In [49]:
```
!pip install apache-flink

from pyflink.datastream.connectors.file_system import FileSource, StreamFormat
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.datastream.functions import MapFunction
from pyflink.common.watermark_strategy import WatermarkStrategy
from pyflink.datastream import RuntimeExecutionMode
import json

class JsonObjectMapFunction(MapFunction):
    def map(self, value):
        return json.loads(value)

def read_json_as_datastream(file_path: str, env: StreamExecutionEnvironment):
    source = FileSource.for_record_stream_format(StreamFormat.text_line_format(), file_path).build()
    data_stream = env.from_source(source, WatermarkStrategy.no_watermarks(), "txt_source")
    parsed_stream = data_stream.map(JsonObjectMapFunction())
    return parsed_stream
```

```
Requirement already satisfied: apache-flink in /usr/local/lib/python3.12/dist-packages (2.1.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (0.10.9.7)
Requirement already satisfied: python-dateutil<3,>=2.8.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.9.0.post0)
Requirement already satisfied: apache-beam<=2.61.0,>=2.54.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.61.0)
Requirement already satisfied: cloudpickle>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.2.1)
Requirement already satisfied: avro>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (1.12.1)
Requirement already satisfied: pytz>=2018.3 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2025.2)
Requirement already satisfied: fastavro!=1.8.0,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (1.12.1)
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.32.4)
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (5.29.5)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.0.2)
Requirement already satisfied: pandas<2.3,>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.2.2)
Requirement already satisfied: pyarrow<21.0.0,>=5.0.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (16.1.0)
Requirement already satisfied: pemja<0.5.6,>=0.5.5 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (0.5.5)
Requirement already satisfied: httplib2>=0.19.0 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (0.22.0)
Requirement already satisfied: ruamel.yaml>=0.18.4 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (0.18.16)
Requirement already satisfied: apache-flink-libraries<2.1.2,>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from apache-flink) (2.1.1)
Requirement already satisfied: crcmod<2.0,>=1.7 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (1.7)
Requirement already satisfied: orjson<4,>=3.9.7 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (3.11.4)
Requirement already satisfied: dill<0.3.2,>=0.3.1.1 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (0.3.1.1)
Requirement already satisfied: fasteners<1.0,>=0.3 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (0.20)
Requirement already satisfied: grpcio!=1.48.0,!=1.59.*,!=1.60.*,!=1.61.*,!=1.62.0,!=1.62.1,<1.66.0,<2,>=1.33.1 in /usr/local/lib/python3.12/dist-packages (from
apache-beam<=2.61.0,>=2.54.0->apache-flink) (1.65.5)
Requirement already satisfied: hdfs<3.0.0,>=2.1.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (2.7.3)
Requirement already satisfied: jsonschema<5.0.0,>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (4.25.1)
Requirement already satisfied: jsonpickle<4.0.0,>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (3.4.2)
Requirement already satisfied: objsize<0.8.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (0.7.1)
Requirement already satisfied: packaging>=22.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (25.0)
Requirement already satisfied: pymongo<5.0.0,>=3.8.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (4.15.4)
Requirement already satisfied: proto-plus<2,>=1.7.1 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (1.26.1)
Requirement already satisfied: pydot<2,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (1.4.2)
Requirement already satisfied: redis<6,>=5.0.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (5.3.1)
Requirement already satisfied: regex>=2020.6.8 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (2025.11.3)
Requirement already satisfied: sortedcontainers>=2.4.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (2.4.0)
Requirement already satisfied: typing-extensions>=3.7.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (4.15.0)
Requirement already satisfied: zstandard<1,>=0.18.0 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (0.25.0)
Requirement already satisfied: pyyaml<7.0.0,>=3.12 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (6.0.3)
Requirement already satisfied: pyarrow-hotfix<1 in /usr/local/lib/python3.12/dist-packages (from apache-beam<=2.61.0,>=2.54.0->apache-flink) (0.7)
Requirement already satisfied: pyparsing!=3.0.0,!=3.0.1,!=3.0.2,!=3.0.3,<4,>=2.4.2 in /usr/local/lib/python3.12/dist-packages (from httplib2>=0.19.0->apache-fli
nk) (3.2.5)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas<2.3,>=1.3.0->apache-flink) (2025.2)
Requirement already satisfied: find-libpython in /usr/local/lib/python3.12/dist-packages (from pemja<0.5.6,>=0.5.5->apache-flink) (0.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil<3,>=2.8.0->apache-flink) (1.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.26.0->apache-flink) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.26.0->apache-flink) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.26.0->apache-flink) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.26.0->apache-flink) (2025.11.12)
Requirement already satisfied: ruamel.yaml.clib>=0.2.7 in /usr/local/lib/python3.12/dist-packages (from ruamel.yaml>=0.18.4->apache-flink) (0.2.15)
Requirement already satisfied: docopt in /usr/local/lib/python3.12/dist-packages (from hdfs<3.0.0,>=2.1.0->apache-beam<=2.61.0,>=2.54.0->apache-flink) (0.6.2)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from jsonschema<5.0.0,>=4.0.0->apache-beam<=2.61.0,>=2.54.0->apache-fli
nk) (25.4.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12/dist-packages (from jsonschema<5.0.0,>=4.0.0->apache-beam<=2.6
1.0,>=2.54.0->apache-flink) (2025.9.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (from jsonschema<5.0.0,>=4.0.0->apache-beam<=2.61.0,>=2.54.0->apac
he-flink) (0.37.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.12/dist-packages (from jsonschema<5.0.0,>=4.0.0->apache-beam<=2.61.0,>=2.54.0->apache-fl
ink) (0.29.0)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /usr/local/lib/python3.12/dist-packages (from pymongo<5.0.0,>=3.8.0->apache-beam<=2.61.0,>=2.54.0->ap
ache-flink) (2.8.0)
```

```
Requirement already satisfied: PyJWT>=2.9.0 in /usr/local/lib/python3.12/dist-packages (from redis<6,>=5.0.0->apache-beam<=2.61.0,>=2.54.0->apache-flink) (2.10.
1)
```

