

Laboratory 9

Lab 3

1. Create two topics:

topic item in left part of menu, above ksqlDB.

Add topic --> ConfluentTextLinesTopic --> partitions 10

Add topic --> UppercasedTextLinesTopic --partitions 10

2. Produce some low-case values into ConfluentTextLinesTopic

topic item in left part of menu, above ksqlDB.

choose --> ConfluentTextLinesTopic --> Messages tab --> "+ Produce a new message to this topic"

"alpha" --> Produce

"beta" --> Produce

3. Run KSQL

ksqlDB item in left part of menu, under Topics.

ksqlDB_cluster_0 --> Editor

(or the name of cluster that you have created)

4. Run the commands inside KSQL

```
SET 'auto.offset.reset'='earliest';
```

The screenshot shows the Confluent Cloud interface for running KSQL. The left sidebar has a tree view with 'ksqlDB' selected under 'Topics'. The main area is the 'Editor' tab, showing KSQL code:

```
1 INSERT INTO users (id, gender, name, age) VALUES (0, 'female', 'sarah', 42);
2 INSERT INTO users (id, gender, name, age) VALUES (1, 'male', 'john', 28);
3 INSERT INTO users (id, gender, name, age) VALUES (42, 'female', 'jessica', 70);
4
5 -- Note: The editor is read-only during the ksqlDB Getting Started tutorial
```

Below the code, there's a dropdown for 'auto.offset.reset' set to 'Earliest'. A 'Run query' button is visible. Another JSON command is partially visible at the bottom:

```
1 {
2   "type": "currentStatus",
3   "statementText": "CREATE STREAM USERS (ID INTEGER KEY, GENDER STRING, NAME STRING, AGE INTEGER) WITH
4     (CLEANUP_POLICY='delete', KAFKA_TOPIC='users', KEY_FORMAT='KAFKA', PARTITIONS=1, VALUE_FORMAT='JSON');",
5   "commandId": "stream/USERS/create",
6   "commandStatus": {
7     "status": "PENDING",
8     "message": "Stream created",
9     "sequenceNumber": null
10    },
11   "commandSequenceNumber": 2,
12   "warnings": []
}
```

```
CREATE STREAM ConfluentTextLinesStream (line VARCHAR) WITH (KAFKA_TOPIC='ConfluentTextLinesTopic',
VALUE_FORMAT='KAFKA');
```

The screenshot shows the Confluent KSQL Editor interface. On the left sidebar, under the 'ksqlDB' section, the 'Topics' option is selected. In the main editor area, a single-line query is present:

```
1 CREATE STREAM ConfluentTextLinesStream (line VARCHAR) WITH (KAFKA_TOPIC='ConfluentTextLinesTopic', VALUE_FORMAT='KAFKA');
```

Below the query, there are configuration options for 'Add query properties' (auto.offset.reset set to Earliest), a 'Run query' button, and a message log window showing the response from the server:

```
1 {
  "type": "currentStatus",
  "statementText": "CREATE STREAM ConfluentTextLinesStream (LINE STRING) WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='ConfluentTextLinesTopic', KEY_FORMAT='KAFKA', VALUE_FORMAT='KAFKA')",
  "command": "CREATE STREAM CONFLUENTTEXTLINESSTREAM /create",
  "commandStatus": {
    "status": "SUCCESS",
    "message": "Stream created",
    "queryId": null
  },
  "commandSequenceNumber": 6,
  "warnings": []
}
```

SELECT * FROM ConfluentTextLinesStream EMIT CHANGES;

The screenshot shows the Confluent KSQL Editor interface. The 'Topics' section is selected on the left sidebar. In the main editor area, a query is running:

```
1 SELECT * FROM ConfluentTextLinesStream EMIT CHANGES;
```

The status bar indicates 'Running...'. Below the editor, real-time statistics are displayed for the stream:

- Data structure**: STREAM
- Total messages**: 2 (with a dropdown showing 'LINE': "\beta")
- Messages/sec**: 0
- Total message bytes**: 0 (with a dropdown showing 'LINE': "\alpha")
- Message fields**: LINE

A note states: 'The above statistics are computed for the query source.'

