可能是史上覆盖flinksql功能最全的demo--part1

该demo基于flnk 1.10版本，由flink大佬fhueske发布到github：https://github.com/fhueske/flink-sql-demo。

动手实践前请先git clone https://github.com/fhueske/flink-sql-demo.git。

由于该demo内容较多，所以文章拆成了2部分，此为第一部分。

场景和数据介绍

此demo主要演示：

- Flink SQL如何处理不同存储系统中的数据
- Flink SQL如何使用Hive Metastore作为外部持久化catalog
- 批流统一查询实践
- 几种Join动态数据的不同方式
- 使用DDL创建表
- 在Kafka和MySQL中使用连续SQL查询维护物化视图

此demo的场景是电商收到订单

该订单系统由7张表组成

- PROD_ORDERS 主订单表
- PROD_LINEITEM: 子订单表
- PROD_CUSTOMER: 用户表
- PROD_NATION: 国家表
- PROD_REGION: 地区表
- PROD_RATES: 货币汇率表
- PROD_RATES_HISTORY: 货币汇率变更历史记录表

根据数据的更新特征（更新频率、insert-only）,上面的表存在不同的系统中：

- 存在KAFKA中的：PROD_ORDERS, PROD_LINEITEM, PROD_RATES_HISTORY
- 存在MySQL中的：PROD_CUSTOMER, PROD_NATION, PROD_REGION, PROD_RATES

数据下载

可以从Google Drive上下载https://drive.google.com/file/d/15LWUBGZenWaW3R_WNxqvcTQOuA_Kgtjv，然后解压到demo的"./data"目录。

部分同学可能下载不方便，我把数据另存了一份在csdn，受csdn文件大小限制，我分成了2个部分，请分别下载后和解压:

part1:https://download.csdn.net/download/cndotaci/12540851

part2:https://download.csdn.net/download/cndotaci/12540856

启动demo

```
# build
docker-compose build
# start
docker-compose up -d
```

验证demo环境是否成功

```
# 验证kafka
docker-compose exec kafka kafka-console-consumer.sh --bootstrap-server k
afka:9092 --from-beginning --topic orders
docker-compose exec kafka kafka-console-consumer.sh --bootstrap-server k
afka:9092 --from-beginning --topic lineitem
docker-compose exec kafka kafka-console-consumer.sh --bootstrap-server k
afka:9092 --from-beginning --topic rates
```

```
# 验证MySQL
docker-compose exec mysql mysql -Dsql-demo -usql-demo -pdemo-sql

SHOW TABLES;
DESCRIBE PROD_CUSTOMER;

```

```
SELECT * FROM PROD_CUSTOMER LIMIT 10;
quit;
```

以上，如果kafka能够消费到数据，MySQL能够查询到数据，说明环境部署成功。

此外，demo还启动了以下服务：

Grafana:http://localhost:3000

Minio:http://localhost:9000 (user: sql-demo, password: demo-sql)

Flink WebUI:http://localhost:8081

Flink SQL client

Hive Metastore

正题开始

启动Flink SQL Client:

```
docker-compose exec sql-client ./sql-client.sh
```

所有的动态表（kafka表）均在默认的catalog中，所有的静态表（MySQL表）均在Hive catalog中：
可以通过命令USE CATALOG xxx语法切换catalog，通过命令SHOW TABLES查看表。

