

# Задание по SQL. Остатки по счетам.

## Дано

Имеется таблица проводок по счетам, отражающая факты начисления или списания денежных средств по счетам клиента. Состоит из столбцов:

- BUSINESS\_DT - дата проводки
- ACCOUNT\_DEBIT\_ID - идентификатор счета дебита, с которого списываются средства
- ACCOUNT\_CREDIT\_ID - идентификатор счета кредита, на который начисляются средства
- POSTING\_AMT - сумма проводки в рублях. Т.е. сумма средств, которая списывается со счета дебита и зачисляется на счет кредита. Значение всегда положительное.

```
APEX.FA.RU ORDS ACADEMY02
APEX App Builder SQL Workshop Team Development Gallery
SQL Scripts \ Script Editor
Script Name SberSQL
Cancel Download Delete Save Create App Run

1 CREATE TABLE Sber (
2   BUSINESS_DT DATE NOT NULL,
3   ACCOUNT_DEBIT_ID INT NOT NULL,
4   ACCOUNT_CREDIT_ID INT NOT NULL,
5   POSTING_AMT INT NOT NULL
6 );
7
8 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
9 VALUES (TO_DATE('01.01.20','DD.MM.YY'), 1, 2, 1000);
10 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
11 VALUES (TO_DATE('05.01.20','DD.MM.YY'), 2, 1, 500);
12 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
13 VALUES (TO_DATE('04.02.20','DD.MM.YY'), 2, 1, 500);
14 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
15 VALUES (TO_DATE('04.03.20','DD.MM.YY'), 1, 2, 1000);
16 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
17 VALUES (TO_DATE('05.04.20','DD.MM.YY'), 2, 1, 500);
18 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
19 VALUES (TO_DATE('06.04.20','DD.MM.YY'), 1, 2, 100);
20 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
21 VALUES (TO_DATE('07.04.20','DD.MM.YY'), 1, 2, 230);
22 INSERT INTO Sber (BUSINESS_DT, ACCOUNT_DEBIT_ID, ACCOUNT_CREDIT_ID, POSTING_AMT)
23 VALUES (TO_DATE('08.04.20','DD.MM.YY'), 2, 1, 600);
```

```
1 SELECT * FROM SBER
```

BUSINESS_DT	ACCOUNT_DEBIT_ID	ACCOUNT_CREDIT_ID	POSTING_AMT
01/01/2020	1	2	1000
01/05/2020	2	1	500
02/04/2020	2	1	500
03/04/2020	1	2	1000
04/05/2020	2	1	500
04/06/2020	1	2	100
04/07/2020	1	2	230
04/08/2020	2	1	600

## Задание 1

Написать SQL-запрос, который на основании таблицы проводок рассчитает остатки по счетам на текущую дату. Начальные остатки по всем счетам полагаем равными нулю. Таблица должна состоять из следующих столбцов:

- Account\_ID – идентификатор счета
- Current\_Date – дата, на которую посчитан остаток
- Account\_Balance – остаток на счете, может быть отрицательным.

