SQL запросы к БД по событиям

Описание задачи

Заказчиком предоставлен скрипт для генерации датасета, который описывает действия пользователей на веб-сайте за период с '2022-09-12' по '2022-09-25'.

Необходимо:

- Применить скрипт для генерации датасета.
- Написать 3 различных SQL-запроса, которые рассчитывают количество уникальных пользователей (user_id), которые совершали событие click_go_to_sberbank и событие click_link_part за период с '2022-09-12' по '2022-09-22', включительно.
- Запросы должны различаться логически.

Загрузка данных

Скрипт выполнен в программе pgAdmin4, сгенерированный датасет сохранен на google drive. Загрузим данные в ноутбук.

In [1]:
```
! gdown --id 1cG3kaaqHAxMGP9PPHgQewJEUspQGDFgR
```

```
/usr/local/lib/python3.8/dist-packages/gdown/cli.py:127: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You don't need to pass it anymore to use a file ID.
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=1cG3kaaqHAxMGP9PPHgQewJEUspQGDFgR
To: /content/events.csv
100% 404k/404k [00:00<00:00, 78.0MB/s]
```

Импорт библиотеки pandas

In [2]:
```
import pandas as pd
```

Чтение файла с данными

In [3]:
```
df = pd.read_csv('/content/events.csv', low_memory=False)
```

Изучение общей информации о данных

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4663 entries, 0 to 4662
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   timestamp     4663 non-null   int64 
 1   user_id       4663 non-null   object
 2   url           4663 non-null   object
 3   event_action  3827 non-null   object
dtypes: int64(1), object(3)
memory usage: 145.8+ KB
```

In [5]:
```
df.head()
```

Out[5]:

	timestamp	user_id	url	event_action
0	1663756768784	1638363559642215448	87A5824E5ED8514B0C28B2368C224F90	informing_leads
1	1663756762532	1638363559642215448	87A5824E5ED8514B0C28B2368C224F90	scroll_article
2	1663756771101	1638363559642215448	87A5824E5ED8514B0C28B2368C224F90	scroll_article
3	1663764211532	1638363559642215448	8DB1F778F36CF51427B3A36184FEE096	informing_leads
4	1663765759516	1637300269699117760	87A5824E5ED8514B0C28B2368C224F90	NaN

Преобразование формата даты

Дата в формате unix time в миллисекундах, переведем дату в формат datetime.

In [6]:
```
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
```

In [7]:
```
df.dtypes
```

Out[7]:
```
timestamp       datetime64[ns]
user_id                 object
url                     object
event_action            object
dtype: object
```

In [8]:
```
df.head()
```

Out[8]:

	timestamp	user_id	url	event_action
0	2022-09-21 10:39:28.784	1638363559642215448	87A5824E5ED8514B0C28B2368C224F90	informing_leads
1	2022-09-21 10:39:22.532	1638363559642215448	87A5824E5ED8514B0C28B2368C224F90	scroll_article
2	2022-09-21 10:39:31.101	1638363559642215448	87A5824E5ED8514B0C28B2368C224F90	scroll_article
3	2022-09-21 12:43:31.532	1638363559642215448	8DB1F778F36CF51427B3A36184FEE096	informing_leads
4	2022-09-21 13:09:19.516	1637300269699117760	87A5824E5ED8514B0C28B2368C224F90	NaN

Подключение к СУБД

In [9]:
```
# Install postgresql server
!sudo apt-get -y -qq update
!sudo apt-get -y -qq install postgresql
!sudo service postgresql start

# Setup a password `postgres` for username `postgres`
!sudo -u postgres psql -U postgres -c "ALTER USER postgres PASSWORD 'postgres';"
```