查询静态表和动态表

得益于flink-sql的批流一体功能，有很多查询的sql语法对批处理和流处理是一致的。

对动态表做快照

- 新建一张以S3为存储的表

使用WATERMARK将o_ordertime定义为延时5分钟的eventtime：

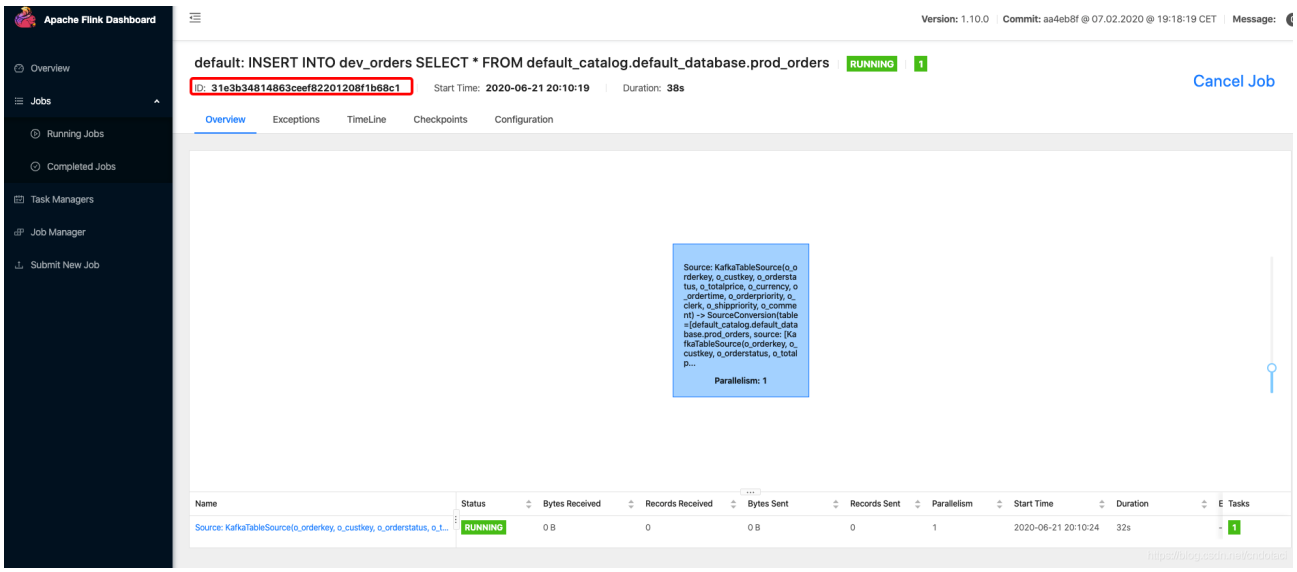
```
USE CATALOG hive;

CREATE TABLE dev_orders (
  o_orderkey      INTEGER,
  o_custkey       INTEGER,
  o_orderstatus   STRING,
  o_totalprice    DOUBLE,
  o_currency      STRING,
  o_ordertime     TIMESTAMP(3),
```

```
  o_orderpriority STRING,
  o_clerk         STRING,
  o_shippriority  INTEGER,
  o_comment       STRING,
  WATERMARK FOR o_ordertime AS o_ordertime - INTERVAL '5' MINUTE
) WITH (
  'connector.type' = 'filesystem',
  'connector.path' = 's3://sql-demo/orders.tbl',
  'format.type' = 'csv',
  'format.field-delimiter' = '|'
);
```

- 从kafka动态表中写一些样例数据到上面创建的S3静态表dev_orders

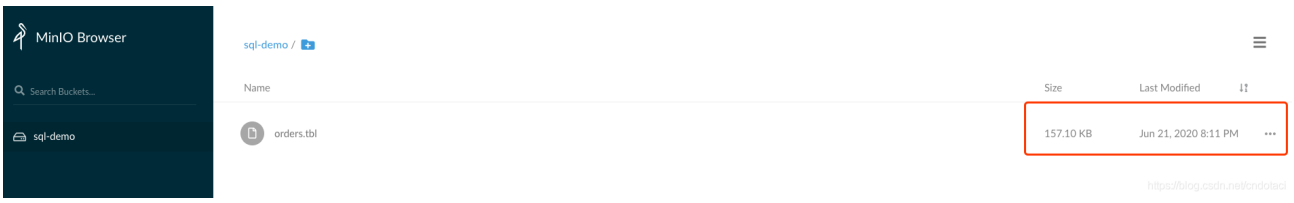
```
Flink SQL> INSERT INTO dev_orders SELECT * FROM default_catalog.default_
database.prod_orders;
[INFO] Submitting SQL update statement to the cluster...
[INFO] Table update statement has been successfully submitted to the clu
ster:
Job ID: 31e3b34814863ceef82201208f1b68c1
```