SELECT Ucase(line) FROM ConfluentTextLinesStream EMIT CHANGES;

The screenshot shows the Confluent KSQLDB interface. On the left sidebar, under the 'ksqldb' section, 'Clients' is selected. The main area features an 'Editor' tab where the following KSQL command is entered:

```
1   SELECT Ucase(line) FROM ConfluentTextLinesStream EMIT CHANGES;
```

Below the editor, there are 'Add query properties' settings, specifically 'auto.offset.reset = Earliest'. The status bar indicates 'Running...'. To the right of the editor, there's a monitoring panel with sections for 'Data structure' (STREAM), 'Total messages' (with a dropdown showing '{"KSQL_COL_0": "\BETA"}'), 'Messages/sec' (with a dropdown showing '{"KSQL_COL_0": "\ALPHA"}'), and 'Total message bytes'. A 'Message fields' section is also present.

CREATE STREAM UppercasedTextLinesStream WITH (KAFKA_TOPIC='UppercasedTextLinesTopic', VALUE_FORMAT='KAFKA') AS SELECT Ucase(line) FROM ConfluentTextLinesStream;

The screenshot shows the Confluent KSQLDB interface. On the left sidebar, under the 'ksqldb' section, 'Clients' is selected. The main area features an 'Editor' tab where the following KSQL command is entered:

```
1   CREATE STREAM UppercasedTextLinesStream WITH (KAFKA_TOPIC='UppercasedTextLinesTopic', VALUE_FORMAT='KAFKA') AS SELECT Ucase(line)
```

Below the editor, there are 'Add query properties' settings, specifically 'auto.offset.reset = Earliest'. The status bar indicates 'Stop' and 'Run query'. To the right of the editor, there's a monitoring panel with sections for 'Data structure' (STREAM), 'Total messages' (with a dropdown showing '{"KSQL_COL_0": "\BETA"}'), 'Messages/sec' (with a dropdown showing ' {"KSQL_COL_0": "\ALPHA"}'), and 'Total message bytes'. A 'Message fields' section is also present. On the far right, a sidebar lists 'All available streams and tables' including CONFLUENTTEXTLINESSTREAM, KSQL_PROCESSING_LOG, UPPERCASEDTEXTLINESSTREAM, USERS, and USERS_FILTERED.

5. Check UppercasedTextLinesTopic topic;

SELECT * FROM UppercasedTextLinesStream EMIT CHANGES;

The screenshot shows the Confluent KSQLDB Cluster interface. On the left, a sidebar navigation includes 'Cluster cluster_1' and sections for 'Networking', 'API Keys', 'Cluster Settings', 'Stream Lineage', 'Stream Designer', 'Topics', 'ksqlDB' (which is selected), 'Connectors', and 'Clients'. The main area is titled 'ksqlDB_cluster_0' and contains a query editor with the SQL command: 'SELECT * FROM UppercasedTextLinesStream EMIT CHANGES;'. Below the editor are 'Add_query_properties' and a dropdown for 'auto.offset.reset' set to 'Earliest'. A 'Running...' status bar indicates the query is active. To the right, a monitoring dashboard displays 'Data structure STREAM' with metrics like 'Total messages', 'Messages/sec', 'Total message bytes', and 'Message fields'. A search bar at the top right allows for searching 'All available streams and tables'.

Topics:

Topic name	Partitions	Production (last min)	Consumption (last min)	Schema
ConfluentTextLinesTopic	10	0B/s	0B/s	Set a schema
pksqlc-11n23-processing-log	8	--	--	Set a schema
pksqlc-11n23USERS_FILTERED	1	0B/s	--	Set a schema
UppercasedTextLinesTopic	10	0B/s	0B/s	Set a schema
users	1	0B/s	0B/s	Set a schema

Lab 4

1. Create topic:
readings --> partitions 6

New topic

Topic name* readings

Partitions* 6

Enable infinite retention for this topic

Retain your data in Kafka infinitely to access historical and current data all in one place for data analysis, regulatory purposes, or other use cases.