In [50]:
```
# Path of the data files, change it accordingly
petowners_path = "/content/petowners.jsonl"
pets_path = "/content/pets.jsonl"
products_path = "/content/products.jsonl"
groomers_path = "/content/groomers.jsonl"
users_path = "/content/users.jsonl"
appointments_path = "/content/appointments.jsonl"
services_path = "/content/services.jsonl"
```

In [51]:
```
## example for PyFlink DataStream API
env = StreamExecutionEnvironment.get_execution_environment()

# Create the DataStream for users.jsonl
groomers_stream = read_json_as_datastream(groomers_path, env)

# Print each user object and its type
groomers_stream.map(lambda x: print(x, type(x)))

env.execute("sample_stream")
```

```
Exception in thread read_grpc_client_inputs:
Traceback (most recent call last):
  File "/usr/lib/python3.12/threading.py", line 1075, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.12/threading.py", line 1012, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/local/lib/python3.12/dist-packages/apache_beam/runners/worker/data_plane.py", line 721, in <lambda>
    target=lambda: self._read_inputs(elements_iterator),
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/apache_beam/runners/worker/data_plane.py", line 704, in _read_inputs
    for elements in elements_iterator:
                    ^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/grpc/_channel.py", line 543, in __next__
    return self._next()
           ^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/grpc/_channel.py", line 969, in _next
    raise self
grpc._channel._MultiThreadedRendezvous: <_MultiThreadedRendezvous of RPC that terminated with:
        status = StatusCode.CANCELLED
        details = "Multiplexer hanging up"
        debug_error_string = "UNKNOWN:Error received from peer ipv6:%5B::1%5D:43685 {created_time:"2025-11-30T00:19:54.970158362+00:00", grpc_status:1, grpc_mes
sage:"Multiplexer hanging up"}"
>
{'user_id': 'user_001', 'certification': 'AKC S.A.F.E. Groomer', 'rating': 4.51, 'offers': ['service_qGHboIDL']} <class 'dict'>
{'user_id': 'user_002', 'certification': 'National Certified Master Groomer', 'rating': 4.71, 'offers': ['service_001', 'service_002']} <class 'dict'>
{'user_id': 'user_003', 'certification': 'National Certified Master Groomer', 'rating': 3.5, 'offers': ['service_001', 'service_002', 'service_003']} <class 'di
ct'>
{'user_id': 'user_004', 'certification': 'Professional Dog Groomer Certification', 'rating': 4.39, 'offers': ['service_001', 'service_002']} <class 'dict'>
{'user_id': 'user_005', 'certification': 'Certified Professional Pet Stylist', 'rating': 4.61, 'offers': ['service_002']} <class 'dict'>
{'user_id': 'user_006', 'certification': 'AKC S.A.F.E. Groomer', 'rating': 4.58, 'offers': ['service_001']} <class 'dict'>
{'user_id': 'user_007', 'certification': 'AKC S.A.F.E. Groomer', 'rating': 4.67, 'offers': ['service_002', 'service_003']} <class 'dict'>
{'user_id': 'user_008', 'certification': 'AKC S.A.F.E. Groomer', 'rating': 4.73, 'offers': ['service_003']} <class 'dict'>
```

