Тестовое задание по SQL.

Елизавета Рыжова

```
import sqlite3
import pandas as pd

# Establish a connection to an in-memory SQLite database
conn = sqlite3.connect(":memory:")
```

Задача 1

Пусть имеется таблица рейсов flights (id, from, to) и таблица городов cities (label, name). Поля from, to и label содержат английские названия городов, поле name — русское. Выведите список рейсов flights с русскими названиями городов.

```
# Create the flights table
create_flights_table = """
CREATE TABLE flights (id INT, from_city TEXT, to_city TEXT);
"""
conn.execute(create_flights_table)

# Create the cities table
create_cities_table = """
CREATE TABLE cities (label TEXT, name TEXT);
"""
conn.execute(create_cities_table)

# Populate the flights table with sample data
insert_flights_data = """
INSERT INTO flights (id, from_city, to_city)
VALUES
      (1, 'moscow', 'omsk'),
      (2, 'novgorod', 'kazan'),
      (3, 'irkutsk', 'moscow'),
      (4, 'omsk', 'irkutsk'),
      (5, 'moscow', 'kazan');
"""
conn.execute(insert_flights_data)

# Populate the cities table with sample data
insert_cities_data = """
INSERT INTO cities (label, name)
VALUES
  ('moscow', 'Москва'),
  ('irkutsk', 'Иркутск'),
```

```
31   ('novgorod', 'Новгород'),
32   ('kazan', 'Казань'),
33   ('omsk', 'Омск');
34 """
35 conn.execute(insert_cities_data)
```

<sqlite3.Cursor at 0x24a2e5c88f0>

```
 1 # Define the SQL query
 2 sql_query = """
 3 SELECT f.id, cfrom.name AS from_city, cto.name AS to_city -- выбираем поле id
 4 FROM flights f
 5 JOIN cities cfrom ON f.from_city = cfrom.label -- первое соединение выполняет
 6 JOIN cities cto ON f.to_city = cto.label; -- второе соединение — по полю to и
 7 """
 8
 9 # Execute the query and fetch the results
10 results = conn.execute(sql_query).fetchall()
11
12 # Convert the results to a dataframe for easier visualization
13 df = pd.DataFrame(results, columns=['id', 'from_city', 'to_city'])
14 df
```

	id	from_city	to_city
0	1	Москва	Омск
1	2	Новгород	Казань
2	3	Иркутск	Москва
3	4	Омск	Иркутск
4	5	Москва	Казань

Задача 2

Таблица TRANSACTION – транзакции клиентов

CUST_ID	TRANSACTION_DT	CURRENCY_ID	TRANSACTION_AMT
1	01.01.2021	RUR	1000
1	02.01.2021	EUR	100
1	03.01.2021	EUR	100
1	04.01.2021	USD	50
2	01.01.2021	USD	150
2	02.01.2021	USD	200
2	04.01.2021	USD	50
3	01.01.2021	RUR	2000
3	04.01.2021	RUR	5000

Таблица CURRENCY_RATE – курсы валют

CURRENCY_ID	EXCH_RATE	VALID_FROM	VALID_TO
RUR	1	01.01.1900	01.01.4000
EUR	90	01.01.2021	03.01.2021
EUR	95	03.01.2021	01.01.4000
USD	70	01.01.2021	02.01.2021
USD	75	02.01.2021	04.01.2021
USD	70	04.01.2021	01.01.4000

Необходимо: Посчитать сумму транзакций клиента в рублях за месяц.

План запроса:

1. Возьмём идентификатор клиента (CUST_ID).
2. Отформатируем дату транзакции в столбце TRANSACTION_DT в формате 'гггг-мм' (по порядку года и месяца).
3. Вычислим общую сумму транзакций в рублях (TOTAL_TRANSACTION_IN_RUB). Умножим сумму транзакции (TRANSACTION_AMT) на курс валюты (EXCH_RATE), если такой курс валюты найден в таблице CURRENCY_RATE для соответствующей валюты транзакции и даты транзакции если входит в период действия курса валюты. Если курс валюты не найден, применим значение 1, чтобы сохранить оригинальную сумму транзакции в валюте.
4. Сгруппируем результаты по идентификатору клиента (CUST_ID) и по году и месяцу транзакции (TRANSACTION_MONTH).

```
# Create the TRANSACTION table
create_transaction_table = """
CREATE TABLE "TRANSACTION" (
  CUST_ID INT,
  TRANSACTION_DT TEXT,
  CURRENCY_ID TEXT,
  TRANSACTION_AMT INT
);
"""
conn.execute(create_transaction_table)

# Create the CURRENCY_RATE table
create_currency_rate_table = """
CREATE TABLE CURRENCY_RATE (
  CURRENCY_ID TEXT,
  EXCH_RATE INT,
  VALID_FROM TEXT,
  VALID_TO TEXT
);
"""
conn.execute(create_currency_rate_table)

# Populate the TRANSACTION table with sample data
```

```
insert_transaction_data = """
INSERT INTO "TRANSACTION"(CUST_ID, TRANSACTION_DT, CURRENCY_ID, TRANSACTION_A
VALUES
  (1, '01.01.2021', 'RUR', 1000),
  (1, '02.01.2021', 'EUR', 100),
  (1, '03.01.2021', 'EUR', 100),
  (1, '04.01.2021', 'USD', 50),
  (2, '01.01.2021', 'USD', 150),
  (2, '02.01.2021', 'USD', 200),
  (2, '04.01.2021', 'USD', 50),
  (3, '01.01.2021', 'RUR', 2000),
  (3, '04.01.2021', 'RUR', 5000);
"""
conn.execute(insert_transaction_data)

# Populate the CURRENCY_RATE table with sample data
insert_currency_rate_data = """
INSERT INTO CURRENCY_RATE (CURRENCY_ID, EXCH_RATE, VALID_FROM, VALID_TO)
VALUES
  ('RUR', 1, '01.01.1900', '01.01.4000'),
  ('EUR', 90, '01.01.2021', '03.01.2021'),
  ('EUR', 95, '03.01.2021', '01.01.4000'),
  ('USD', 70, '01.01.2021', '02.01.2021'),
  ('USD', 75, '02.01.2021', '04.01.2021'),
  ('USD', 70, '04.01.2021', '01.01.4000');
"""
conn.execute(insert_currency_rate_data)

```

```
<sqlite3.Cursor at 0x24a2e703a40>
```

```
# Define the SQL query
sql_query = """
SELECT
    CUST_ID, -- Выбираем идентификатор клиента
    SUBSTR(TRANSACTION_DT, 7, 4) || '-' || SUBSTR(TRANSACTION_DT, 4, 2) AS TR
    SUM(
        TRANSACTION_AMT * IFNULL(
            (SELECT
            EXCH_RATE
        FROM
            CURRENCY_RATE
        WHERE
            CURRENCY_RATE.CURRENCY_ID = [TRANSACTION].CURRENCY_ID -- Сопостав
            AND CURRENCY_RATE.VALID_FROM <= [TRANSACTION].TRANSACTION_DT -- П
            AND (CURRENCY_RATE.VALID_TO > [TRANSACTION].TRANSACTION_DT OR VAL
        ),
         1) -- Если курс валюты не найден, используем значение 1, чтобы сохра
    ) AS TOTAL_TRANSACTION_IN_RUB -- Вычисляем общую сумму транзакций в рубля
FROM
    [TRANSACTION] -- Выбираем из таблицы [TRANSACTION]
```

```
GROUP BY
    CUST_ID, -- Группируем результаты по идентификатору клиента
    TRANSACTION_MONTH; -- И по году и месяцу транзакции

"""

# Execute the query and fetch the results
results = conn.execute(sql_query).fetchall()

# Convert the results to a dataframe for easier visualization
df = pd.DataFrame(results, columns=['CUST_ID', 'TRANSACTION_MONTH', 'TRANSACT
df

```

	CUST_ID	TRANSACTION_MONTH	TRANSACTION_AMOUNT_RUB
0	1	2021-01	23000
1	2	2021-01	29000
2	3	2021-01	7000

Задача 3

Имеется таблица Orders, в которой содержатся ордера за 5 лет. В таблице 10 млн записей, есть ключ по полю ID, индексов нет. Примечание: скрипты приведены на диалекте Transact-SQL.

```
DROP TABLE IF EXISTS dbo.Orders;

CREATE TABLE dbo.Orders
(
         ID               int             NOT NULL identity(1,1) primary key
,DT datetime          NOT NULL
,ClientID int              NULL
,Summ       decimal(10,2)  NOT NULL
,comment  varchar(max)   NULL
);
```

