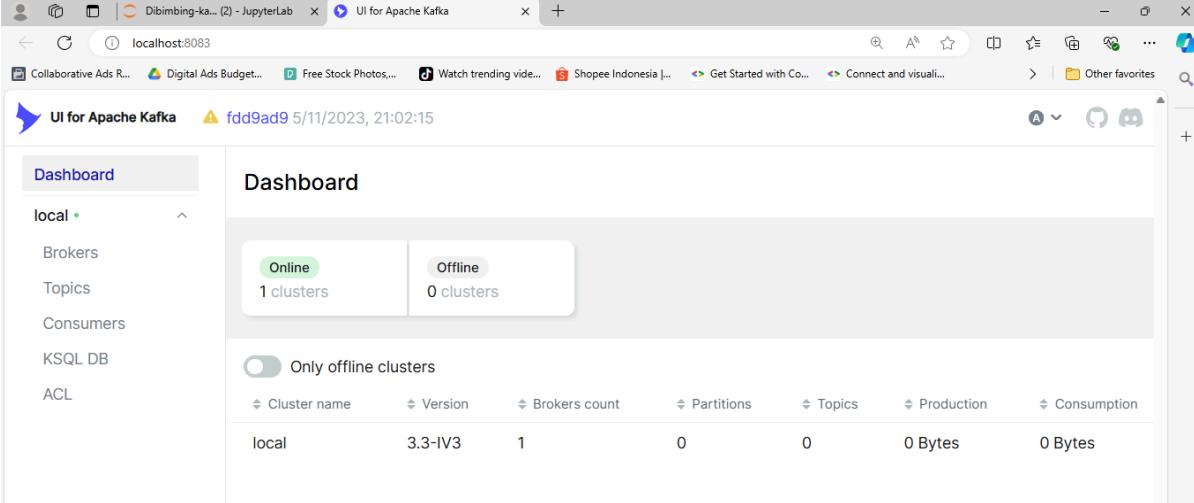


Exercise - 25 Agustus 2024 - Hijir

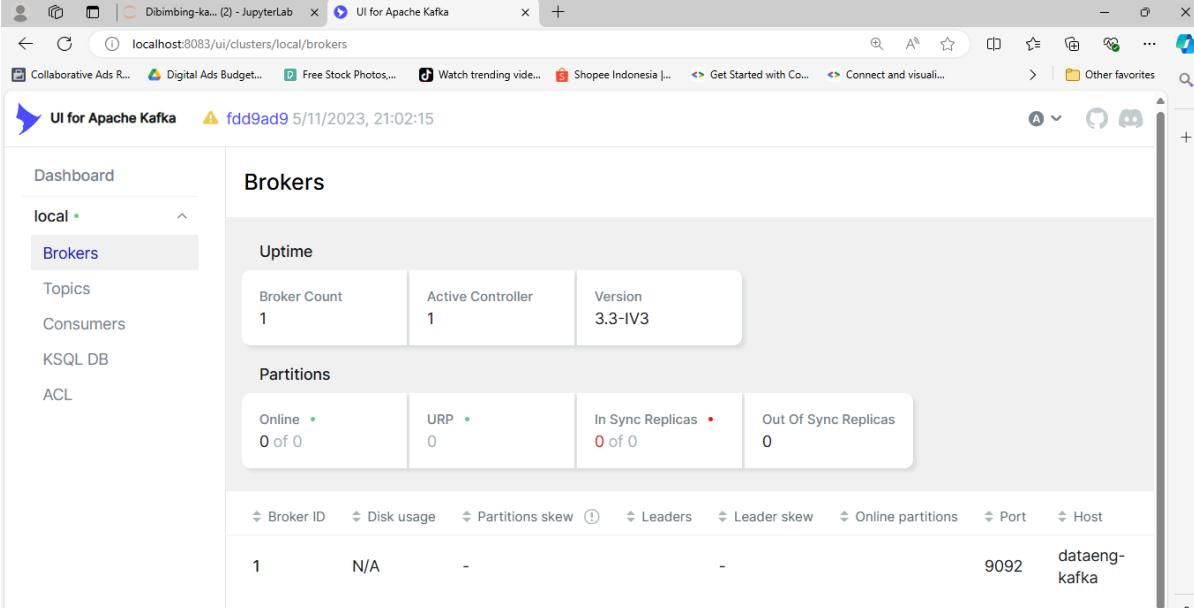
UI Kafka



The screenshot shows the Kafka dashboard interface. On the left, a sidebar menu lists 'Dashboard', 'Brokers', 'Topics', 'Consumers', 'KSQL DB', and 'ACL'. The 'local' cluster is selected. The main area displays a summary of clusters: 1 Online cluster and 0 Offline clusters. Below this, a table provides detailed information about the local cluster, including its name, version (3.3-IV3), broker count (1), partitions (0), topics (0), production (0 Bytes), and consumption (0 Bytes).

Cluster name	Version	Brokers count	Partitions	Topics	Production	Consumption
local	3.3-IV3	1	0	0	0 Bytes	0 Bytes

Broker



The screenshot shows the brokers page of the Kafka UI. The sidebar menu is identical to the dashboard, with 'Brokers' now selected. The main area displays 'Uptime' metrics: Broker Count (1), Active Controller (1), and Version (3.3-IV3). Below this, a table shows partition statistics: Online (0 of 0), URP (0), In Sync Replicas (0 of 0), and Out Of Sync Replicas (0). A detailed table for the single broker shows fields like Broker ID (1), Disk usage (N/A), Partitions skew (-), Leaders (-), Leader skew (-), Online partitions (-), Port (9092), and Host (dataeng-kafka).

Broker ID	Disk usage	Partitions skew	Leaders	Leader skew	Online partitions	Port	Host
1	N/A	-	-	-	-	9092	dataeng-kafka

Jupyter Kafka 1 - untuk producer

The screenshot shows a Jupyter Notebook interface with two files open: 'Dibimbing-kafka-1.ipynb' and 'Dibimbing-spark.ipynb'. The code in 'Dibimbing-kafka-1.ipynb' is as follows:

```
[1]: import json
import uuid
import os
import json
from dotenv import load_dotenv
from pathlib import Path