Out[51]:
```
<pyflink.common.job_execution_result.JobExecutionResult at 0x7ca4c1aefa40>
```

In [52]:
```
## example for PyFlink Table API
from pyflink.table import EnvironmentSettings, TableEnvironment
```

```
# Step 1: Initialize the TableEnvironment
env_settings = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env_settings)

table_env.execute_sql("""
    CREATE TABLE pets (
        pet_id STRING,
        name STRING,
        species STRING,
        breed STRING,
        dob STRING,
        owner_id STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'pets.jsonl',
        'format' = 'json'
    )
""")

result_table = table_env.sql_query("""
    SELECT
        *
    FROM pets
""")

with result_table.execute().collect() as results:
    for result in results:
        print(result)
```

```
<Row('pet_001', 'Cathy', 'Cat', 'Golden Retriever', '2015-12-15', 'user_008')>
<Row('pet_002', 'Stephen', 'Cat', 'Golden Retriever', '2020-06-07', 'user_015')>
<Row('pet_003', 'Luke', 'Cat', 'Siamese', '2021-04-07', 'user_018')>
<Row('pet_004', 'Rachel', 'Rabbit', 'Golden Retriever', '2020-10-14', 'user_015')>
<Row('pet_005', 'Andrew', 'Rabbit', 'Persian', '2017-12-16', 'user_007')>
<Row('pet_006', 'Katherine', 'Rabbit', 'Bulldog', '2020-02-23', 'user_011')>
<Row('pet_007', 'Stephen', 'Dog', 'Persian', '2016-11-01', 'user_004')>
<Row('pet_008', 'Adam', 'Dog', 'Bulldog', '2022-07-24', 'user_007')>
<Row('pet_009', 'Danny', 'Cat', 'Bulldog', '2020-11-21', 'user_016')>
<Row('pet_010', 'Brian', 'Dog', 'Corgi', '2020-01-21', 'user_011')>
<Row('pet_011', 'Donna', 'Rabbit', 'Corgi', '2020-03-10', 'user_002')>
<Row('pet_012', 'Matthew', 'Cat', 'Mixed', '2019-07-17', 'user_007')>
<Row('pet_013', 'Joseph', 'Dog', 'Siamese', '2019-01-20', 'user_019')>
<Row('pet_014', 'Shawn', 'Cat', 'Bulldog', '2021-06-16', 'user_020')>
<Row('pet_015', 'Linda', 'Rabbit', 'Golden Retriever', '2017-09-21', 'user_018')>
<Row('pet_016', 'Suzanne', 'Rabbit', 'Mixed', '2021-11-30', 'user_007')>
<Row('pet_017', 'Eric', 'Rabbit', 'Siamese', '2023-10-13', 'user_001')>
<Row('pet_018', 'Diana', 'Dog', 'Corgi', '2017-02-17', 'user_015')>
```