- 提交INSERT语句后，可以通过Flink Web UI(http://localhost:8081)查看正在运行的任务



- 手动取消任务

- 打开Minio(http://localhost:9000)，可以看到刚刚INSERT操作写入的文件



- 在SQL client中查看刚刚写入的数据

```
SELECT * FROM dev_orders;
SELECT COUNT(*) AS rowCnt FROM dev_orders;
```

查询快照数据

- 使用批处理引擎效率更高

```
SET execution.type=batch;
```

- 计算不同币种每分钟的交易额

```
SELECT
  CEIL(o_ordertime TO MINUTE) AS `minute`,
  o_currency AS `currency`,
  SUM(o_totalprice) AS `revenue`,
  COUNT(*) AS `orderCnt`
```

```
FROM dev_orders
GROUP BY
  o_currency,
  CEIL(o_ordertime TO MINUTE);
```

```
Table program finished.
```

minute	currency	revenue	orderCnt
2020-04-01T00:50	RUB	97369.41	1
2020-04-01T00:50	CAD	530774.66	3
2020-04-01T00:50	USD	146430.33	1
2020-04-01T00:51	CHF	515509.41000000003	4
2020-04-01T00:51	NOK	261720.06000000003	4
2020-04-01T00:51	HKD	345152.28	2
2020-04-01T00:51	CAD	180436.97	3
2020-04-01T00:51	JPY	557195.3099999999	3
2020-04-01T00:51	USD	520958.93	3
2020-04-01T00:51	CNY	526544.27	3
2020-04-01T00:51	RUB	218840.93	1
2020-04-01T00:52	USD	476458.76	4
2020-04-01T00:52	JPY	727244.47	4
2020-04-01T00:52	CAD	565475.65	4
2020-04-01T00:52	HKD	946236.65	6
2020-04-01T00:52	RUB	368969.78	2
2020-04-01T00:52	GBP	113163.09	1
2020-04-01T00:52	CHF	16100.11	1
2020-04-01T00:52	NOK	233480.52	1
2020-04-01T00:52	CNY	354930.25	2
2020-04-01T00:53	CAD	717031.47	3
2020-04-01T00:53	NOK	615417.3300000001	5
2020-04-01T00:53	GBP	740957.13	5
2020-04-01T00:53	RUB	561315.28	3
2020-04-01T00:53	HKD	128389.98	1
2020-04-01T00:53	USD	236770.04	2
2020-04-01T00:53	CHF	397833.66	2
2020-04-01T00:54	CNY	470128.55000000005	2
2020-04-01T00:54	CAD	519370.83	2
2020-04-01T00:54	RUB	265632.11	2
2020-04-01T00:54	GBP	442336.5	3
2020-04-01T00:54	CHF	182288.15	2
2020-04-01T00:54	NOK	136042.84	2
2020-04-01T00:54	HKD	271328.84	1
2020-04-01T00:54	JPY	85246.19	2
2020-04-01T00:54	USD	23874.18	1

- 使用流处理引擎执行上面的查询

```
SET execution.type=streaming;
```

```
Table program finished.
```

minute	currency	revenue	orderCnt
2020-04-01T00:50	USD	146430.33	1
2020-04-01T00:50	NOK	765221.51	5
2020-04-01T00:50	CHF	1016828.3300000001	5
2020-04-01T00:51	JPY	557195.3099999999	3
2020-04-01T00:51	CAD	180436.97	3
2020-04-01T00:51	USD	520958.93	3
2020-04-01T00:51	CHF	515509.41000000003	4
2020-04-01T00:51	HKD	345152.28	2
2020-04-01T00:51	CNY	526544.27	3
2020-04-01T00:51	NOK	261720.06000000003	4
2020-04-01T00:51	RUB	218840.93	1
2020-04-01T00:52	GBP	113163.09	1
2020-04-01T00:52	RUB	368969.78	2
2020-04-01T00:52	JPY	727244.47	4
2020-04-01T00:52	CHF	16100.11	1
2020-04-01T00:52	NOK	233480.52	1
2020-04-01T00:52	HKD	946236.65	6
2020-04-01T00:52	CNY	354930.25	2
2020-04-01T00:52	USD	476458.76	4
2020-04-01T00:52	CAD	565475.65	4
2020-04-01T00:53	CAD	717031.47	3
2020-04-01T00:53	HKD	128389.98	1
2020-04-01T00:53	RUB	561315.28	3
2020-04-01T00:53	NOK	615417.3300000001	5
2020-04-01T00:53	USD	236770.04	2
2020-04-01T00:53	CHF	397833.66	2
2020-04-01T00:53	GBP	740957.13	5
2020-04-01T00:54	CAD	519370.83	2
2020-04-01T00:54	CNY	470128.55000000005	2
2020-04-01T00:54	HKD	271328.84	1
2020-04-01T00:54	JPY	85246.19	2
2020-04-01T00:54	NOK	136042.84	2
2020-04-01T00:54	CHF	182288.15	2
2020-04-01T00:54	RUB	265632.11	2
2020-04-01T00:54	GBP	442336.5	3
2020-04-01T00:54	USD	23874.18	1

