As with practically any implementation, the tasks outlined in this assignment can be solved in many ways. Nonetheless the simpler, faster and more elegant the solution is strongly preferred. Please note some more advanced tasks here could be best solved using Window Analytical Functions.

In thaim to make your code structured and formatted in a clear, readable way.

Prerequisite Step 1: Familiarise yourself and learn about the concept of SQL window functions on PostgreSQL.
Here are some tutorials to get you started:

http://www.postgresqltutorial.com/postgresql-window-function
https://www.postgresql.org/docs/9.6/static/functions-window.html
https://robots.thoughtbot.com/postgres-window-functions
https://www.periscopedata.com/blog/window-functions-by-example.htm

Prerequisite Step 2: Install PostgreSQL on your computer

Task 1: Load Dataset into PostgreSQL

We have a small data set of a mobile company available for download here. This extract contains the history of top-up dates and the amounts. Load this dataset into the `topups` table. Would you able to load this input file as-is (without any manual modifications to it) using the Foreign Table PostgreSQL feature rather than COPY command?

seq	id_user	topup_date	topup_value
⋮			
68	3	2017-06-28	15
67	5	2017-06-26	5
66	2	2017-06-20	5
65	1	2017-06-20	15
64	4	2017-06-17	25
63	5	2017-06-15	20
62	5	2017-06-12	5
61	4	2017-06-07	10
60	4	2017-06-05	5
59	2	2017-06-05	10
58	5	2017-06-02	10
57	5	2017-05-31	25
56	4	2017-05-30	25
55	3	2017-05-25	5
54	1	2017-05-22	10
⋮			

Output: (1) DDL/DML script for table/data

Task 2: Simple Aggregations. Print out the list of user IDs and the total of all top-ups done by them **but ONLY** for the users that had at least one topup ever done by the amount of €15 exactly. Can this be solved in a single SELECT statement, without Window Aggregates or subqueries (nested SELECTs)?

Output: The SQL query

Task 3: Row Sequencing. Show the **5 (but not more) rows containing most recent top-ups per user.** In case of more top-ups done within a day, print those with higher amounts first.

Output: The SQL query

Task 4: Row Sequencing. Show the **5 largest top ups** done per user. Aim to limit the result to just 5 rows per user but allow for more if the immediately following topups (6th, 7th etc...) still have the same value (as in 5th).

Output: The SQL query

Task 5: Inter-row calculations

Using the window functions enrich the original set (create new, derived table) to include extra columns as per description below:

prv_topup_dt - previous topup date of the same user
days_since - number of days since previous topup by user
promo_ind - Y/N flag. Put Y for top-ups of €20 or more, otherwise N.
previous_qual_topup_dt - the date of previous topup of €20 or more done by the same user
to_1st_ratio - (bonus) Y/X fraction value where Y is the current topup value and X is the amount of the first ever topup done by the user.

The table below illustrates the format and the logic of the output. CTAS syntax preferred.

id_user	date	topup_value	prv_topup_dt	days_since	promo_ind	previous_qual_topup_dt	to_1st_ratio
1	2017-06-20	15	2017-05-20	31	N	2017-04-20	0.6
1	2017-05-20	10	2017-04-20	30	N	2017-04-20	0.4
1	2017-04-20	20	2017-03-20	31	Y	2017-03-20	0.8
1	2017-03-20	20	2017-02-20	28	Y	NULL	0.8
1	2017-02-20	15	**NULL**	NULL	N	NULL	0.6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2	2017-06-20	5	2017-06-05	15	N	2017-01-15	1
2	2017-06-05	10	2017-04-22	44	N	2017-01-15	2
2	2017-04-22	10	2017-03-30	23	N	2017-01-15	2
2	2017-03-30	10	2017-03-15	15	N	2017-01-15	2
2	2017-03-15	15	2017-02-10	33	N	2017-01-15	3
2	2017-02-10	10	2017-02-10	0	N	2017-01-15	2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Output: The SQL query. Preferable use of as little subselects as possible (graded higher)

Bonus Task 6: Merging periods

The mobile operator runs a promotion. **If you top up by at least €20 you get a free credit for 28 calendar days immediately following the day of that top up**. Print out the list of consolidated periods when users were eligible to make free calls. Include initial eligibility date and the date when the free credit effectively ends.

Consider that with regular top-ups the free credit period may be effectively prolonged multiple times. For instance for id_user=4 the first promotion period started on 2016-12-20 and with the four qualifying topups being done *on time* it was effectively extended up until 2017-04-12:

id_user	promo_start	promo_end
⋮	⋮	⋮
4	2016-12-20	2017-04-12
4	2017-05-30	2017-07-15
⋮	⋮	⋮

Hint: Review the functions used in the previous task. You may use its outcome and process further.
Output: The SQL Query. If you use any subselects please stick them with a comment explaining logic.

Task 7: Narrative

Write up your findings in a short report with an introduction on window functions, your core findings, and a conclusion.