from kafka import KafkaProducer
from faker import Faker
from time import sleep

[2]: dotenv_path = Path('/resources/.env')
load_dotenv(dotenv_path=dotenv_path)

[3]: kafka_host = os.getenv('KAFKA_HOST')
kafka_topic = os.getenv('KAFKA_TOPIC_NAME')
kafka_topic_partition = os.getenv('KAFKA_TOPIC_NAME') + "-1"

[4]: print(kafka_host)
dateng-kafka

[5]: producer = KafkaProducer(bootstrap_servers=f'{kafka_host}:9092')

[6]: _instance = Faker()
global faker
```

Membuat koneksi produser

The screenshot shows a Jupyter Notebook interface with two files open: 'Dibimbing-ka... (2) - JupyterLab' and 'Dibimbing-kafka-1.ipynb'. The code in 'Dibimbing-kafka-1.ipynb' is as follows:

```
[6]: _instance = Faker()
global Faker
faker = Faker()

[*]: class DataGenerator(object):
    @staticmethod
    def get_data():
        return [
            uuid.uuid4().__str__(),
            faker.name(),
            faker.random_element(elements=['IT', 'HR', 'Sales', 'Marketing']),
            faker.random_element(elements=['CA', 'NV', 'TX', 'FL', 'IL', 'WA']),
            faker.random_int(min=10000, max=150000),
            faker.random_int(min=18, max=60),
            faker.random_int(min=0, max=100000),
            faker.unique_time()
        ]

    for i in range(1,400):
        columns = ["emp_id", "employee_name", "department", "state", "salary", "age", "bonus", "ts", "new"]
        data_list = DataGenerator.get_data()
        json_data = dict(
            zip(columns,data_list)
        )
        payload = json.dumps(json_data).encode("utf-8")
        response = producer.send(topic=kafka_topic_partition, value=payload)
        # response = producer.send(topic=kafka_topic_partition, value=payload, partition=2)
        print(json_data['emp_id'],response.get())
        sleep(5)
```