Данные для теста:

```
INSERT INTO dbo.Orders(ID, DT, ClientID, Summ, comment)
VALUES (1,'20200101 01:00',1,100,NULL)
,(2,'20200102 01:00',2,200,NULL)
,(3,'20200103 01:00',2,300,NULL)
,(4,'20200112 01:00',3,400,NULL)
,(5,'20200115 01:00',3,500,NULL)
,(6,'20200202 01:00',1,100,NULL)
```

```
,(7,'20200212 02:00',2,200,NULL)
,(8,'20200220 06:00',3,300,NULL)
,(9,'20200225 04:00',1,400,NULL)
,(10,'20200320 01:00',1,100,NULL)
,(11,'20200331 01:00',1,100,NULL)
,(12,'20200505 01:00',1,100,NULL)
```

P.S. не нужно решать проблемы вставки данных в автоинкремент, важна логика решения задачи.

В начале месяца первой строкой записать сумму всех заказов за предыдущий месяц следующим образом - DT = начало месяца без времени (к примеру 2020-01-01) - ClientID = NULL - Summ = сумма всех ордеров за предшествующий месяц - comment = все уникальные ClientID за предыдущий месяц через запятую. При этом, если есть месяца без ордеров, писать в summ = 0, а в comment = '' В результате решения первой задачи должны добавиться следующие строки:

```
,('20200201',NULL,1500,'1,2,3')
,('20200301',NULL,1000,'1,2,3')
,('20200401',NULL,200,'1')
,('20200501',NULL,0,'')
,('20200601',NULL,100,'1')
```

Задача состоит в добавлении строк в таблицу Orders в начале каждого месяца для предыдущего месяца. Эти строки должны содержать сумму всех заказов за предшествующий месяц, уникальные идентификаторы клиентов через запятую и другие значения, в соответствии с требованиями задачи.

Нам нужно выполнить следующие шаги, чтобы решить эту задачу:

Создать подзапрос, который будет вычислять сумму всех ордеров за предыдущий месяц и список уникальных идентификаторов клиентов.

Внутри этого подзапроса используем функцию DATEADD и DATEDIFF, чтобы получить начало текущего месяца. Это позволит получить предыдущий месяц. Сгруппируем записи по началу предыдущего месяца и используем функцию SUM для вычисления суммы ордеров и функцию STRING_AGG, чтобы объединить идентификаторы клиентов через запятую. В основном запросе используем оператор INSERT INTO для добавления строк в таблицу Orders с результатом подзапроса из предыдущего шага.

Положим в поле DT начало месяца без времени. Положим в поле ClientID NULL. Поле Summ должно содержать сумму ордеров за предыдущий месяц. Поле comment должно содержать уникальные идентификаторы клиентов через запятую. Добавим сценарии, когда нет ордеров в предыдущем месяце.

Для месяцев без ордеров положим в поле Summ - 0. Положим в поле comment в пустую строку.

```
-- Шаг 1: Создать подзапрос
WITH PreviousMonthOrders AS (
  SELECT DATEADD(MONTH, DATEDIFF(MONTH, 0, DT), 0) AS StartOfMonth,
         NULL AS ClientID,
         SUM(Summ) AS Summ,
         STRING_AGG(CAST(ClientID AS VARCHAR), ',') WITHIN GROUP (ORDER BY ClientID) AS Comment
  FROM dbo.Orders
  WHERE DT < DATEADD(MONTH, -1, GETDATE())
  GROUP BY DATEADD(MONTH, DATEDIFF(MONTH, 0, DT), 0)
),

-- Шаг 2: Вставить строки в таблицу Orders
InsertOrders AS (
  SELECT StartOfMonth, ClientID, Summ, Comment
  FROM PreviousMonthOrders

  UNION ALL

  SELECT DATEADD(MONTH, 0, DT) AS StartOfMonth,
         NULL AS ClientID,
         0 AS Summ,
         '' AS Comment
  FROM dbo.Orders
  WHERE DT >= DATEADD(MONTH, -1, GETDATE())
  AND NOT EXISTS (
    SELECT 1
    FROM PreviousMonthOrders
    WHERE StartOfMonth = DATEADD(MONTH, 0, DT)
  )
)

INSERT INTO dbo.Orders (DT, ClientID, Summ, Comment)
SELECT StartOfMonth, ClientID, Summ, Comment
FROM InsertOrders;
```