SQL-Запрос с комментариями:

```
1 WITH
2
3 -- SBER_DEBIT - это таблица, которая хранит в себе накопительную сумму проводок по дебиту счёта
4 SBER_DEBIT AS
5 (SELECT
6     ACCOUNT_DEBIT_ID,
7     BUSINESS_DT,
8     SUM(POSTING_AMT) OVER (PARTITION BY -- Для накопительной суммы используется оконная функция SUM с группировкой по счетам и сортировкой по датам
9                             ACCOUNT_DEBIT_ID
10                            ORDER BY
11                             BUSINESS_DT) AS SUM_POST
12 FROM
13     SBER),
14
15
16 -- SBER_CREDIT - это таблица, которая хранит в себе накопительную сумму проводок по кредиту счёта
17 SBER_CREDIT AS
18 (SELECT
19     ACCOUNT_CREDIT_ID,
20     BUSINESS_DT,
21     SUM(POSTING_AMT) OVER (PARTITION BY -- Для накопительной суммы используется оконная функция SUM с группировкой по счетам и сортировкой по датам
22                             ACCOUNT_CREDIT_ID
23                            ORDER BY
24                             BUSINESS_DT) AS SUM_POST
25 FROM
26     SBER),
27
28
29 -- ACCOUNTS - это множество всех используемых счетов
30 ACCOUNTS AS
31 (SELECT
32     DISTINCT ACCOUNT_DEBIT_ID AS ACCOUNT_ID
33 FROM
34     SBER
35 UNION -- Используется UNION, так как существует возможность, что счёт использовался только по дебиту или только по кредиту
36 SELECT
37     DISTINCT ACCOUNT_CREDIT_ID AS ACCOUNT_ID
38 FROM
39     SBER),
40
41
42
43
44 -- DATES - это множество уникальных дат, в которые были совершены проводки по различным счетам
45 DATES AS
46 (SELECT
47     DISTINCT BUSINESS_DT AS BUSINESS_DATE
48 FROM
49     SBER)
50
51
52 SELECT
53     ACCOUNT_ID AS "Account_ID",
54     BUSINESS_DATE AS "Current_Date",
55     -- Баланс счёта рассчитывается как разница между накопительными суммами проводок по кредиту и дебиту
56     -- Подзапрос ищет накопительную сумму проводок по кредиту для счёта ACCOUNT_ID на дату, соответствующую BUSINESS_DATE
57     -- (наиболее близкую к BUSINESS_DATE, но не превосходящую её)
58     -- Если на текущую дату нет никакой информации о счёте, то по умолчанию используется ноль
59     NVL((SELECT
60         SUM_POST
61     FROM
62         SBER_CREDIT
63     -- Отбор строки по номеру счёта и по неперевосходящей дате
64     WHERE
65         BUSINESS_DT <= BUSINESS_DATE
66     AND
67         ACCOUNT_CREDIT_ID = ACCOUNT_ID
68     -- Отбор строки с наиболее близкой к BUSINESS_DATE дате
69     ORDER BY
70         BUSINESS_DT DESC
71     FETCH FIRST 1 ROWS ONLY), 0)
72
73     -- Подзапрос ищет накопительную сумму проводок по дебиту для счёта ACCOUNT_ID на дату, соответствующую BUSINESS_DATE
74     -- (наиболее близкую к BUSINESS_DATE, но не превосходящую её)
75     -- Если на текущую дату нет никакой информации о счёте, то по умолчанию используется ноль
76     NVL((SELECT
77         SUM_POST
78     FROM
79         SBER_DEBIT
80     -- Отбор строки по номеру счёта и по неперевосходящей дате
81     WHERE
82         BUSINESS_DT <= BUSINESS_DATE
83     AND
84         ACCOUNT_DEBIT_ID = ACCOUNTS.ACCOUNT_ID
85     -- Отбор строки с наиболее близкой к BUSINESS_DATE дате
86     ORDER BY
87         BUSINESS_DT DESC
88     FETCH FIRST 1 ROWS ONLY), 0) AS "Account_Balance"
89
90 FROM
91     ACCOUNTS, DATES
92 ORDER BY
93     BUSINESS_DATE, ACCOUNT_ID
94
```

Вывод после выполнения SQL-запроса:

Account_ID	Current_Date	Account_Balance
1	01/01/2020	-1000
2	01/01/2020	1000
1	01/05/2020	-500
2	01/05/2020	500
1	02/04/2020	0
2	02/04/2020	0
1	03/04/2020	-1000
2	03/04/2020	1000
1	04/05/2020	-500
2	04/05/2020	500
1	04/06/2020	-600
2	04/06/2020	600
1	04/07/2020	-830
2	04/07/2020	830