Membuat data faker ke kafka

The screenshot shows the 'Topics' page of the UI for Apache Kafka. On the left sidebar, under the 'local' cluster, 'Topics' is selected. The main area displays a table for 'test-topic-1'. The table has columns: Topic Name, Partitions, Out of sync replicas, Replication Factor, Number of messages, and Size. The data row for 'test-topic-1' shows 1 partition, 0 replicas, 1 replication factor, 29 messages, and 6 KB in size.

Event sudah terkirim ke kafka yaitu test-topic-1

The screenshot shows the 'Messages' page for 'test-topic-1'. The top navigation bar includes tabs for Overview, Messages, Consumers, Settings, and Statistics. The 'Messages' tab is active. The main area shows a table of messages with columns: Offset, Partition, Timestamp, Key, Preview, Value, and Preview. There are 6 messages listed, each with its offset, partition, timestamp, key, and a preview of the JSON value. The messages were produced 221 ms ago and are 10 KB in size, with 58 messages consumed.

Hasil dapat dilihat pada messages: berarti kita sudah berhasil menulis event ke kafka

The screenshot shows the 'Messages' page for 'test-topic-1', focusing on the first message at offset 0. The table columns are Key, Value, Headers, Timestamp, Key Serde, and Value Serde. The 'Value' column shows a JSON object representing a faker-generated employee record. The 'Headers' section is collapsed. The 'Timestamp' is 9/9/2024, 20:06:34. The 'Key Serde' is String and 'Value Serde' is String.

Contoh salah satu data faker yang berhasil ditulis

Jupyter Kafka 2 - untuk consumer

The image shows two side-by-side Jupyter Notebook interfaces. Both have the title 'Dibimb... (2) - JupyterLab' and 'UI for Apache Kafka'.

The left notebook (Dibimb...-1.ipynb) contains the following code:

```
[2]: import json
import uuid
import os
import json
from dotenv import load_dotenv
from pathlib import Path
from kafka import KafkaProducer
from faker import Faker
from time import sleep

[3]: dotenv_path = Path('/resources/.env')
load_dotenv(dotenv_path=dotenv_path)

[3]: True

[4]: kafka_host = os.getenv('KAFKA_HOST')
kafka_topic = os.getenv('KAFKA_TOPIC_NAME')
kafka_topic_partition = os.getenv('KAFKA_TOPIC_NAME') + "-1"

[6]: from kafka import KafkaConsumer

# To consume latest messages and auto-commit offsets
consumer = KafkaConsumer(kafka_topic_partition,
                          group_id='dibimb...-group',
                          bootstrap_servers=[f'{kafka_host}:9092'])
```

The right notebook (Dibimb...-2.ipynb) contains the following code:

```
[6]: from kafka import KafkaConsumer

# To consume latest messages and auto-commit offsets
consumer = KafkaConsumer(kafka_topic_partition,
                          group_id='dibimb...-group',
                          bootstrap_servers=[f'{kafka_host}:9092'])

[*]: for message in consumer:
    print ("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,
                                             message.offset, message.key,
                                             message.value))

test-topic-1:0:131: key=None value=b'{"emp_id": "78126090-79bc-441e-9bcb-03382c2e8f6d", "employee_name": "Adam Gilbert", "department": "IT", "state": "RJ", "salary": 70653, "age": 34, "bonus": 94980, "ts": 334793406}'
test-topic-1:0:132: key=None value=b'{"emp_id": "a1dea844-8920-4081-92eb-4c9a8d4aeca3", "employee_name": "Peggy Hernandez", "department": "IT", "state": "NY", "salary": 66083, "age": 51, "bonus": 44362, "ts": 1120085623}'
```

Membaca data dari producer

The screenshot shows the 'UI for Apache Kafka' interface at the URL localhost:8083/ui/clusters/local/consumer-groups/dibimb...-group.