In [53]:
```
# Q1.a
env = StreamExecutionEnvironment.get_execution_environment()

# Your code here
users_stream = read_json_as_datastream(users_path, env)
users_stream.map(lambda x: print(x, type(x)))

env.execute("Q1.a")

# Each user record coming out of the datastream has type 'dict'
```

```
{'user_id': 'user_001', 'name': 'Kristen Mckee', 'email': 'diamondfields@yahoo.com', 'join_date': '2024-10-07', 'roles': {'is_groomer': True, 'is_pet_owner': Tr
ue}, 'address': {'street': '6681 Laura Ways Apt. 107', 'city': '', 'state': 'MA'}} <class 'dict'>
{'user_id': 'user_002', 'name': 'Mark Collins', 'email': 'christianjackson@gmail.com', 'join_date': '2025-04-06', 'roles': {'is_groomer': True, 'is_pet_owner':
True}, 'address': {'street': '0065 Anthony Knoll Suite 856', 'city': 'Duncanborough', 'state': ''}} <class 'dict'>
{'user_id': 'user_003', 'name': 'Jay Allen', 'email': 'villegasjimmy@lee.com', 'join_date': '2022-07-22', 'roles': {'is_groomer': True, 'is_pet_owner': True},
'address': {'street': '8256 Smith Ways', 'city': 'Hallville', 'state': 'GA'}} <class 'dict'>
{'user_id': 'user_004', 'name': 'Maria Castaneda', 'email': 'ehubbard@yahoo.com', 'join_date': '2022-03-26', 'roles': {'is_groomer': True, 'is_pet_owner': Tru
e}, 'address': {'street': '30164 Cabrera Drive Apt. 091', 'city': 'Dianaton', 'state': 'MT'}} <class 'dict'>
{'user_id': 'user_005', 'name': 'Sylvia Ho', 'email': 'kennedyjerry@hotmail.com', 'join_date': '2025-07-15', 'roles': {'is_groomer': True, 'is_pet_owner': Tru
e}, 'address': {'street': '003 William River Suite 037', 'city': '', 'state': 'MT'}} <class 'dict'>
{'user_id': 'user_006', 'name': 'Miguel White', 'email': 'thomas95@gmail.com', 'join_date': '2022-09-06', 'roles': {'is_groomer': True, 'is_pet_owner': True},
'address': {'street': '267 Paul Trace Apt. 408', 'city': 'Butlerport', 'state': 'MN'}} <class 'dict'>
{'user_id': 'user_007', 'name': 'Kathy Roberts', 'email': 'pstewart@yahoo.com', 'join_date': '2023-10-12', 'roles': {'is_groomer': True, 'is_pet_owner': True},
'address': {'street': '226 Strickland Walk', 'city': 'Maxwellborough', 'state': 'NM'}} <class 'dict'>
{'user_id': 'user_008', 'name': 'Hunter Anderson', 'email': 'tiffany35@armstrong-kirby.com', 'join_date': '2025-03-05', 'roles': {'is_groomer': True, 'is_pet_ow
ner': True}, 'address': {'street': '3356 Smith Junction', 'city': 'New Matthew', 'state': 'PA'}} <class 'dict'>
{'user_id': 'user_009', 'name': 'Jenna Gonzalez', 'email': 'stephenslori@stephens.com', 'join_date': '2024-08-07', 'roles': {'is_groomer': False, 'is_pet_owne
r': True}, 'address': {'street': '06368 Smith Prairie Suite 019', 'city': 'North Shelbyberg', 'state': 'NV'}} <class 'dict'>
{'user_id': 'user_010', 'name': 'Jesse Alvarado', 'email': 'allenclaudia@silva.com', 'join_date': '2023-05-03', 'roles': {'is_groomer': False, 'is_pet_owner': T
rue}, 'address': {'street': '534 Finley Port', 'city': 'Normantown', 'state': 'TN'}} <class 'dict'>
{'user_id': 'user_011', 'name': 'Cory Bass', 'email': 'umitchell@yahoo.com', 'join_date': '2023-04-04', 'roles': {'is_groomer': False, 'is_pet_owner': True}, 'a
ddress': {'street': '6884 Jonathan Extension', 'city': 'Anthonymouth', 'state': 'ME'}} <class 'dict'>
{'user_id': 'user_012', 'name': 'Logan Hughes', 'email': 'gentrywilliam@gmail.com', 'join_date': '2024-05-03', 'roles': {'is_groomer': False, 'is_pet_owner': Tr