[Learn more](#)

[Show advanced settings](#)

[Cancel](#) [Create with defaults](#)

2. create stream for topic readings

CREATE STREAM readings (

```
sensor VARCHAR KEY,
val DOUBLE,
location VARCHAR) WITH (
    kafka_topic='readings',
    partitions=6,
    value_format='JSON');
```

Cluster cluster_1

Editor Flow Streams Tables Persistent queries Performance Settings CLI instructions

```
CREATE STREAM readings (
    sensor VARCHAR KEY,
    val DOUBLE,
    location VARCHAR) WITH (
        kafka_topic='readings',
        partitions=6,
        value_format='JSON');
```

[Add query properties](#)

auto.offset.reset = Earliest

+ Add another field

Stop Run query

All available streams and tables

- CONFLUENTTEXTLINESSTREAM
- KSQL_PROCESSING_LOG
- READINGS
- UPPERCASEDTEXTLINESSTREAM
- USERS
- USERS_FILTERED

3. Produce some events into readings topic

```
INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 45, 'wheel');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-2', 41, 'motor');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 42, 'wheel');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 42, 'muffler');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 48, 'muffler');
```

The screenshot shows the Confluent KSQLDB interface. On the left, a sidebar lists clusters: 'cluster_1' (selected), 'cluster_2', 'cluster_3', 'cluster_4', 'cluster_5', and 'cluster_6'. Under 'cluster_1', there are links for Cluster Overview, Networking, API Keys, Cluster Settings, Stream Lineage, Stream Designer, Topics, and 'ksqldb' (selected). Below these are Connectors, Clients, Schema Registry, and CLI and Tools.

The main area has tabs: Editor (selected), Flow, Streams, Tables, Persistent queries, Performance, Settings, and CLI instructions. The Editor tab contains a code editor with the following SQL query:

```

1 | INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 45, 'wheel');
2 | INSERT INTO readings (sensor, val, location) VALUES ('sensor-2', 41, 'motor');
3 | INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 42, 'wheel');
4 | INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 42, 'muffler');
5 | INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 48, 'muffler');
6 |

```

Below the code editor are sections for 'Add query properties' (auto.offset.reset = Earliest) and '+Add another field'. To the right are 'Stop' and 'Run query' buttons. A message browser on the right shows 'No new messages' with a note: 'The message browser shows messages that have arrived since this page was opened.'

4. Run some queries

CREATE STREAM clean AS SELECT sensor, val, UCASE(location) as location FROM readings EMIT CHANGES;

ksqldb_cluster_0

The screenshot shows the KSQLDB interface with the title 'ksqldb_cluster_0'. The top navigation bar includes tabs: Editor (selected), Flow, Streams, Tables, Persistent queries, Performance, Settings, and CLI instructions.

The Editor tab contains a code editor with the following SQL query:

```

1 | CREATE STREAM clean AS SELECT sensor, val, UCASE(location) as location FROM readings EMIT CHANGES;

```

Below the code editor are sections for 'Add query properties' (auto.offset.reset = Earliest) and '+Add another field'. To the right are 'Stop' and 'Run query' buttons.

Below the editor is a JSON response block containing the following data:

```

1 | {
2 |   "type": "currentStatus",
3 |   "statementText": "CREATE STREAM CLEAN WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='pksqlc-11n23CLEAN', PARTITIONS=6, REPLICAS=3, RETENTION_MS=604800000) AS SELECT\n  READINGS.SENSOR SENSOR,\n  READINGS.VAL VAL,\n  UCASE(READINGS.LOCATION)\n LOCATION\nFROM READINGS\nEMIT CHANGES;",
4 |   "commandId": "stream/'CLEAN'/create",
5 |   "commandStatus": {
6 |     "status": "SUCCESS",
7 |     "message": "Created query with ID CSAS_CLEAN_15",
8 |     "queryId": "CSAS_CLEAN_15"
9 |   },
10 |   "commandSequenceNumber": 16,
11 |   "warnings": []
12 |

```

```
SELECT sensor, val, location FROM readings WHERE sensor='sensor-1' EMIT CHANGES;
```

Editor Flow Streams Tables Persistent queries Performance Settings CLI instructions

1 | `SELECT sensor, val, location FROM readings WHERE sensor='sensor-1' EMIT CHANGES;`

● [Add query properties](#)

`auto.offset.reset` = Earliest

[+Add another field](#)

Running...

Data structure
STREAM

Total messages
--

Messages/sec
--

Total message bytes
--

Message fields

The above statistics are computed for the dauer source.