可见同样的sql，使用批处理引擎和流处理引擎会得到同样的结果。

- 下面将查询结果的返回格式由table改为changelog

```
SET execution.result-mode=changelog;
```

再次执行上面的sql:

```
Table program finished.
```

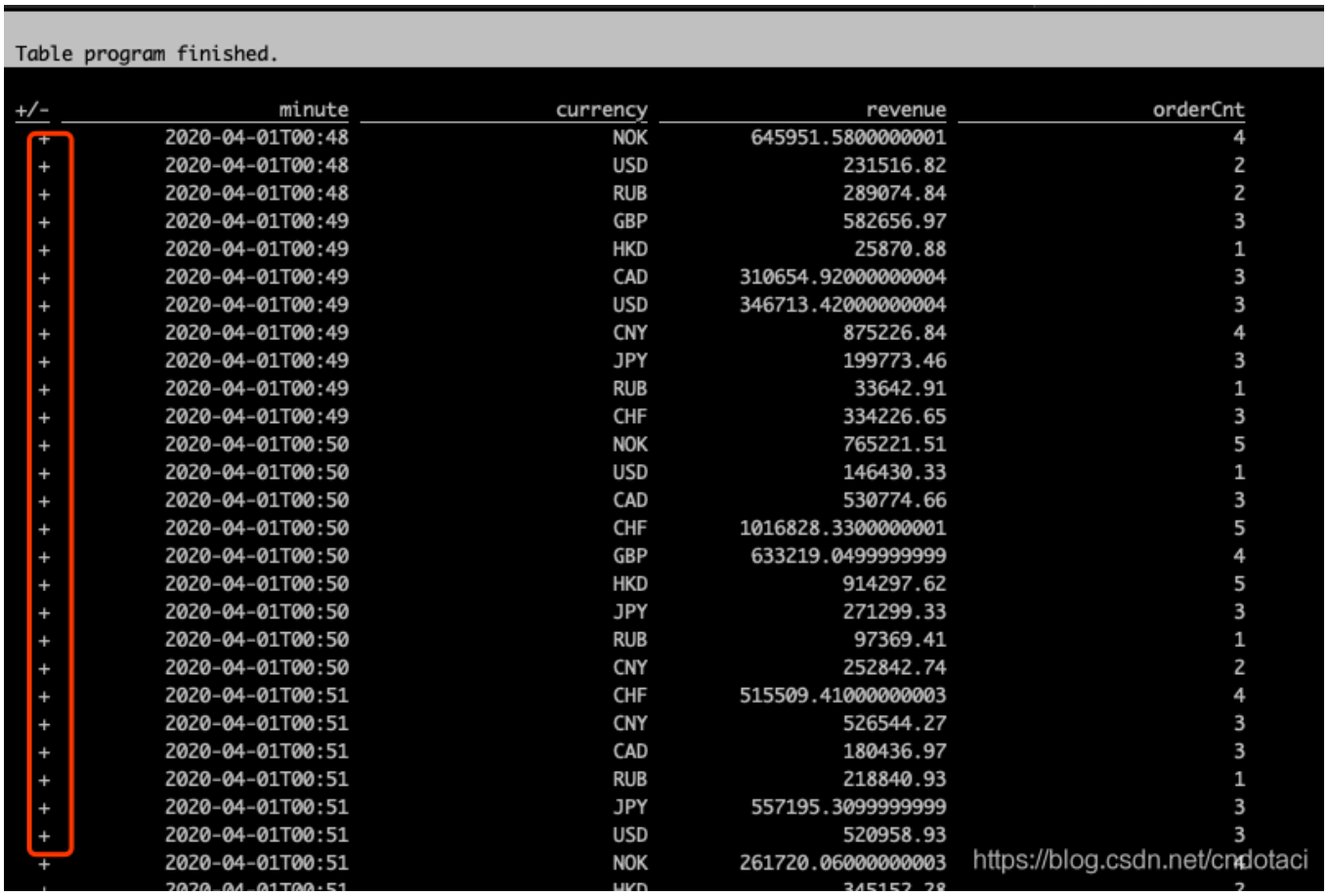
+/-	minute	currency	revenue	orderCnt
+	2020-04-01T00:53	CAD	525872.59	2
-	2020-04-01T00:53	GBP	88232.55	1
+	2020-04-01T00:53	GBP	228424.84000000003	2
-	2020-04-01T00:53	GBP	228424.84000000003	2
+	2020-04-01T00:53	GBP	289361.4	3
-	2020-04-01T00:53	RUB	196662.02	1
+	2020-04-01T00:53	RUB	255641.28	2
-	2020-04-01T00:53	GBP	289361.4	3
+	2020-04-01T00:53	GBP	529097.64	4
-	2020-04-01T00:53	NOK	88336.54	2
+	2020-04-01T00:53	NOK	365963.32	3
-	2020-04-01T00:53	CAD	525872.59	2
+	2020-04-01T00:53	CAD	717031.47	3
+	2020-04-01T00:53	HKD	128389.98	1
+	2020-04-01T00:53	USD	63120.51	1
+	2020-04-01T00:53	CHF	89477.79	1
-	2020-04-01T00:53	RUB	255641.28	2
+	2020-04-01T00:53	RUB	561315.28	3
-	2020-04-01T00:53	NOK	365963.32	3
+	2020-04-01T00:53	NOK	507100.05000000005	4