## Задание 2

Написать SQL-запрос, который выводит таблицу остатков за весь период, за который нам известны проводки (например, с января по март). Начальные остатки по всем счетам полагаем равными нулю. Таблица должна состоять из следующих столбцов:

- Account\_ID – идентификатор счета
- Business\_From\_DT – дата, начиная с которой (включительно) сформировался указанный остаток на счете.
- Business\_To\_DT – дата, по которую (включительно) на счете находится указанный остаток. Если остаток действует по текущий момент (дата следующего изменения остатка не известна), то поле следует заполнить датой 9999-12-31.
- Account\_Balance – остаток на счете Account\_ID в период с Business\_From\_DT по Business\_To\_DT включительно.

## SQL-Запрос с комментариями:

```
1 WITH
2
3 -- SBER_DEBIT - это таблица, которая хранит в себе накопительную сумму проводок по дебиту счёта
4 SBER_DEBIT AS
5 (SELECT
6     ACCOUNT_DEBIT_ID,
7     BUSINESS_DT,
8     SUM(POSTING_AMT) OVER (PARTITION BY -- Для накопительной суммы используется оконная функция SUM с группировкой по счетам и сортировкой по датам
9                             ACCOUNT_DEBIT_ID
10                            ORDER BY
11                             BUSINESS_DT) AS SUM_POST
12 FROM
13     SBER),
14
15
16 -- SBER_CREDIT - это таблица, которая хранит в себе накопительную сумму проводок по кредиту счёта
17 SBER_CREDIT AS
18 (SELECT
19     ACCOUNT_CREDIT_ID,
20     BUSINESS_DT,
21     SUM(POSTING_AMT) OVER (PARTITION BY -- Для накопительной суммы используется оконная функция SUM с группировкой по счетам и сортировкой по датам
22                             ACCOUNT_CREDIT_ID
23                            ORDER BY
24                             BUSINESS_DT) AS SUM_POST
25 FROM
26     SBER),
27
28
29 -- DATES - это объединение множества уникальных дат и всех счетов
30 DATES AS
31 (SELECT
32     DISTINCT ACCOUNT_DEBIT_ID AS ACCOUNT_ID, BUSINESS_DT
33 FROM
34     SBER_DEBIT
35 UNION -- Используется UNION, так как существует возможность, что счёт использовался только по дебиту или только по кредиту
36 SELECT
37     DISTINCT ACCOUNT_CREDIT_ID AS ACCOUNT_ID, BUSINESS_DT
38 FROM
39     SBER_CREDIT)
40
41
42 SELECT
43     DT.ACCOUNT_ID AS "Account_ID",
44     DT.BUSINESS_DT AS "Business_From_DT",
45     -- Business_To_DT - это дата, по которую (включительно) на счете находится указанный остаток
46     -- Подзапрос выбирает следующую дату после изменения счёта, отнимает от нее один день и при помощи отбора подставляет в нужную строку Business_To_DT
47     -- Если остаток действует по текущий момент (дата следующего изменения остатка не известна), то по умолчанию в Business_To_DT вставляется дата 31-12-9999
48     NVL((SELECT
49         DATES.BUSINESS_DT - NUMTODSINTERVAL(1, 'day')
50     FROM
51         DATES
52     -- Отбор строки по номеру счёта и по превосходящей дате
53     WHERE
54         DATES.BUSINESS_DT > DT.BUSINESS_DT
55     AND
56         DATES.ACCOUNT_ID = DT.ACCOUNT_ID
57     -- Отбор строки с наиболее близкой к BUSINESS_DT дате
58     ORDER BY
59         BUSINESS_DT
60     FETCH FIRST 1 ROWS ONLY), TO_DATE('9999-12-31','YYYY-MM-DD')) AS "Business_To_DT",
61
62 -- Баланс счёта рассчитывается как разница между накопительными суммами проводок по кредиту и дебиту
63
64 -- Подзапрос ищет накопительную сумму проводок по кредиту для счёта ACCOUNT_ID на дату, соответствующую BUSINESS_DT
65 -- (наиболее близкую к BUSINESS_DT, но не превосходящую её)
66 -- Если на текущую дату нет никакой информации о счёте, то по умолчанию используется ноль
67 NVL((SELECT
68     SBER_CREDIT.SUM_POST
69 FROM
70     SBER_CREDIT
71 -- Отбор строки по номеру счёта и по не превосходящей дате
72 WHERE
73     SBER_CREDIT.BUSINESS_DT <= DT.BUSINESS_DT
74 AND
75     SBER_CREDIT.ACCOUNT_CREDIT_ID = DT.ACCOUNT_ID
76 -- Отбор строки с наиболее близкой к BUSINESS_DT дате
77 ORDER BY
78     SBER_CREDIT.BUSINESS_DT DESC
79     FETCH FIRST 1 ROWS ONLY), 0)
80
81 -- Подзапрос ищет накопительную сумму проводок по дебиту для счёта ACCOUNT_ID на дату, соответствующую BUSINESS_DT
82 -- (наиболее близкую к BUSINESS_DT, но не превосходящую её)
83 -- Если на текущую дату нет никакой информации о счёте, то по умолчанию используется ноль
84 NVL((SELECT
85     SBER_DEBIT.SUM_POST
86 FROM
87     SBER_DEBIT
88 WHERE
89     SBER_DEBIT.BUSINESS_DT <= DT.BUSINESS_DT
90 AND
91     SBER_DEBIT.ACCOUNT_DEBIT_ID = DT.ACCOUNT_ID
92 -- Отбор строки с наиболее близкой к BUSINESS_DT дате
93 ORDER BY
94     SBER_DEBIT.BUSINESS_DT DESC
95     FETCH FIRST 1 ROWS ONLY), 0) AS "Account_Balance"
96
97 FROM
98     DATES DT
99 ORDER BY
100     BUSINESS_DT, ACCOUNT_ID
```