ue}, 'address': {'street': '5809 Curtis Parkways', 'city': 'South Evanhaven', 'state': 'PA'}} <class 'dict'>
{'user_id': 'user_013', 'name': 'Nicholas Aguilar', 'email': 'richardbarnes@gordon-hayden.org', 'join_date': '2022-06-30', 'roles': {'is_groomer': False, 'is_pe
t_owner': True}, 'address': {'street': '43737 Ryan Village', 'city': 'Williamston', 'state': 'NH'}} <class 'dict'>
{'user_id': 'user_014', 'name': 'Leslie Hunter', 'email': 'rlopez@hotmail.com', 'join_date': '2023-04-03', 'roles': {'is_groomer': False, 'is_pet_owner': True},
'address': {'street': '3017 Lauren Unions', 'city': 'Farmerchester', 'state': 'NJ'}} <class 'dict'>
{'user_id': 'user_015', 'name': 'Virginia Marshall', 'email': 'csimpson@austin.com', 'join_date': '2022-03-24', 'roles': {'is_groomer': False, 'is_pet_owner': T
rue}, 'address': {'street': '51550 Oneill Ports Suite 975', 'city': 'East Robin', 'state': 'FL'}} <class 'dict'>
{'user_id': 'user_016', 'name': 'Keith Gardner', 'email': 'burgessjulie@yahoo.com', 'join_date': '2023-11-29', 'roles': {'is_groomer': False, 'is_pet_owner': Tr
ue}, 'address': {'street': '71707 Jeffrey Gardens Apt. 691', 'city': 'South Todd', 'state': 'CO'}} <class 'dict'>
{'user_id': 'user_017', 'name': 'Alexis Carey', 'email': 'amber35@gmail.com', 'join_date': '2023-11-25', 'roles': {'is_groomer': False, 'is_pet_owner': True},
'address': {'street': '23723 Rodriguez Junctions Apt. 098', 'city': 'Heidiville', 'state': 'SD'}} <class 'dict'>
{'user_id': 'user_018', 'name': 'Erin Johnson', 'email': 'nwilliams@hotmail.com', 'join_date': '2025-05-27', 'roles': {'is_groomer': False, 'is_pet_owner': Tru
e}, 'address': {'street': '3069 Sarah Roads', 'city': 'Port Annaton', 'state': 'DC'}} <class 'dict'>
{'user_id': 'user_019', 'name': 'Mary Anderson', 'email': 'wmason@levy.com', 'join_date': '2022-03-24', 'roles': {'is_groomer': False, 'is_pet_owner': True}, 'a
ddress': {'street': '758 Christina Port Apt. 136', 'city': 'Snyderborough', 'state': 'AZ'}} <class 'dict'>
{'user_id': 'user_020', 'name': 'Richard Paul', 'email': 'jakejohnson@gmail.com', 'join_date': '2022-09-23', 'roles': {'is_groomer': False, 'is_pet_owner': Tru
e}, 'address': {'street': '824 Cruz Burgs Apt. 710', 'city': 'Sanchezfort', 'state': 'WI'}} <class 'dict'>
```

```
Out[53]: <pyflink.common.job_execution_result.JobExecutionResult at 0x7ca4c28b6e40>
```

```
In [54]: # Q1.b

         # The JsonObjectMapFunction class is used to turn each line of text from a JSONL file
         # into a Python dictionary object. The Map function takes each line (type: JSON string)
         # and applies json.loads() on it to convert it into a Python dictionary. This way,
         # accessing fields using dictionary syntax instead of working with strings is possible.
```

```
In [55]: # Q1.c

         # In Flink's DataStream API, a subtask is one parallel copy of an operator. If an
         # operator's parallelism degree goes beyond 1, Flink creates multiple subtasks that
         # work simultaneously where each subtask handles a different partition of incoming
         # data. The "1>" shows which subtask generated that particular output line, which
         # demonstrates how Flink divides work across parallel instances to improve throughput
         # and handle larger loads of work.
```