-	2020-04-01T00:53	NOK	507100.05000000005	4
+	2020-04-01T00:53	NOK	615417.3300000001	5
-	2020-04-01T00:53	USD	63120.51	1
+	2020-04-01T00:53	USD	236770.04	2
-	2020-04-01T00:53	CHF	89477.79	1
+	2020-04-01T00:53	CHF	397833.66	2
-	2020-04-01T00:53	GBP	529097.64	4
+	2020-04-01T00:53	GBP	740957.13	5
+	2020-04-01T00:54	CNY	303066.96	1
+	2020-04-01T00:54	CAD	315007.76	1
+	2020-04-01T00:54	RUB	181668.52	1
+	2020-04-01T00:54	GBP	171612.15	1
-	2020-04-01T00:54	CAD	315007.76	1
+	2020-04-01T00:54	CAD	519370.83	2
-	2020-04-01T00:54	GBP	171612.15	1
+	2020-04-01T00:54	GBP	327482.23	2
-	2020-04-01T00:54	CNY	303066.96	1
+	2020-04-01T00:54	CNY	470128.55000000005	2
+	2020-04-01T00:54	CHF	29448.72	1
+	2020-04-01T00:54	NOK	17957.97	1
+	2020-04-01T00:54	HKD	271328.84	1
+	2020-04-01T00:54	JPY	54501.26	1
-	2020-04-01T00:54	JPY	54501.26	1
+	2020-04-01T00:54	JPY	85246.19	2
-	2020-04-01T00:54	NOK	17957.97	1

可见使用流处理引擎时，结果是通过插入（+）、删除（-）、再插入（+）的方式输出的，也就是流引擎会实时计算每一条数据并输出计算结果，相对而言批处理引擎则将全量数据统一计算，一次性返回计算结果。