The sidebar on the left has the following navigation items:

- Dashboard
- local
- Brokers
- Topics
- Consumers** (selected)
- KSQL DB
- ACL

The main content area displays information for the consumer group 'dibimb...-group':

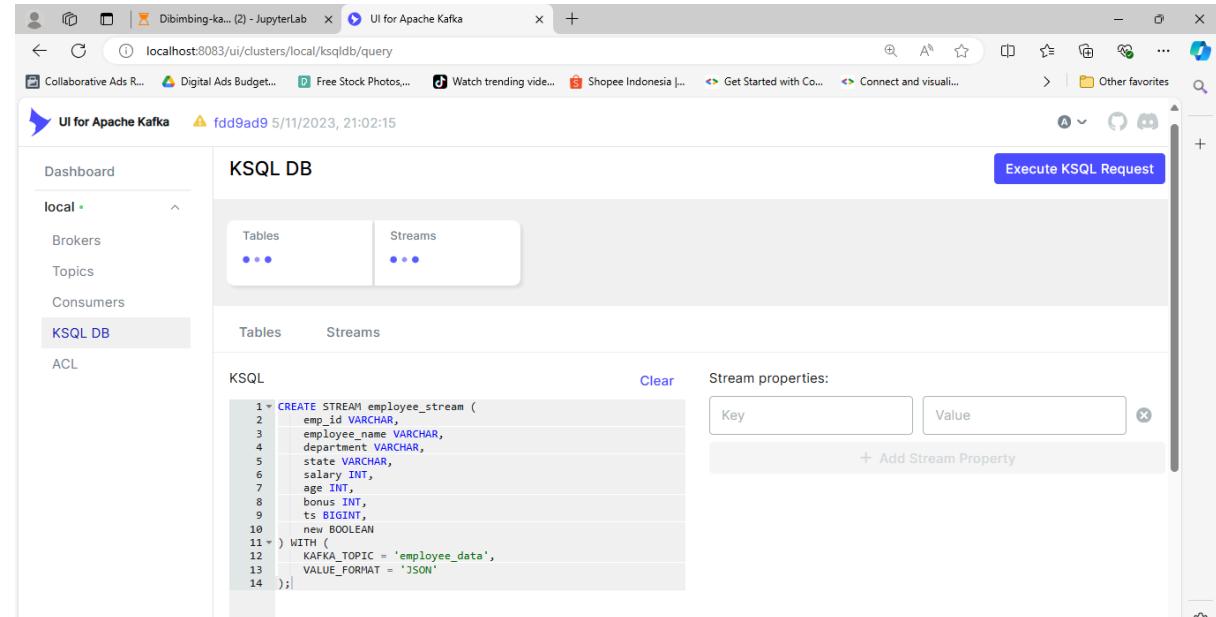
State	Members	Assigned Topics	Assigned Partitions	Coordinator ID	Total lag
STABLE	1	1	1	1	0

A search bar at the bottom of the table header says 'Search by Topic Name'. Below the table, there is a table with columns 'Topic' and 'Messages behind'.