Вывод после выполнения SQL-запроса:

Account_ID	Business_From_DT	Business_To_DT	Account_Balance
1	01/01/2020	01/04/2020	-1000
2	01/01/2020	01/04/2020	1000
1	01/05/2020	02/03/2020	-500
2	01/05/2020	02/03/2020	500
1	02/04/2020	03/03/2020	0
2	02/04/2020	03/03/2020	0
1	03/04/2020	04/04/2020	-1000
2	03/04/2020	04/04/2020	1000
1	04/05/2020	04/05/2020	-500
2	04/05/2020	04/05/2020	500
1	04/06/2020	04/06/2020	-600
2	04/06/2020	04/06/2020	600
1	04/07/2020	04/07/2020	-830
2	04/07/2020	04/07/2020	830

### Задание 3

Написать SQL запрос, который выведет остатки на каждый день за период с января по март. Т.е. по каждому счету таблица должна содержать около 90 записей. Начальные остатки по всем счетам полагаем равными нулю. Столбцы таблицы:

- Account\_ID – идентификатор счета
- Business\_Date – дата, на которую посчитан остаток
- Account\_Balance – остаток на счете, может быть отрицательным. Например, по счету 2 должны быть такие записи:

2, 2020-01-01, 1000

2, 2020-01-02, 1000

2, 2020-01-03, 1000

2, 2020-01-04, 1000

2, 2020-01-05, 500

## SQL-Запрос с комментариями:

```
1  WITH
2
3  -- SBER_DEBIT - это таблица, которая хранит в себе накопительную сумму проводок по дебиту счёта
4  SBER_DEBIT AS
5  (SELECT
6    ACCOUNT_DEBIT_ID,
7    BUSINESS_DT,
8    SUM(POSTING_AMT) OVER (PARTITION BY -- Для накопительной суммы используется оконная функция SUM с группировкой по счетам и сортировкой по датам
9                           ACCOUNT_DEBIT_ID
10                          ORDER BY
11                           BUSINESS_DT) AS SUM_POST
12  FROM
13    SBER),
14
15
16
17
18  -- SBER_CREDIT - это таблица, которая хранит в себе накопительную сумму проводок по кредиту счёта
19  SBER_CREDIT AS
20  (SELECT
21    ACCOUNT_CREDIT_ID,
22    BUSINESS_DT,
23    SUM(POSTING_AMT) OVER (PARTITION BY -- Для накопительной суммы используется оконная функция SUM с группировкой по счетам и сортировкой по датам
24                           ACCOUNT_CREDIT_ID
25                          ORDER BY
26                           BUSINESS_DT) AS SUM_POST
27  FROM
28    SBER),
29
30
31
32
33  -- ACCOUNTS - это множество всех используемых счетов
34  ACCOUNTS AS
35  (SELECT
36    DISTINCT ACCOUNT_DEBIT_ID AS ACCOUNT_ID
37  FROM
38    SBER
39  UNION -- Используется UNION, так как существует возможность, что счёт использовался только по дебиту или только по кредиту
40  SELECT
41    DISTINCT ACCOUNT_CREDIT_ID AS ACCOUNT_ID
42  FROM
43    SBER),
44
45
46
47
