

04 ДЗ - Функции для работы с типами данных, агрегатные функции и UDF.

Собираем Dockerfile для clickhouse.

Dockerfile:

```
FROM clickhouse/clickhouse-server:25.2.1
MAINTAINER Maksim Kulikov <max.uoles@rambler.ru>

RUN apt-get update -y --fix-missing
RUN DEBIAN_FRONTEND=noninteractive apt-get -yq upgrade
RUN apt-get install nano mc python3 pip kafkacat -y
RUN pip install clickhouse_driver

COPY clickhouse/config.xml /etc/clickhouse-server/config.d/config.xml

COPY clickhouse/user_scripts/transaction_state.py /var/lib/clickhouse/user_scripts/transaction_state.py
COPY clickhouse/user_scripts/transaction_sum.py /var/lib/clickhouse/user_scripts/transaction_sum.py

COPY clickhouse/transaction_state.xml /etc/clickhouse-server/transaction_state.xml
COPY clickhouse/transaction_sum.xml /etc/clickhouse-server/transaction_sum.xml

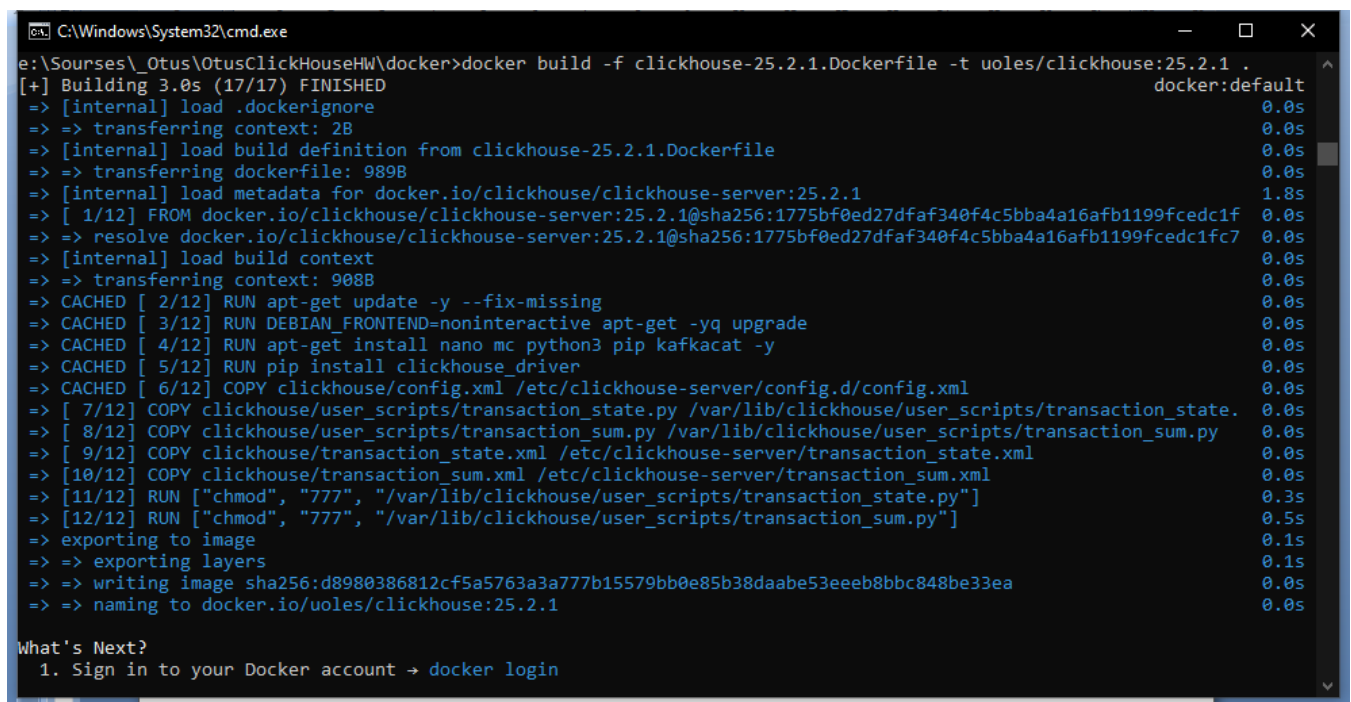
RUN ["chmod", "+x", "/var/lib/clickhouse/user_scripts/transaction_state.py"]
RUN ["chmod", "+x", "/var/lib/clickhouse/user_scripts/transaction_sum.py"]

EXPOSE 8123 9000

ENTRYPOINT ["/entrypoint.sh"]
```