Topic	Messages behind
test-topic-1	0

KsqlDb

Melakukan agregrasi di kafka menggunakan sql



```
CREATE STREAM employee_stream (
    emp_id VARCHAR,
    employee_name VARCHAR,
    department VARCHAR,
    state VARCHAR,
    salary INT,
    age INT,
    bonus INT,
    ts BIGINT,
    new BOOLEAN
) WITH (
    KAFKA_TOPIC = 'test-topic-1',
    VALUE_FORMAT = 'JSON'
);
```

localhost:8083/ui/clusters/local/ksqldbstreams

UI for Apache Kafka

fdd9ad9 5/11/2023, 21:02:15

Dashboard local +

Brokers Topics Consumers KSQL DB ACL

KSQL DB

Tables Streams

Name	Topic	Key Format	Value Format	Is Windowed
EMPLOYEE_STREAM	test-topic-1	KAFKA	JSON	-
KSQ_LPROCESSING_LOG	default_ksql_processing_log	KAFKA	JSON	-

Execute KSQL Request

localhost:8083/ui/clusters/local/ksqldb/query

UI for Apache Kafka

fdd9ad9 5/11/2023, 21:02:15

Dashboard local +

Brokers Topics Consumers KSQL DB ACL

KSQL DB

Tables Streams

KSQ L

```
1 SELECT * FROM EMPLOYEE_STREAM EXIT CHANGES;
```

Stream properties:

Key Value

+ Add Stream Property

localhost:8083/ui/clusters/local/ksqldb/query

UI for Apache Kafka

fdd9ad9 5/11/2023, 21:02:15

Dashboard local +

Brokers Topics Consumers KSQL DB ACL

Schema

'EMP_ID'	STRING	'EMPLOYEE_NAME'	STRING	'DEPARTMENT'	STRING	'STATE'	STRING	'SALARY'	INTEGER	'AGE'	INTEGER	'BONUS'	INTEGER	'TS'	BIGINT	'NEW'	BOOLEAN
e3e4da12-c292-41ce-a61c-4312cc571b14		Vanessa Thomas		IT		CA		103114	40	81508		1251016071		null			
63550edf-5c7a-465b-8e25-3e1becfa65a3		Paula Harris		IT		RJ		82308	46	83662		1486438710		null			
55d137a2-4953-44e4-af71-89d5b0d685ef		Belinda Turner		Marketing		NY		95935	44	75661		89763512		null			
43e623a4-8d99-4884-b8f8-8b2f00cedfd7		James Kelly		Sales		FL		38817	53	60670		1032031982		null			
d38d1282-b227-		Patrick Garcia		HR		FL		82471							Consuming query execution result... Abort		

localhost:8083/ui/clusters/local/ksqldb/query

UI for Apache Kafka

fdd9ad9 5/11/2023, 21:02:15

Dashboard local KSQL DB Streams

Tables Streams

KSQ

```
1 CREATE TABLE avg_salary_table_2 AS
2 SELECT
3   department,
4   AVG(salary) AS avg_salary
5   FROM employee_stream
6   GROUP BY department
7   EMIT CHANGES;
```

Execute KSQL Request

localhost:8083/ui/clusters/local/ksqldb/db

UI for Apache Kafka

fdd9ad9 5/11/2023, 21:02:15

Dashboard local KSQL DB Streams

Tables Streams

AVG_SALARY_TABLE_2 AVG_SALARY_TABLE_2 KAFKA JSON false

Execute KSQL Request

localhost:8083/ui/clusters/local/ksqldb/query

UI for Apache Kafka

fdd9ad9 5/11/2023, 21:02:15

Dashboard local KSQL DB Streams

Tables Streams

KSQ

```
1 SELECT * FROM AVG_SALARY_TABLE_2
2 EMIT CHANGES;
```

Stream properties:

Key Value

+ Add Stream Property

Execute KSQL Request

The screenshot shows the UI for Apache Kafka interface. On the left, there's a sidebar with navigation links: Dashboard, local (selected), Brokers, Topics, Consumers, KSQL DB (selected), and ACL. The main area is titled "Schema" and displays a table of departmental average salaries:

'DEPARTMENT'	STRING	'AVG_SALARY'	DOUBLE
HR		67457.5	
Sales		83022.07692307692	
Marketing		100812	
IT		49006.2	
IT		52437.09090909091	
HR		66885.11111111111	
Marketing		94854.6	
Sales		84497.642	Consuming query execution result... Abort
IT		56035.083333333336	