In [56]:
```
# Q2.A DataStream API

env = StreamExecutionEnvironment.get_execution_environment()

pets_stream = read_json_as_datastream(pets_path, env)

filter_pets = pets_stream.filter(lambda pet: pet['species'] == 'Rabbit' and pet['breed'] == 'Golden Retriever')

filtered_pets = filter_pets.map(lambda pet: print({
     'pet_id': pet.get('pet_id'), 'name': pet.get('name'), 'dob': pet.get('dob')}))

env.execute("Q2.a DataStream API")
```

```
{'pet_id': 'pet_004', 'name': 'Rachel', 'dob': '2020-10-14'}
{'pet_id': 'pet_015', 'name': 'Linda', 'dob': '2017-09-21'}
```

Out[56]:
```
<pyflink.common.job_execution_result.JobExecutionResult at 0x7ca4c0dd5ee0>
```

In [57]:
```
# Q2.a Table API

# Your code here
env = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env)

table_env.execute_sql("""
    CREATE TABLE pets (
        pet_id STRING,
        name STRING,
        species STRING,
        breed STRING,
        dob STRING,
        owner_id STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'pets.jsonl',
        'format' = 'json'
    )
""")

result_table = table_env.sql_query("""
    SELECT pet_id, name, dob
    FROM pets
    WHERE species = 'Rabbit' AND breed = 'Golden Retriever'
""")

with result_table.execute().collect() as results:
    for result in results:
        print(result)
```

```
<Row('pet_004', 'Rachel', '2020-10-14')>
<Row('pet_015', 'Linda', '2017-09-21')>
```

In [58]:
```
# Q2.b DataStream API

env = StreamExecutionEnvironment.get_execution_environment()

# to output only final answer and not show all iterations
env.set_runtime_mode(RuntimeExecutionMode.BATCH)

# Your code here
appointments_stream = read_json_as_datastream(appointments_path, env)
```

```
pet_011 = (appointments_stream
        .filter(lambda val: val['pet_id'] == 'pet_011')
        .map(lambda val: 1)
        .key_by(lambda x: "all")
        .reduce(lambda prev, curr: prev + curr))

pet_011.map(lambda count: print(f"Total number of appointments for pet_011: {count}"))

env.execute("Q2.b DataStream API")
```

```
Total number of appointments for pet_011: 14
```

```
Out[58]: <pyflink.common.job_execution_result.JobExecutionResult at 0x7ca4c0da0f80>
```

In [59]:
```
# Q2.b Table API

# Your code here
env = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env)

table_env.execute_sql("""
    CREATE TABLE appointments (
        appointment_id STRING,
        pet_id STRING,
        service_id STRING,
        `date` STRING,
        `time` STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'appointments.jsonl',
        'format' = 'json'
    )
""")

result_table = table_env.sql_query("""
    SELECT COUNT(*) as appointment_count_for_pet_011
    FROM appointments
    WHERE pet_id = 'pet_011'
""")

with result_table.execute().collect() as results:
    for result in results:
        print(f"Total number of appointments for pet_011: {result[0]}")
```

```
Total number of appointments for pet_011: 14
```

