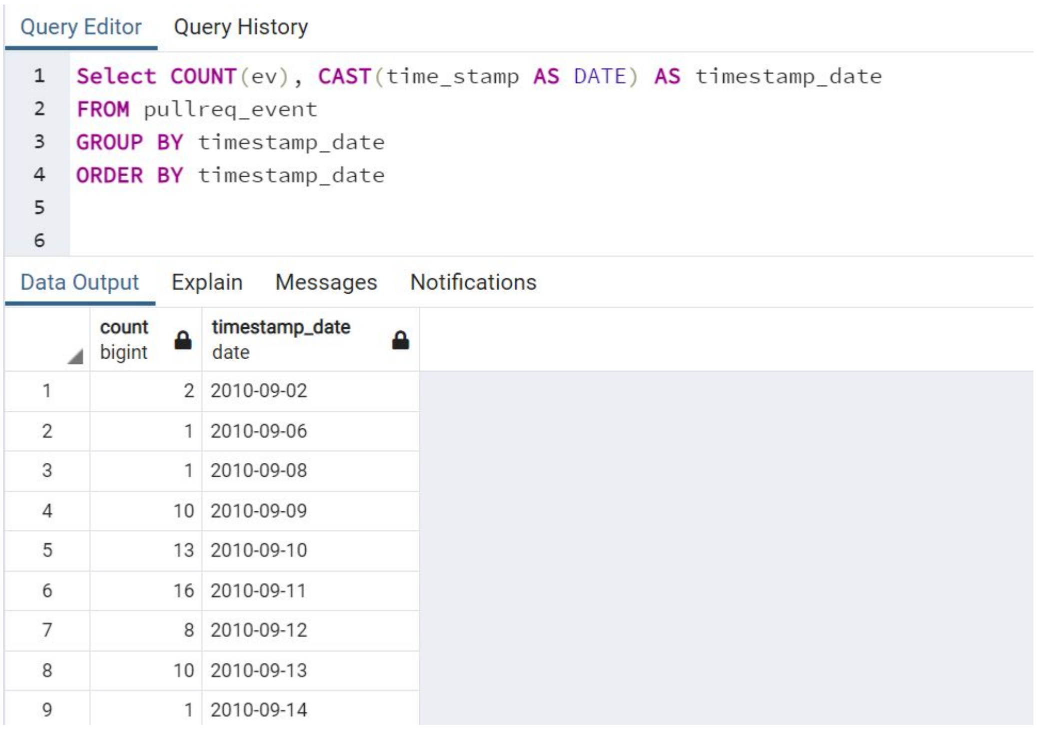
Data Output   Explain   Messages   Notifications

	date_month double precision	coalesce bigint
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0
7	7	0
8	8	0
9	9	0
10	10	0
11	11	0
12	12	0

Explanation: First of all, we create a table having 12 distinct months as table months. Next we fetch the answer of the query, that is merged in 2010 grouped by per month using Extract and Year functions. Then using FULL OUTER JOIN we join on the months and using COALESCE function if answer is NULL, it is reported as zero.

6. Per day report the total number of events. (Points: 5)

```
Select COUNT(ev), CAST(time_stamp AS DATE) AS timestamp_date
FROM pullreq_event
GROUP BY timestamp_date
ORDER BY timestamp_date
```



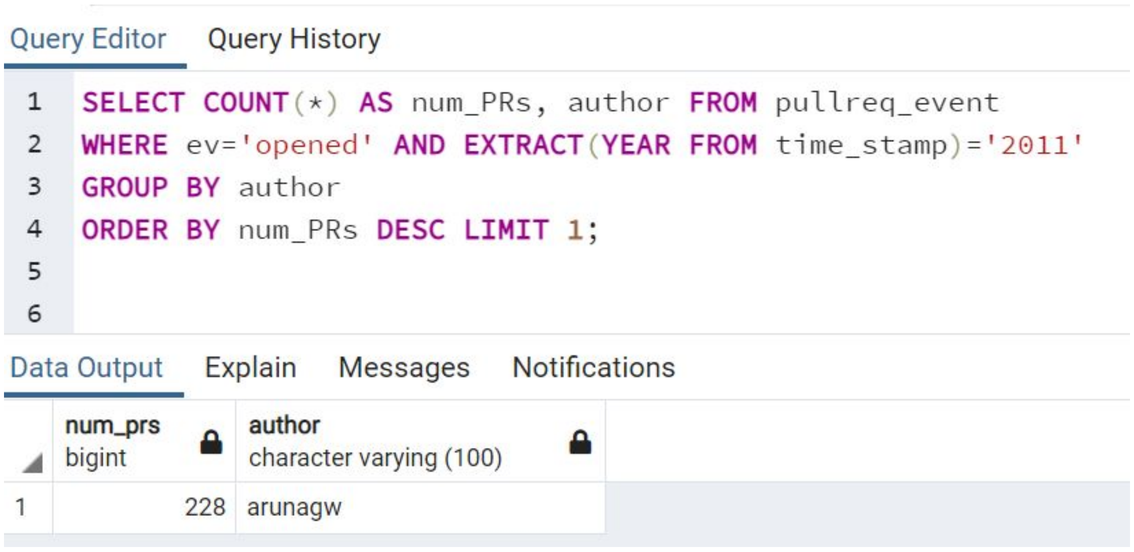
```
Select COUNT(ev), CAST(time_stamp AS DATE) AS timestamp_date
FROM pullreq_event
GROUP BY timestamp_date
ORDER BY timestamp_date
```

	count bigint	timestamp_date date
1	2	2010-09-02
2	1	2010-09-06
3	1	2010-09-08
4	10	2010-09-09
5	13	2010-09-10
6	16	2010-09-11
7	8	2010-09-12
8	10	2010-09-13
9	1	2010-09-14

<u>Explanation</u>: Here WHERE clause is not needed since we will consider all the rows. Again we extract the DATE from our timestamp field and GROUP the rows BY this DATE. Thus COUNT(*) shall give the count of dataset entries per day which is nothing but the number of events that took place that day.

7. Report the user who has opened the highest number of PRs in the year 2011. (Points: 15)

```
SELECT COUNT(*) AS num_PRs, author FROM pullreq_event
WHERE ev='opened' AND EXTRACT(YEAR FROM time_stamp)='2011'
GROUP BY author
ORDER BY num_PRs DESC LIMIT 1;
```



```sql
SELECT COUNT(*) AS num_PRs, author FROM pullreq_event
WHERE ev='opened' AND EXTRACT(YEAR FROM time_stamp)='2011'
GROUP BY author
ORDER BY num_PRs DESC LIMIT 1;
```

	num_prs bigint	author character varying (100)
1	228	arunagw

<u>Explanation</u>: Here we filtered the rows to contain only 'opened' type events from the year 2011 using a WHERE clause. Next, we GROUPed the filtered rows BY the author. Thus COUNT(*) gives per AUTHOR count of pull requests opened in the year 2011. Now to find the one with the maximum count, we sorted the results in descending order using ORDER BY num_prs DESC and show the 1st entry as the result using LIMIT 1.

4. Make a section "Learning", which describes your learning in doing this assignment.

- Set up PostgreSQL and pgAdmin
- Syntax of PostgreSQL
- Functions such as Extract, Month, Year, Week from TimeStamp datatype
- JOINS such as self join, full outer join.
- WITH operator which is used to create ALIAS or temporary tables for querying in Subsequent parts.
- Creating Alias and querying on them
- Group By operations can take multiple parameters
- Aggregate functions such as Order by, Group by, LIMIT