Rata-rata gaji diupdate setiap ada data baru yang masuk

Homework - Kafka - Hijir

Without setup new docker - just in case the new docker kafka setup fails

The screenshot shows a Jupyter Notebook interface with three tabs: "Dibimbing-kafka-1.ipynb", "hijir-kafka-producer.ipynb" (selected), and "Dibimbing-kafka-2.ipynb". The code cell [7] contains the following Python code:

```

import json
import uuid
import os
import json
from dotenv import load_dotenv
from pathlib import Path
from kafka import KafkaProducer
from faker import Faker
from time import sleep

```

The code cell [8] contains:

```

# Using the variables in .env file
dotenv_path = Path('/resources/.env')
load_dotenv(dotenv_path=dotenv_path)

```

The code cell [9] contains:

```

kafka_host = os.getenv('KAFKA_HOST')

```

The code cell [10] contains:

```

producer = KafkaProducer(bootstrap_servers=f'{kafka_host}:9092')

```

Jupyter ‘hijir-kafka-producer’

The screenshot shows a JupyterLab environment. On the left, there is a file browser with a sidebar for filtering files by name. The main area displays two tabs: 'Dibimbing-kafka-1.ipynb' and 'hijir-kafka-producer.ipynb'. The 'hijir-kafka-producer.ipynb' tab is active, showing Python code in a code cell. The code defines a class 'DataGenerator' that uses the Faker library to generate transaction data. The code cell is numbered [13].

```
[11]: __instance = Faker()
global faker
faker = Faker()

[13]: class DataGenerator(object):
    @staticmethod
    def get_data():
        return [
            uuid.uuid4().__str__(),
            faker.name(),
            faker.random_element(elements=['Electronics', 'Clothing', 'Groceries']),
            faker.credit_card_number(), # Payment Method (using credit card)
            faker.random_int(min=1, max=100), # Quantity
            faker.random_int(min=10, max=1000), # Transaction Amount
            faker.random_element(elements=['Success', 'Failed', 'Pending']),
            faker.unix_time(), # Timestamp of transaction
        ]
```

```

for i in range(1,400):
    columns = ["transaction_id", "customer_name", "product_category", "payment_method", "quantity",
               "address", "city", "state", "zip_code", "lat", "long"]
    data_list = DataGenerator.get_data()
    json_data = dict(
        zip(columns, data_list)
    )
    _payload = json.dumps(json_data).encode("utf-8")
    response = producer.send(topic='kafka-hijir', value=_payload)
    print(json_data['transaction_id'], response.get())
    sleep(5) # Sleep for 5 seconds between producing records

```


Topic Name	Partitions	Out of sync replicas	Replicas
AVG_SALARY_TABLE_2	1	0	1
_consumer_offsets	50	0	1
_transaction_state	50	0	1
_confluent-ksql-default_command_topic	1	0	1
_confluent-ksql-default_query_CTAS_AVG_SALARY_TABLE_2...	1	0	1
_confluent-ksql-default_query_CTAS_AVG_SALARY_TABLE_2...	1	0	1
default_ksql_processing_log	1	0	1
kafka-hijir	1	0	1

Event sudah terkirim ke kafka (**kafka-hijir**)

Jupyter “hijir-kafka-consumer”

The screenshot shows a JupyterLab interface. On the left, a file tree displays several notebooks, with 'hijir-kafka-consumer.ipynb' selected. The main area contains the code for the consumer:

```
[*]: import json
[*]: import uuid
[*]: import os
[*]: import json
[*]: from dotenv import load_dotenv
[*]: from pathlib import Path
[*]: from kafka import KafkaProducer
[*]: from faker import Faker
[*]: from time import sleep
[*]: # Using the variables in .env file
[*]: dotenv_path = Path('/resources/.env')
[*]: load_dotenv(dotenv_path=dotenv_path)
[*]: # Testing one of the value inside .env
[*]: kafka_host = os.getenv('KAFKA_HOST')
[*]: from kafka import KafkaConsumer
[*]: # To consume latest messages and auto-commit offsets
[*]: consumer = KafkaConsumer('kafka-hijir',
[*]:                           group_id='group1',
[*]:                           bootstrap_servers=[f'{kafka_host}:9092'])
```