48  -- DATE_RANGE - это таблица, которая хранит в себе строго ограниченный набор дат (в данном случае с января по март)
49  -- Чтобы получить нужное количество дней, первая дата (2020-01-01) вычитается из последней (2020-03-31)
50  -- При помощи оператора LEVEL создается полученное количество дат
51  -- В запросе присутствует вычитание (- 1) и сложение (+ 1), с помощью данных операций в диапазон дат включаются даты начала и окончания периода
52  -- (2020-01-01 и 2020-03-31 соответственно)
53  DATE_RANGE AS
54  (SELECT
55    TO_DATE('2020-01-01', 'YYYY-MM-DD') + LEVEL - 1 AS BUSINESS_DATE
56  FROM
57    DUAL
58  CONNECT BY LEVEL <= (
59    TO_DATE('2020-03-31', 'YYYY-MM-DD') - TO_DATE('2020-01-01', 'YYYY-MM-DD') + 1
60  ))
61
62
63
64
65  SELECT
66    ACCOUNT_ID as "Account_ID",
67    BUSINESS_DATE AS "Business_Date",
68    -- Баланс счёта рассчитывается как разница между накопительными суммами проводок по кредиту и дебиту
69
70    -- Подзапрос ищет накопительную сумму проводок по кредиту для счёта ACCOUNT_ID на дату, соответствующую BUSINESS_DATE
71    -- (наиболее близкую к BUSINESS_DATE, но не превосходящую её)
72    -- Если на текущую дату нет никакой информации о счёте, то по умолчанию используется ноль
73    NVL((SELECT
74          SUM_POST
75        FROM
76          SBER_CREDIT
77        -- Отбор строки по номеру счёта и по непревосходящей дате
78        WHERE
79          BUSINESS_DT <= BUSINESS_DATE
80        AND
81          ACCOUNT_CREDIT_ID = ACCOUNT_ID
82        -- Отбор строки с наиболее близкой к BUSINESS_DATE дате
83        ORDER BY
84          BUSINESS_DT DESC
85        FETCH FIRST 1 ROWS ONLY), 0)
86
87    -- Подзапрос ищет накопительную сумму проводок по дебиту для счёта ACCOUNT_ID на дату, соответствующую BUSINESS_DATE
88    -- (наиболее близкую к BUSINESS_DATE, но не превосходящую её)
89    -- Если на текущую дату нет никакой информации о счёте, то по умолчанию используется ноль
90    NVL((SELECT
91          SUM_POST
92        FROM
93          SBER_DEBIT
94        -- Отбор строки по номеру счёта и по непревосходящей дате
95        WHERE
96          BUSINESS_DT <= BUSINESS_DATE
97        AND
98          ACCOUNT_DEBIT_ID = ACCOUNTS.ACCOUNT_ID
99        -- Отбор строки с наиболее близкой к BUSINESS_DATE дате
100        ORDER BY
101          BUSINESS_DT DESC
102        FETCH FIRST 1 ROWS ONLY), 0) AS "Account_Balance"
103  FROM
104    ACCOUNTS, DATE_RANGE
105  ORDER BY
106    ACCOUNT_ID, BUSINESS_DATE
```