- 通过滚动窗口简化上面需求的查询

```
SELECT
  TUMBLE_END(o_ordertime, INTERVAL '1' MINUTE) AS `minute`,
  o_currency AS `currency`,
```

```
  SUM(o_totalprice) AS `revenue`,
  COUNT(*) AS `orderCnt`
FROM dev_orders
GROUP BY
  o_currency,
  TUMBLE(o_ordertime, INTERVAL '1' MINUTE);
```



```
Table program finished.
```

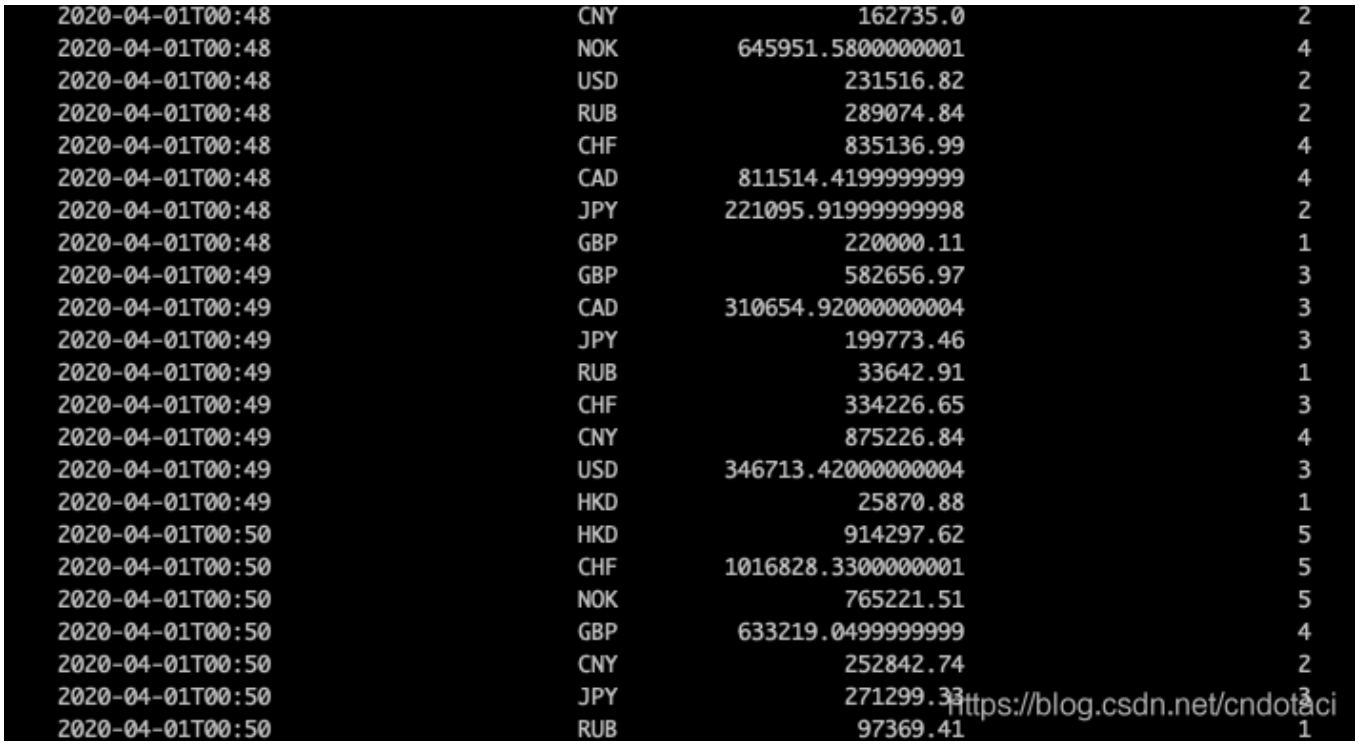
+/-	minute	currency	revenue	orderCnt
+	2020-04-01T00:48	NOK	645951.5800000001	4
+	2020-04-01T00:48	USD	231516.82	2
+	2020-04-01T00:48	RUB	289074.84	2
+	2020-04-01T00:49	GBP	582656.97	3
+	2020-04-01T00:49	HKD	25870.88	1
+	2020-04-01T00:49	CAD	310654.92000000004	3
+	2020-04-01T00:49	USD	346713.42000000004	3
+	2020-04-01T00:49	CNY	875226.84	4
+	2020-04-01T00:49	JPY	199773.46	3
+	2020-04-01T00:49	RUB	33642.91	1
+	2020-04-01T00:49	CHF	334226.65	3
+	2020-04-01T00:50	NOK	765221.51	5
+	2020-04-01T00:50	USD	146430.33	1
+	2020-04-01T00:50	CAD	530774.66	3
+	2020-04-01T00:50	CHF	1016828.3300000001	5
+	2020-04-01T00:50	GBP	633219.0499999999	4
+	2020-04-01T00:50	HKD	914297.62	5
+	2020-04-01T00:50	JPY	271299.33	3
+	2020-04-01T00:50	RUB	97369.41	1
+	2020-04-01T00:50	CNY	252842.74	2
+	2020-04-01T00:51	CHF	515509.41000000003	4
+	2020-04-01T00:51	CNY	526544.27	3
+	2020-04-01T00:51	CAD	180436.97	3
+	2020-04-01T00:51	RUB	218840.93	1
+	2020-04-01T00:51	JPY	557195.3099999999	3
+	2020-04-01T00:51	USD	520958.93	3
+	2020-04-01T00:51	NOK	261720.06000000003	4