The screenshot shows the 'Consumers' page in the UI for Apache Kafka. The sidebar is set to 'local'. The main table lists consumer groups:

Group ID	Num Of Members	Num Of Topics	Messages Behind	Coordinator	
_confluent-ksql-default_query_CTAS_AVG_SALARY_TABLE_2_3	4	2	0	1	S
dibimbing-group	1	1	0	1	S
group_1	1	1	0	1	S

The screenshot shows the 'Consumers / group_1' page. The sidebar is set to 'local'. The main table provides details for the 'group_1' consumer:

State	Members	Assigned Topics	Assigned Partitions	Coordinator ID	Total lag
STABLE	1	1	1	1	0

Below the table, a search bar for 'Topic' shows 'kafka-hijir' and a 'Messages behind' count of 0.

The screenshot shows the UI for Apache Kafka interface. On the left, there's a sidebar with options like Dashboard, local brokers, Topics (which is selected), Consumers, KSQL DB, and ACL. The main area is titled 'Topics / kafka-hijir' and has tabs for Overview, Messages (which is selected), Consumers, Settings, and Statistics. Under the 'Messages' tab, there are sections for Seek Type (Offset), Partitions (All items are selected), Key Serde (String), Value Serde (String), and a search bar. Below these are buttons for 'Submit' and 'Oldest First'. A message list table is present, showing one message at offset 0, partition 0, timestamp 9/9/2024, 22:55:50. The message value is a JSON object: {"transaction_id": "abd68628-2ce0-4c32-baff-b7"}. There are tabs for Key, Value, and Headers. To the right of the message table, there are details: Timestamp 9/9/2024, 22:55:50 and Timestamp type: CREATE_TIME.

KsqlDB

Melakukan agregasi di kafka menggunakan sql

```
CREATE STREAM transaction_stream (
    transaction_id VARCHAR,
    customer_name VARCHAR,
    product_category VARCHAR,
    payment_method VARCHAR,
    quantity INT,
    transaction_amount INT,
    status VARCHAR,
    timestamp BIGINT
) WITH (
```

```
KAFKA_TOPIC = 'kafka-hijir',
  VALUE_FORMAT = 'JSON'
);
```

The screenshot shows the 'UI for Apache Kafka' interface. In the sidebar, 'local' is selected under 'KSQL DB'. The main area is titled 'KSQL DB' and contains a code editor with the following KSQL code:

```
CREATE STREAM transaction_stream (
  transaction_id VARCHAR,
  customer_name VARCHAR,
  product_category VARCHAR,
  payment_method VARCHAR,
  quantity INT,
  transaction_amount INT,
  status VARCHAR,
  timestamp BIGINT
) WITH (
  KAFKA_TOPIC = 'kafka-hijir',
  VALUE_FORMAT = 'JSON'
);
```

Below the code editor, there are tabs for 'Tables' and 'Streams', and a 'Stream properties' section with fields for 'Key' and 'Value' and a button to '+ Add Stream Property'. A large blue button at the top right says 'Execute KSQL Request'.