Filter by keyword

▼ {"SENSOR":"sensor-1","VAL":42,"LOCATION":"wheel"}
▼ {"SENSOR":"sensor-1","VAL":45,"LOCATION":"wheel"}

```
CREATE STREAM high_readings AS SELECT sensor, val, location FROM clean where val > 42 EMIT CHANGES;
```

ksqlDB_cluster_0

Editor Flow Streams Tables Persistent queries Performance Settings CLI instructions

```
1 CREATE STREAM high_readings AS SELECT sensor, val, location FROM clean WHERE val > 42 EMIT CHANGES;
```

● Add query properties

auto.offset.reset = Earliest

+Add another field

Stop

Run query

```
1 {
2   "@type": "currentStatus",
3   "statementText": "CREATE STREAM HIGH_READINGS WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='pksqlc-11n23HIGH_READINGS', PARTITIONS=6, REPLICAS=3, RETENTION_MS=604800000) AS SELECT\n  CLEANSENSOR SENSOR,\n  CLEANVAL VAL,\n  CLEANLOCATION LOCATION\nFROM CLEAN\nWHERE (CLEANVAL > 42)\nEMIT CHANGES;",
4   "commandId": "stream/'HIGH_READINGS'/create",
5   "commandStatus": {
6     "status": "SUCCESS",
7     "message": "Created query with ID CSAS_HIGH_READINGS_17",
8     "queryId": "CSAS_HIGH_READINGS_17"
9   },
10   "commandSequenceNumber": 18,
11   "warnings": []
12 }
```

5. Produce some events into readings topic

INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 36, 'motor');

Editor Flow Streams Tables Persistent queries Performance Settings CLI instructions

```
1 INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 36, 'motor');
```

● Add query properties

6. Run some queries

CREATE STREAM high_pri AS SELECT sensor, val, UCASE(location) as location
FROM readings WHERE val > 42 EMIT CHANGES;

```

1 CREATE STREAM high_pri AS SELECT sensor, val, UCASE(location) as location
2   FROM readings where val > 42 EMIT CHANGES;
3

```

● Add query properties

auto.offset.reset = Earliest

+Add another field

Stop

Run query

```

1 {
2   "@type": "currentStatus",
3   "statementText": "CREATE STREAM HIGH_PRI WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='pkqlc-11n23HIGH_PRI', PARTITIONS=6,
REPLICAS=3, RETENTION_MS=604800000) AS SELECT\n  READINGS.SENSOR SENSOR,\n  READINGS.VAL VAL,\n  UCASE(READINGS.LOCATION)
LOCATION\nFROM READINGS READINGS\nWHERE (READINGS.VAL > 42)\nEMIT CHANGES;",
4   "commandId": "stream/HIGH_PRI/create",
5   "commandStatus": {
6     "status": "SUCCESS",
7     "message": "Created query with ID CSAS_HIGH_PRI_19",
8     "queryId": "CSAS_HIGH_PRI_19"
9   },
10  "commandSequenceNumber": 20,
11  "warnings": []
12 }

```

CREATE STREAM by_location AS SELECT * FROM high_pri
PARTITION BY location EMIT CHANGES;

```

1 CREATE STREAM by_location AS SELECT * FROM high_pri
2   PARTITION BY location EMIT CHANGES;
3
4

```

● Add query properties

auto.offset.reset = Earliest

+Add another field

Stop

Run query

```

1 {
2   "@type": "currentStatus",
3   "statementText": "CREATE STREAM BY_LOCATION WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='pkqlc-11n23BY_LOCATION',
PARTITIONS=6, REPLICAS=3, RETENTION_MS=604800000) AS SELECT *\nFROM HIGH_PRI HIGH_PRI\nPARTITION BY HIGH_PRI.LOCATION\nEMIT
CHANGES;",
4   "commandId": "stream/BY_LOCATION/create",
5   "commandStatus": {
6     "status": "SUCCESS",
7     "message": "Created query with ID CSAS_BY_LOCATION_21",
8     "queryId": "CSAS_BY_LOCATION_21"
9   },
10  "commandSequenceNumber": 22,
11  "warnings": []
12 }

```

CREATE STREAM by_val AS SELECT * FROM high_pri
PARTITION BY val EMIT CHANGES;

```
1 CREATE STREAM by_val AS SELECT * FROM high_pri
2 PARTITION BY val EMIT CHANGES;
3
4
5
```