Вывод после выполнения SQL-запроса:

Results	Explain	Describe	Saved SQL	History
1			03/27/2020	-1000
1			03/28/2020	-1000
1			03/29/2020	-1000
1			03/30/2020	-1000
1			03/31/2020	-1000
2			01/01/2020	1000
2			01/02/2020	1000
2			01/03/2020	1000
2			01/04/2020	1000
2			01/05/2020	500
2			01/06/2020	500
2			01/07/2020	500
2			01/08/2020	500
2			01/09/2020	500
2			01/10/2020	500

# Задание по Python

Задание предлагается сделать, используя язык Python3

На входе есть таблица отделений банка (кол-во записей 100 – 10 000) с полями:

- DIV\_ID (ID отделения),
- CUR\_CAPACITY (целое – текущее кол-во клиентов в отделении),
- MAX\_CAPACITY (целое – максимальное кол-во клиентов в отделении)

И таблица с клиентами (кол-во записей 1000 – 1 000 000), которым нужно рекомендовать отделение для посещения, с полями:

- CLIENT\_ID (ID клиента),
- DIV1\_ID (самое предпочтительное отделение для клиента),
- DIV2\_ID (второе предпочтительное отделение для клиента),
- DIV3\_ID (третье предпочтительное отделение для клиента)

Результатом работы программы должна быть таблица CLIENT\_ID, DIV\_ID в которой мы для каждого клиента рекомендуем одно из отделений DIV1\_ID, DIV2\_ID, DIV3\_ID

При этом новая CUR\_CAPACITY для каждого отделения (считаем, что каждый клиент согласился на рекомендацию) не должна превышать MAX\_CAPACITY. (в качестве входных и выходных таблиц можно использовать Pandas DataFrame или иную подходящую структуру данных)

BANK:

	A	B	C
1	DIV_ID	CUR_CAPACITY	MAX_CAPACITY
2	1	4903	5000
3	2	2605	5000
4	3	1249	5000
5	4	2380	5000
6	5	4806	5000
7	6	1423	5000
8	7	1176	5000
9	8	4928	5000
10	9	1603	5000
11	10	1122	5000
12	11	4344	5000
13	12	4019	5000
14	13	1710	5000
15	14	2373	5000

KLI:

	A	B	C	D
1	CLIENT_ID	DIV1_ID	DIV2_ID	DIV3_ID
2	1	10	41	38
3	2	65	96	65
4	3	1	46	35
5	4	95	33	69
6	5	71	64	91
7	6	80	34	36
8	7	96	34	36
9	8	46	30	52
10	9	87	94	25
11	10	74	8	34
12	11	56	51	96
13	12	86	23	90
14	13	65	75	83

В BANK 100 записей

В KLI 1000 записей



## Программа с комментариями:

### Подготовка среды

```
import numpy as np
import pandas as pd
```

[1] ✓ 3.2s

Python

### Считывание входных данных, их обработка и создание DataFrame для итогового распределения

```
# Загрузка данных клиентов, установка ID клиента в качестве индекса
clients = pd.read_excel("./KLI.xlsx").set_index('CLIENT_ID')

# Загрузка данных отделений, установка ID отделения в качестве индекса
banks = pd.read_excel("./BANK.xlsx").set_index('DIV_ID')

# Создание вспомогательного столбца FREE_CAPACITY, который соответствует количеству свободных мест в отделении
banks['FREE_CAPACITY'] = banks['MAX_CAPACITY'] - banks['CUR_CAPACITY']

# Создание DataFrame для итогового распределения клиентов по отделениям
# В качестве индекса устанавливается ID клиента
distribution = pd.DataFrame(columns=['CLIENT_ID', 'DIV_ID'], dtype='int64').set_index('CLIENT_ID')
```