The screenshot shows the 'UI for Apache Kafka' interface after executing the KSQL code. The sidebar still shows 'local' under 'KSQL DB'. The main area now displays the executed KSQL code:

```
CREATE STREAM transaction_stream (
  transaction_id VARCHAR,
  customer_name VARCHAR,
  product_category VARCHAR,
  payment_method VARCHAR,
  quantity INT,
  transaction_amount INT,
  status VARCHAR,
  timestamp BIGINT
) WITH (
  KAFKA_TOPIC = 'kafka-hijir',
  VALUE_FORMAT = 'JSON'
);
```

At the bottom, there are buttons for 'Clear results' and 'Execute'. Below that, a 'Status' section shows 'status' and 'message' both set to 'SUCCESS' and 'Stream created' respectively.

Berhasil membuat stream

```
SELECT * FROM transaction_stream EMIT CHANGES;
```

localhost:8083/ui/clusters/local/ksqldb/query

UI for Apache Kafka fdd9ad9 5/11/2023, 21:02:15

Dashboard local * Brokers Topics Consumers KSQL DB ACL

Tables 1 Streams 3

Tables Streams

KSQL Clear Stream properties:

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

Key Value

'TRANSACTION_ID'	'CUSTOMER_NAME'	'PRODUCT_CATEGORY'	'PAYMENT_METHOD'	'QUANTITY'	'TRANSACTION_A'
642e0eb3-5aae-4e65-aab0-a74467aa399a	Daniel Castro	Electronics	3571090123750766	16	419
1d2ff838-2b69-4295-a6f5-e3653c8e4f9d	Joseph Holland	Books	6550666370346263	41	189
6749af49-f019-44db-8438-df2ceb85e0df	Donald Medina	Sports	2231038418797251	48	407

localhost:8083/ui/clusters/local/ksqldb/query

UI for Apache Kafka fdd9ad9 5/11/2023, 21:02:15

Dashboard local * Brokers Topics Consumers KSQL DB ACL

Clear results Execute

Schema

'TRANSACTION_ID'	'CUSTOMER_NAME'	'PRODUCT_CATEGORY'	'PAYMENT_METHOD'	'QUANTITY'	'TRANSACTION_A'
642e0eb3-5aae-4e65-aab0-a74467aa399a	Daniel Castro	Electronics	3571090123750766	16	419
1d2ff838-2b69-4295-a6f5-e3653c8e4f9d	Joseph Holland	Books	6550666370346263	41	189
6749af49-f019-44db-8438-df2ceb85e0df	Donald Medina	Sports	2231038418797251	48	407

Consuming query execution results Abort

'TRANSACTION_ID'	'CUSTOMER_NAME'	'PRODUCT_CATEGORY'	'PAYMENT_METHOD'	'QUANTITY'	'TRANSACTION_A'
642e0eb3-5aae-4e65-aab0-a74467aa399a	Daniel Castro	Electronics	3571090123750766	16	419
1d2ff838-2b69-4295-a6f5-e3653c8e4f9d	Joseph Holland	Books	6550666370346263	41	189
6749af49-f019-44db-8438-df2ceb85e0df	Donald Medina	Sports	2231038418797251	48	407

Agregasi

The screenshot shows the UI for Apache Kafka interface for KSQL DB. The left sidebar has a 'local' cluster selected. The main area displays the KSQL DB interface with a 'Tables' section showing 1 table and 3 streams. The KSQL query editor contains the following code:

```
1 CREATE TABLE product_category_agg AS
2   SELECT
3     product_category,
4     COUNT(*) AS transaction_count,
5     SUM(transaction_amount) AS
6       total_transaction_amount,
7     SUM(quantity) AS total_quantity
8   FROM transaction_stream
9   GROUP BY product_category
EMIT CHANGES;
```

The results section shows the schema and data for the aggregated table:

'PRODUCT_CATEGORY'	'TRANSACTION_COUNT'	'TOTAL_TRANSACTION_AMOUNT'	'TOTAL_QUANTITY'
Electronics	4	1711	189
Clothing	5	1364	298
Sports	4	2508	223
Groceries	8	2699	288
Books	7	3322	

At the bottom, a message indicates 'Consuming query execution result... Abort'.

Berikut ini adalah hasil agregasi dari tabel product_category