Переходим в папку docker и собираем образ командой:

```
docker build -f clickhouse-25.2.1.Dockerfile -t uoles/clickhouse:25.2.1 .
```

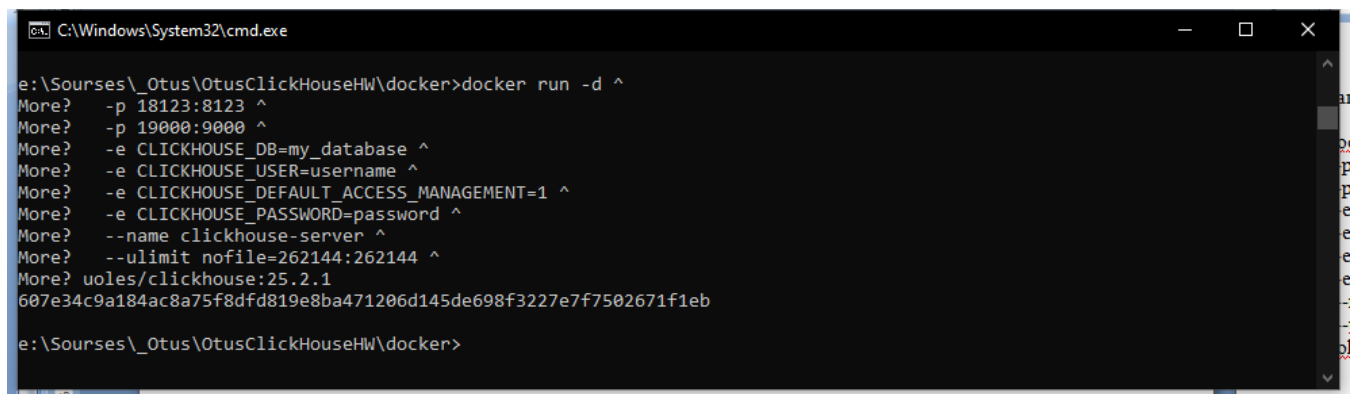


```
C:\Windows\System32\cmd.exe
e:\Sources\Otus\OtusClickHouseHW\docker>docker build -f clickhouse-25.2.1.Dockerfile -t uoles/clickhouse:25.2.1 .
[+] Building 3.0s (17/17) FINISHED
=> [internal] load .dockerignore                                docker:default 0.0s
=> => transferring context: 2B                                   0.0s
=> [internal] load build definition from clickhouse-25.2.1.Dockerfile 0.0s
=> => transferring dockerfile: 989B                             0.0s
=> [internal] load metadata for docker.io/clickhouse/clickhouse-server:25.2.1 1.8s
=> [ 1/12] FROM docker.io/clickhouse/clickhouse-server:25.2.1@sha256:1775bf0ed27dfaf340f4c5bba4a16afb1199fcedc1f 0.0s
=> => resolve docker.io/clickhouse/clickhouse-server:25.2.1@sha256:1775bf0ed27dfaf340f4c5bba4a16afb1199fcedc1f7 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 908B                                  0.0s
=> CACHED [ 2/12] RUN apt-get update -y --fix-missing          0.0s
=> CACHED [ 3/12] RUN DEBIAN_FRONTEND=noninteractive apt-get -yq upgrade 0.0s
=> CACHED [ 4/12] RUN apt-get install nano mc python3 pip kafkacat -y 0.0s
=> CACHED [ 5/12] RUN pip install clickhouse_driver            0.0s
=> CACHED [ 6/12] COPY clickhouse/config.xml /etc/clickhouse-server/config.d/config.xml 0.0s
=> [ 7/12] COPY clickhouse/user_scripts/transaction_state.py /var/lib/clickhouse/user_scripts/transaction_state. 0.0s
=> [ 8/12] COPY clickhouse/user_scripts/transaction_sum.py /var/lib/clickhouse/user_scripts/transaction_sum.py 0.0s
=> [ 9/12] COPY clickhouse/transaction_state.xml /etc/clickhouse-server/transaction_state.xml 0.0s
=> [10/12] COPY clickhouse/transaction_sum.xml /etc/clickhouse-server/transaction_sum.xml 0.0s
=> [11/12] RUN ["chmod", "777", "/var/lib/clickhouse/user_scripts/transaction_state.py"] 0.3s
=> [12/12] RUN ["chmod", "777", "/var/lib/clickhouse/user_scripts/transaction_sum.py"] 0.5s
=> exporting to image                                          0.1s
=> => exporting layers                                          0.1s
=> => writing image sha256:d8980386812cf5a5763a3a777b15579bb0e85b38daabe53eeeb8bbc848be33ea 0.0s
=> => naming to docker.io/uoles/clickhouse:25.2.1             0.0s

What's Next?
1. Sign in to your Docker account -> docker login
```