```
 * Starting PostgreSQL 10 database server
   ...done.
ALTER ROLE
```

In [10]:
```
from sqlalchemy import create_engine
con = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/postgres')
```

In [11]:
```
def select(sql):
  return pd.read_sql(sql,con)
```

In [12]:
```
import csv
from io import StringIO

def psql_insert_copy(table, conn, keys, data_iter):
    # gets a DBAPI connection that can provide a cursor
    dbapi_conn = conn.connection
    with dbapi_conn.cursor() as cur:
        s_buf = StringIO()
        writer = csv.writer(s_buf)
        writer.writerows(data_iter)
        s_buf.seek(0)

        columns = ', '.join('"{}"'.format(k) for k in keys)
        if table.schema:
            table_name = '{}.{}'.format(table.schema, table.name)
        else:
            table_name = table.name

        sql = 'COPY {} ({}) FROM STDIN WITH CSV'.format(
            table_name, columns)
        cur.copy_expert(sql=sql, file=s_buf)
```

Запись таблицы в базу данных SQL

In [13]:
```
df.to_sql('df', con, index=False, if_exists='replace', method=psql_insert_copy)
```

3 SQL-запроса

Напишем 3 различных SQL-запроса, которые рассчитывают количество уникальных пользователей (user_id), совершивших событие click_go_to_sberbank и событие click_link_part за период с 2022-09-12' по '2022-09-22', включительно.

Определения используемых операторов

- SELECT, FROM - запрос в базу данных
- WITH - оператор для создания временной таблицы
- AS - оператор для создания псевдонимов
- WHERE - оператор для определения среза данных
- GROUP BY - деление данных на группы по значениям полей
- HAVING - срез данных после группировки
- DISTINCT - уникальные значения
- COUNT - расчет количества
- IN - фильтр данных по значениям, перечисленным через запятую
- OR, AND - логические операторы, причем AND имеет приоритет, поэтому применяем при необходимости скобки
- BETWEEN - задать диапазон
- LIKE - фильтр по содержанию фрагмента в данных

Для решения задачи используем подзапрос либо временную таблицу.

Запрос 1

In [14]:
```
sql = """SELECT COUNT (count_go_link)
         FROM
              (SELECT user_id, COUNT(DISTINCT event_action) AS count_go_link
               FROM df
               WHERE event_action IN ('click_go_to_sberbank', 'click_link_part')
                AND timestamp BETWEEN '2022-09-11' AND '2022-09-23'
               GROUP BY user_id
               HAVING COUNT(DISTINCT event_action) = 2) AS filter
          """
select(sql)
```

Out[14]:

	count
0	3

Запрос 2

In [15]:
```
sql = """WITH
         filter AS (SELECT user_id, COUNT(DISTINCT event_action) AS count_go_link
                    FROM df
                    WHERE (event_action = 'click_go_to_sberbank' OR event_action = 'click_link_part')
                     AND timestamp>='2022-09-12' AND timestamp<'2022-09-23'
                    GROUP BY user_id
                    HAVING COUNT(DISTINCT event_action) = 2)

         SELECT COUNT (count_go_link)
         FROM filter
         """
select(sql)
```

Out[15]:

	count
0	3

Запрос 3

In [16]:
```
sql = """SELECT COUNT (count_go_link)
         FROM
              (SELECT user_id, COUNT(DISTINCT event_action) AS count_go_link
               FROM df
               WHERE (event_action LIKE '%%click_go_to_sberbank%%' OR event_action LIKE '%%click_link_part%%')
                AND timestamp>='2022-09-12' AND timestamp<'2022-09-23'
               GROUP BY user_id
               HAVING COUNT(DISTINCT event_action) = 2) AS filter
          """
select(sql)
```

Out[16]:

	count
0	3

Решение при помощи Python

Отфильтруем датасет по пользователям, совершившим событие click_go_to_sberbank и событие click_link_part за период с 2022-09-12' по '2022-09-22', включительно.

In [17]:
```
df_filter = df.query("(event_action == "click_go_to_sberbank" or event_action == "click_link_part") and timestamp>="2022-09-12" and timestamp<"2022-09-23"")
```

Рассчитаем количество уникальных пользователей, совершивших событие click_go_to_sberbank и событие click_link_part.

In [18]:
```
df_filter.groupby('user_id')[['event_action']].nunique().query('event_action==2')['event_action'].count()
```

Out[18]:
```
3
```