可见返回结果中，只有数据插入（+），并没有删除（-），这是因为窗口计算时，flink会等到窗口结束后统一计算窗口内的数据并返回结果，而不会每收到一条数据就计算一次然后通过删除旧数据插入新数据的方式更新。通过这种方式，简化了数据计算的复杂度，提高了性能。

- 同样，上面的窗口计算也可以使用批处理引擎进行

```
SET execution.result-mode=table;
SET execution.type=batch;
```

计算结果与使用流处理引擎一致：



2020-04-01T00:48	CNY	162735.0	2
2020-04-01T00:48	NOK	645951.5800000001	4
2020-04-01T00:48	USD	231516.82	2
2020-04-01T00:48	RUB	289074.84	2
2020-04-01T00:48	CHF	835136.99	4
2020-04-01T00:48	CAD	811514.4199999999	4
2020-04-01T00:48	JPY	221095.91999999998	2
2020-04-01T00:48	GBP	220000.11	1
2020-04-01T00:49	GBP	582656.97	3
2020-04-01T00:49	CAD	310654.92000000004	3
2020-04-01T00:49	JPY	199773.46	3
2020-04-01T00:49	RUB	33642.91	1
2020-04-01T00:49	CHF	334226.65	3
2020-04-01T00:49	CNY	875226.84	4
2020-04-01T00:49	USD	346713.42000000004	3
2020-04-01T00:49	HKD	25870.88	1
2020-04-01T00:50	HKD	914297.62	5
2020-04-01T00:50	CHF	1016828.3300000001	5
2020-04-01T00:50	NOK	765221.51	5
2020-04-01T00:50	GBP	633219.0499999999	4
2020-04-01T00:50	CNY	252842.74	2
2020-04-01T00:50	JPY	271299.33	3
2020-04-01T00:50	RUB	97369.41	1