Запускаем контейнер командой:

```
docker run -d ^  
-p 18123:8123 ^  
-p 19000:9000 ^  
-e CLICKHOUSE_DB=my_database ^  
-e CLICKHOUSE_USER=username ^  
-e CLICKHOUSE_DEFAULT_ACCESS_MANAGEMENT=1 ^  
-e CLICKHOUSE_PASSWORD=password ^  
--name clickhouse-server ^  
--ulimit nofile=262144:262144 ^  
uoles/clickhouse:25.2.1
```



```
C:\Windows\System32\cmd.exe  
e:\Sources\_Otus\OtusClickHouseHW\docker>docker run -d ^  
More? -p 18123:8123 ^  
More? -p 19000:9000 ^  
More? -e CLICKHOUSE_DB=my_database ^  
More? -e CLICKHOUSE_USER=username ^  
More? -e CLICKHOUSE_DEFAULT_ACCESS_MANAGEMENT=1 ^  
More? -e CLICKHOUSE_PASSWORD=password ^  
More? --name clickhouse-server ^  
More? --ulimit nofile=262144:262144 ^  
More? uoles/clickhouse:25.2.1  
607e34c9a184ac8a75f8dfd819e8ba471206d145de698f3227e7f7502671f1eb  
e:\Sources\_Otus\OtusClickHouseHW\docker>
```

Создаем таблицу с тестовыми данными.

Создаем таблицу:

```
CREATE TABLE transactions (  
    transaction_id UInt32,  
    user_id UInt32,  
    product_id UInt32,  
    quantity UInt8,  
    price Float32,  
    transaction_date Date  
) ENGINE = MergeTree()  
ORDER BY (transaction_id);
```

Заполняем тестовыми данными:

```
INSERT INTO transactions  
SELECT  
    rand32() AS transaction_id,  
    randUniform(1,1000)::Int AS user_id,  
    randUniform(1,1000)::Int AS product_id,  
    randUniform(1,10)::Int AS cnt,  
    round( randUniform(15.5, 299.99), 2 ) AS price,  
    now() - toIntervalSecond(rand() % (365 * 24 * 60 * 60)) AS datetime  
FROM system.numbers  
LIMIT 1000;
```

	transaction_id	user_id	product_id	quantity	price	transaction_date
182	773 120 441	43	43	3	214,71	2024-10-23
183	777 122 156	689	689	1	216,74	2024-09-06
184	778 376 991	864	864	4	264,61	2024-08-23
185	788 150 486	225	225	9	80,52	2024-05-02
186	788 906 878	294	294	4	137,4	2025-04-23
187	811 308 641	717	717	4	50,1	2024-08-07
188	811 757 571	652	652	9	128,16	2024-08-01
189	812 728 753	916	916	2	223,02	2024-07-21
190	817 091 298	908	908	6	291,65	2024-06-01
191	819 256 662	22	22	1	150,80	2024-05-05

Составляем запросы:

- Рассчитайте общий доход от всех операций.

```
select sum(price * quantity)
from transactions
```

	sum(multiply(price, quantity))
1	744 919,8599624634

- Найдите средний доход с одной сделки.

```
select avg(price * quantity)
from transactions
```

	avg(multiply(price, quantity))
1	744,9198599625

- Определите общее количество проданной продукции.

```
select sum(quantity)
from transactions
```

The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

```
63 -- Определите общее количество проданной продукции.
64 select sum(quantity)
65 from transactions
66
```

The results pane shows a table with one row and one column:

Таблица	1
sum(quantity)	4 852

- Подсчитайте количество уникальных пользователей, совершивших покупку.

```
select count(distinct user_id)
from transactions
```

The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

```
67 -- Подсчитайте количество уникальных пользователей, совершивших покупку.
68 select count(distinct user_id)
69 from transactions
70
```

The results pane shows a table with one row and one column:

Таблица	1
countDistinct(user_id)	641

- Преобразуйте `transaction_date` в строку формата `YYYY-MM-DD`.

```
select formatDateTime(transaction_date, '%Y-%m-%d') as NewDateTime,
       transaction_date
from transactions
```

The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

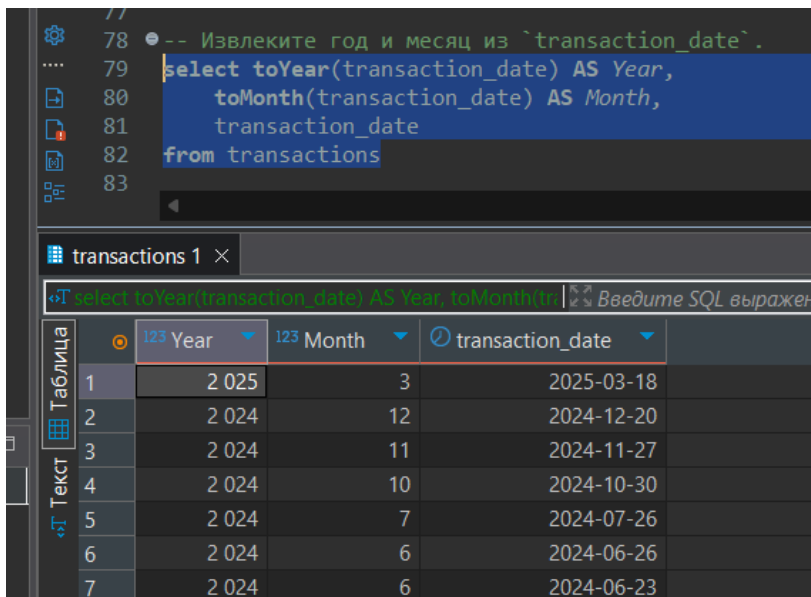
```
71 -- Преобразуйте `transaction_date` в строку формата `YYYY-MM-DD`.
72 select formatDateTime(transaction_date, '%Y-%m-%d') as NewDateTime,
73        transaction_date
74 from transactions
75
```

The results pane shows a table with two columns: NewDateTime and transaction_date. The data is as follows:

Таблица	1	2
NewDateTime	2025-03-18	2025-03-18
transaction_date	2024-12-20	2024-12-20
	2024-11-27	2024-11-27
	2024-10-30	2024-10-30
	2024-07-26	2024-07-26
	2024-06-26	2024-06-26
	2024-06-23	2024-06-23
	2024-05-16	2024-05-16

- Извлеките год и месяц из `transaction_date`.

```
select toYear(transaction_date) AS Year,
       toMonth(transaction_date) AS Month,
       transaction_date
from transactions
```



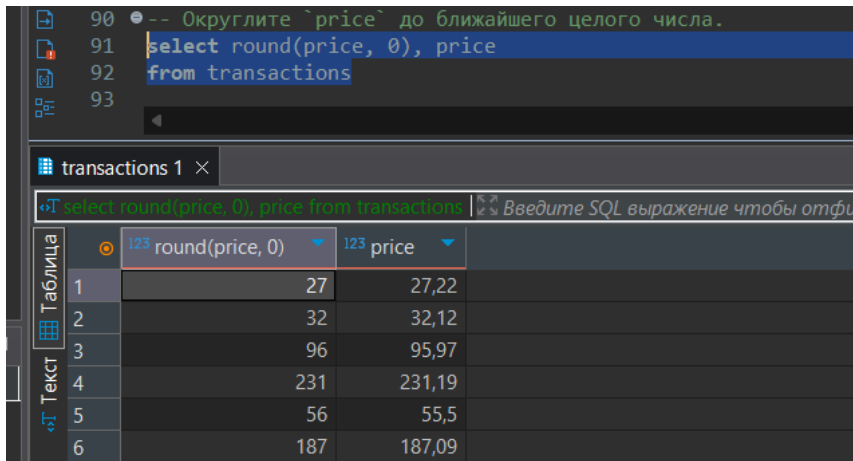
transactions 1 ×

select toYear(transaction_date) AS Year, toMonth(transaction_date) AS Month, transaction_date from transactions