[2] ✓ 0.7s

Python

### Распределение клиентов по их приоритетным отделениям

```
# Алгоритм заключается в следующем:
# Каждый клиент распределяется в свое первое по приоритету отделение
# Отбираются те клиенты, которым не хватило места в своем приоритетном отделении
# Далее эти клиенты распределяются в свое второе по приоритету отделение
# Если и в этот раз для некоторых из клиентов не хватило места в этих отделениях
# Они распределяются в свое третье по приоритету отделение
for div in ['DIV1_ID', 'DIV2_ID', 'DIV3_ID']:

    # grouped_clients - это клиенты сгруппированные по ID текущих приоритетных направлений (DIV1_ID, DIV2_ID, DIV3_ID),
    # которым присвоены номера в своей группе
    grouped_clients = clients.groupby(div).cumcount().to_frame().rename(columns={0: 'NUMBER_IN_DIV'})

    # div_capacities - количество свободных мест в каждом из отделений (используется для маски)
    div_capacities = banks.loc[clients.loc[grouped_clients.index.values][div]]['FREE_CAPACITY'].values

    # numbers_in_divs - это порядковые номера клиентов в своем приоритетном отделении (используется для маски)
    numbers_in_divs = grouped_clients['NUMBER_IN_DIV'].values

    # Отбор тех клиентов, для которых нашлись свободные места в отделениях
    grouped_clients = grouped_clients[numbers_in_divs < div_capacities]

    # Создается новый столбец, который представляет собой ID отделения, в которое попал клиент
    grouped_clients = pd.merge(grouped_clients, clients.rename(columns={div: 'DIV_ID'})[
        ['DIV_ID']], left_index=True, right_index=True)

    # Учет изменения количества свободных мест в отделениях:
    # Для изменения количества свободных мест в отделениях создается пустой DataFrame с индексами отделений, заполненный нулями
    # Из этого DataFrame вычитается количество клиентов попавших в каждое из отделений
    # Так как на этом этапе могут возникнуть NaN значения (не все отделения могли быть задействованы)
    # Эти значения заменяются нулями
    # Получившийся DataFrame представляет собой изменение количества свободных мест в каждом из отделений,
    # которые в последствии складываются со столбцом FREE_CAPACITY
    banks['FREE_CAPACITY'] += (pd.DataFrame(data=np.zeros(shape=banks.index.size),
                                                dtype='int64',
                                                index=banks.index)
                               .iloc[:, 0] - grouped_clients.groupby('DIV_ID').count().iloc[:, 0]).fillna(0)

    # Сохранение информации о клиентах, которые были распределены
    distribution = pd.concat([distribution['DIV_ID'], grouped_clients['DIV_ID']]).to_frame()

    # Удаление информации о распределённых клиентах и переход к следующему по приоритету отделениям
    # (для нераспределённых клиентов)
    clients.drop(grouped_clients.index, inplace=True)
```

[3] ✓ 0.9s

Результат работы программы:

## Вывод результатов

### Нераспределённые клиенты

```
clients #Клиенты, которые никуда не прошли
```

[4] ✓ 0.9s

...

	DIV1_ID	DIV2_ID	DIV3_ID
CLIENT_ID			
1000	98	98	98

### Распределённые клиенты

▷ ▾  
distribution.sort\_index()

[5] ✓ 0.6s

...

	DIV_ID
CLIENT_ID	
1	10
2	65
3	1
4	95
5	71
...	...
995	14
996	27
997	76
998	39
999	75

999 rows × 1 columns