查询流数据

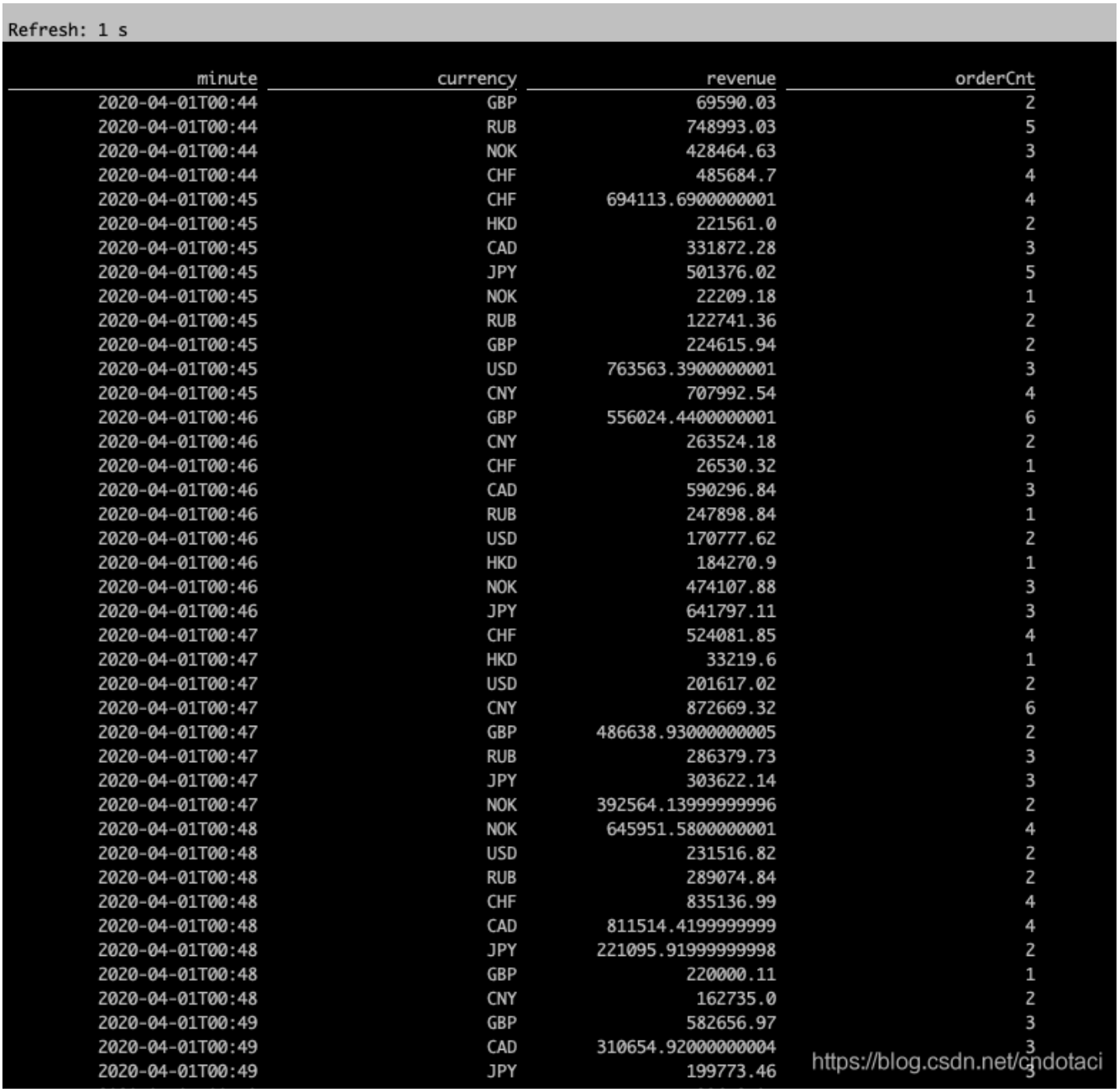
- 在kafka表上执行与上面示例相同的查询

```
SET execution.type=streaming;
```

- 仅修改sql中的表名为kafka表

```
SELECT
  TUMBLE_END(o_ordertime, INTERVAL '1' MINUTE) AS `minute`,
  o_currency AS `currency`,
  SUM(o_totalprice) AS `revenue`,
  COUNT(*) AS `orderCnt`
FROM default_catalog.default_database.prod_orders
GROUP BY
  o_currency,
  TUMBLE(o_ordertime, INTERVAL '1' MINUTE);
```

得到相同的查询结果：



综上，对于静态数据（S3静态表），无论使用批处理引擎还是流处理引擎，同样的sql查询得到的计算结果是一致的。

使用流处理引擎，同样的sql对于静态数据和动态数据（kafka流表）也会得到相同的计算结果，这就是flink流批一体的设计原则——将批数据当做一种有限的流，这样就可以在流和批上共享大部分代码。同时，可以单独对批处理进行特有的优化。