	123 Year	123 Month	transaction_date
1	2 025	3	2025-03-18
2	2 024	12	2024-12-20
3	2 024	11	2024-11-27
4	2 024	10	2024-10-30
5	2 024	7	2024-07-26
6	2 024	6	2024-06-26
7	2 024	6	2024-06-23

- Округлите `price` до ближайшего целого числа.

```
select round(price, 0), price
from transactions
```



transactions 1 ×

select round(price, 0), price from transactions

	123 round(price, 0)	123 price
1	27	27,22
2	32	32,12
3	96	95,97
4	231	231,19
5	56	55,5
6	187	187,09

- Преобразуйте `transaction_id` в строку.

```
select toString(transaction_id), transaction_id
from transactions
```

The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

```
98 -- Преобразуйте `transaction_id` в строку.
99 select toString(transaction_id), transaction_id
100 from transactions
101
```

The results pane shows a table with two columns: `toString(transaction_id)` and `transaction_id`. The table contains four rows of data:

	toString(transaction_id)	transaction_id
1	3598946	3 598 946
2	11179049	11 179 049
3	13218222	13 218 222
4	15642983	15 642 983

- Создайте простую UDF для расчета общей стоимости транзакции. Используйте созданную UDF для расчета общей цены для каждой транзакции.

```
select transaction_sum(price, quantity), price, quantity, transaction_id
from transactions
```

The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

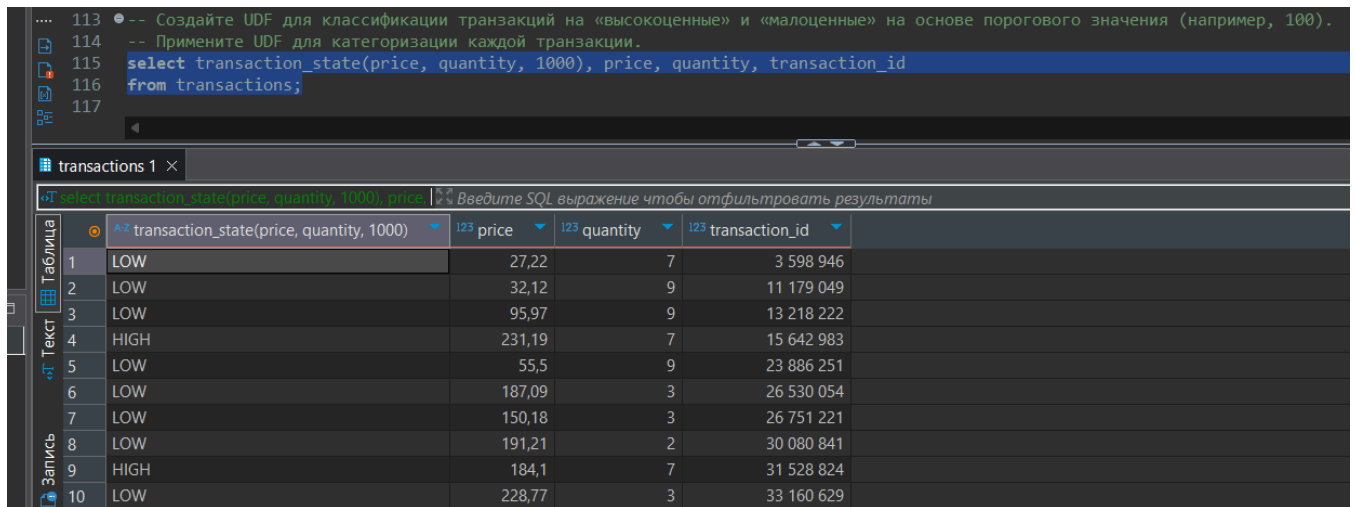
```
107
108 -- Создайте простую UDF для расчета общей стоимости транзакции.
109 -- Используйте созданную UDF для расчета общей цены для каждой транзакции.
110 select transaction_sum(price, quantity), price, quantity, transaction_id
111 from transactions;
112
```