In [60]:
```
# Q2.c Table API

# Your code here
env = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env)

table_env.execute_sql("""
    CREATE TABLE appointments (
        appointment_id STRING,
        pet_id STRING,
        service_id STRING,
        `date` STRING,
        `time` STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'appointments.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql("""
    CREATE TABLE pets (
        pet_id STRING,
        name STRING,
        species STRING,
        breed STRING,
        dob STRING,
        owner_id STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'pets.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql("""
    CREATE TABLE services (
        service_id STRING,
        `type` STRING,
        price INT,
        offered_by STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'services.jsonl',
        'format' = 'json'
    )
""")

result_table = table_env.sql_query("""
    SELECT p.owner_id, SUM(s.price) AS total_spent
    FROM pets p, appointments a, services s
    WHERE p.pet_id = a.pet_id
      AND a.service_id = s.service_id
      AND p.owner_id = 'user_001'
    GROUP BY p.owner_id
""")

with result_table.execute().collect() as results:
    for result in results:
        print(f"owner_id: {result[0]}, total_spent: ${result[1]:.2f}")
```

```
owner_id: user_001, total_spent: $123.00
```

In [61]:

```
# Q2.d

env = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env)

# Your code here

table_env.execute_sql(f"""
    CREATE TABLE appointments (
        appointment_id STRING,
        pet_id STRING,
        service_id STRING,
        `date` STRING,
        `time` STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'appointments.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql(f"""
    CREATE TABLE groomers (
        user_id STRING,
        certification STRING,
        rating DOUBLE,
        offers ARRAY<STRING>
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'groomers.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql(f"""
    CREATE TABLE services (
        service_id STRING,
        `type` STRING,
        price INT,
        offered_by STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'services.jsonl',
        'format' = 'json'
    )
""")

result_table = table_env.sql_query("""
    SELECT g.user_id AS groomer_id, COALESCE(
        (SELECT SUM(s.price)
         FROM services s, appointments a
         WHERE s.service_id = a.service_id
           AND s.offered_by = 'user_002'), 0) AS total_revenue
    FROM groomers g
    WHERE g.user_id = 'user_002'
""")

with result_table.execute().collect() as results:
    for result in results:
        print(f"groomer_id: {result[0]}, total_revenue: ${result[1]:.2f}")
```

```
groomer_id: user_002, total_revenue: $0.00
```

In [64]:
```
# Q2.e

env = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env)

# Your code here

table_env.execute_sql("""
    CREATE TABLE appointments (
        appointment_id STRING,
        pet_id STRING,
        service_id STRING,
        `date` STRING,
        `time` STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'appointments.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql(f"""
    CREATE TABLE groomers (
        user_id STRING,
        certification STRING,
        rating DOUBLE,
        offers ARRAY<STRING>
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'groomers.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql("""
    CREATE TABLE pets (
        pet_id STRING,
        name STRING,
        species STRING,
        breed STRING,
        dob STRING,
        owner_id STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'pets.jsonl',
        'format' = 'json'
    )
""")

table_env.execute_sql("""
    CREATE TABLE services (
        service_id STRING,
        `type` STRING,
        price INT,
        offered_by STRING
    ) WITH (
        'connector' = 'filesystem',
        'path' = 'services.jsonl',
        'format' = 'json'
    )
""")

result_table = table_env.sql_query("""
    SELECT g.user_id AS groomer_id, COUNT(DISTINCT p.owner_id) AS unique_pet_owners_served
    FROM appointments a, groomers g, services s, pets p
    WHERE a.service_id = s.service_id
      AND a.pet_id = p.pet_id
      AND s.offered_by = g.user_id
    GROUP BY g.user_id
    ORDER BY groomer_id
""")

with result_table.execute().collect() as results:
    for result in results:
        print(f"groomer_id: {result[0]}, unique_pet_owners_served: {result[1]}")

print(f"All other groomers have 0 unique_pet_owners_served")
```

```
groomer_id: user_001, unique_pet_owners_served: 7
groomer_id: user_005, unique_pet_owners_served: 9
groomer_id: user_006, unique_pet_owners_served: 8
groomer_id: user_008, unique_pet_owners_served: 7
All other groomers have 0 unique_pet_owners_served
```