● Add query properties

auto.offset.reset = Earliest  

+Add another field

Stop

Run query

```
1 {
2   "@type": "currentStatus",
3   "statementText": "CREATE STREAM BY_VAL WITH (CLEANUP_POLICY='delete', KAFKA_TOPIC='pksqlc-11n23BY_VAL', PARTITIONS=6,
4   REPLICAS=3, RETENTION_MS=604800000) AS SELECT *\\nFROM HIGH_PRI HIGH_PRI\\nPARTITION BY HIGH_PRI.VAL\\nEMIT CHANGES;",
5   "commandId": "stream/`BY_VAL`/create",
6   "commandStatus": {
7     "status": "SUCCESS",
8     "message": "Created query with ID CSAS_BY_VAL_23",
9     "queryId": "CSAS_BY_VAL_23"
10   },
11   "commandSequenceNumber": 24,
12   "warnings": []
}
```

Lab 5

1. Considering the readings topic is already created
readings --> partitions 6

2. Produce some events into readings topic

```
INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 45, 'wheel');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-2', 41, 'motor');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 42, 'wheel');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 42, 'muffler');
INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 48, 'muffler');
```

```
1 INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 45, 'wheel');
2 INSERT INTO readings (sensor, val, location) VALUES ('sensor-2', 41, 'motor');
3 INSERT INTO readings (sensor, val, location) VALUES ('sensor-1', 42, 'wheel');
4 INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 42, 'muffler');
5 INSERT INTO readings (sensor, val, location) VALUES ('sensor-3', 48, 'muffler');
6
7
8
9
```

● [Add query properties](#)

auto.offset.reset = Earliest  

+Add another field

Stop

Run query

Data structure   Filter by keyword 

Total messages  

Messages/sec  

No new messages
The message browser shows messages that have arrived since this page was opened.

Total message bytes  

Message fields

The above statistics are computed for the query source.

3. Create KTable:

```
CREATE TABLE avg_readings AS
    SELECT sensor, AVG(val) as avg
    FROM readings
    GROUP BY sensor
    EMIT CHANGES;
```

```
1 CREATE TABLE avg_readings AS
2     SELECT sensor, AVG(val) as avg
3     FROM readings
4     GROUP BY sensor
5     EMIT CHANGES;
6
7
8
9
```

● Add query properties

auto.offset.reset = Earliest

+Add another field

Stop

Run query

```
1 {
2     "@type": "currentStatus",
3     "statementText": "CREATE TABLE AVG_READINGS WITH (CLEANUP_POLICY='compact', KAFKA_TOPIC='pksqlc-11n23AVG_READINGS',
4     PARTITIONS=6, REPLICAS=3, RETENTION_MS=604800000) AS SELECT\n    READINGS.SENSOR SENSOR,\n    AVG(READINGS.VAL) AVG\nFROM READINGS\nGROUP BY READINGS.SENSOR\nEMIT CHANGES;",
5     "commandId": "table/'AVG_READINGS'/create",
6     "commandStatus": {
7         "status": "SUCCESS",
8         "message": "Created query with ID CTAS_AVG_READINGS_25",
9         "queryId": "CTAS_AVG_READINGS_25"
10    },
11    "commandSequenceNumber": 26,
12    "warnings": []
}
```

```
CREATE TABLE part_avg AS
    SELECT location, AVG(val) as avg
    FROM readings
    GROUP BY location
    EMIT CHANGES;
```

```
1 CREATE TABLE part_avg AS
2     SELECT location, AVG(val) as avg
3     FROM readings
4     GROUP BY location
5     EMIT CHANGES;
6
7
8
9
10
```

● Add query properties

auto.offset.reset = Earliest

+Add another field

Stop

Run query

```
1 {
2     "@type": "currentStatus",
3     "statementText": "CREATE TABLE PART_AVG WITH (CLEANUP_POLICY='compact', KAFKA_TOPIC='pksqlc-11n23PART_AVG', PARTITIONS=6,
4     REPLICAS=3, RETENTION_MS=604800000) AS SELECT\n    READINGS.LOCATION LOCATION,\n    AVG(READINGS.VAL) AVG\nFROM READINGS\nGROUP BY READINGS.LOCATION\nEMIT CHANGES;",
5     "commandId": "table/'PART_AVG'/create",
6     "commandStatus": {
7         "status": "SUCCESS",
8         "message": "Created query with ID CTAS_PART_AVG_27",
9         "queryId": "CTAS_PART_AVG_27"
10    },
11    "commandSequenceNumber": 28,
12    "warnings": []
}
```