The results pane shows a table with four columns: `transaction_sum(price, quantity)`, `price`, `quantity`, and `transaction_id`. The table contains ten rows of data:

	transaction_sum(price, quantity)	price	quantity	transaction_id
1	190.54	27,22	7	3 598 946
2	289.08	32,12	9	11 179 049
3	863.73	95,97	9	13 218 222
4	1618.33	231,19	7	15 642 983
5	499.5	55,5	9	23 886 251
6	561.27	187,09	3	26 530 054
7	450.54	150,18	3	26 751 221
8	382.42	191,21	2	30 080 841
9	1288.7	184,1	7	31 528 824
10	686.3100000000001	228,77	3	33 160 629

- Создайте UDF для классификации транзакций на «высокоценные» и «малоценные» на основе порогового значения (например, 100). Примените UDF для категоризации каждой транзакции.

```
select transaction_state(price, quantity, 1000), price, quantity, transaction_id
from transactions;
```



	transaction_state(price, quantity, 1000)	price	quantity	transaction_id
1	LOW	27,22	7	3 598 946
2	LOW	32,12	9	11 179 049
3	LOW	95,97	9	13 218 222
4	HIGH	231,19	7	15 642 983
5	LOW	55,5	9	23 886 251
6	LOW	187,09	3	26 530 054
7	LOW	150,18	3	26 751 221
8	LOW	191,21	2	30 080 841
9	HIGH	184,1	7	31 528 824
10	LOW	228,77	3	33 160 629

Создание UDF.

Для использования своих функций нужно:

- добавить конфиг config.xml в папку /etc/clickhouse-server/config.d с содержимым:

```
<clickhouse>
  <user_defined_executable_functions_config>*.xml</user_defined_executable_functions_config>
</clickhouse>
```

- добавить xml с описанием функций в папку /etc/clickhouse-server :

- transaction_state.xml

```
<functions>
  <function>
    <type>executable</type>
    <name>transaction_state</name>
    <return_type>String</return_type>
    <argument>
      <type>Float64</type>
      <name>price</name>
    </argument>
    <argument>
      <type>UInt64</type>
      <name>quantity</name>
    </argument>
    <argument>
      <type>UInt64</type>
      <name>limit</name>
    </argument>
    <format>TabSeparated</format>
    <command>transaction_state.py</command>
    <execute_direct>1</execute_direct>
  </function>
</functions>
```

- transaction_sum.xml

```
<functions>
  <function>
    <type>executable</type>
    <name>transaction_sum</name>
    <return_type>String</return_type>
    <argument>
      <type>Float64</type>
      <name>price</name>
    </argument>
    <argument>
      <type>UInt64</type>
      <name>quantity</name>
    </argument>
    <format>TabSeparated</format>
    <command>transaction_sum.py</command>
    <execute_direct>1</execute_direct>
  </function>
</functions>
```

- положить сами скрипты, написанные на python, в папку /var/lib/clickhouse/user_scripts

- transaction_state.py:

```
#!/usr/bin/python3
import sys

if __name__ == '__main__':
    for line in sys.stdin:
        arg1, arg2, arg3 = line.split('\t')

        result = "";
        if (float(arg1) * int(arg2)) > int(arg3):
            result = "HIGH"    # Высокоценная
        else:
            result = "LOW"     # Малоценная
        print(result)

    sys.stdout.flush()
```

- transaction_sum.py:

```
#!/usr/bin/python3
import sys

if __name__ == '__main__':
    for line in sys.stdin:
        arg1, arg2 = line.split('\t')

        result = float(arg1) * int(arg2)
        print(str(result))

    sys.stdout.flush()
```

Эти манипуляции происходят при сборке docker контейнера. Так же скриптам выставляются флаги для запуска:

```
chmod +x /var/lib/clickhouse/user_scripts/transaction_state.py
chmod +x /var/lib/clickhouse/user_scripts/transaction_sum.py
```


После запуска контейнера можно проверить, подтянулись ли новые функции. Нужно выполнить запрос:

103

104 ● SELECT name, origin

105 FROM system.functions

106 WHERE name in ('transaction_sum','transaction_state');

107

functions 1 ×

SELECT name, origin FROM system.functions WHERE

Введите SQL выражение чтобы от

Таблица

	A-Z name	A-Z origin	
1	transaction_sum	ExecutableUserDefined	
2	transaction_state	ExecutableUserDefined	