# REPORT PROGRESS 2

**Main task:** Set up real-time data streaming from database

**Tool:** Redis, Spark

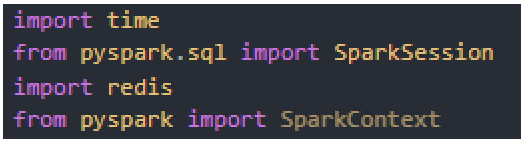
**Task description:**

- Upload data from Redis to Spark
- Set up real-time data streaming

**Task execution process**

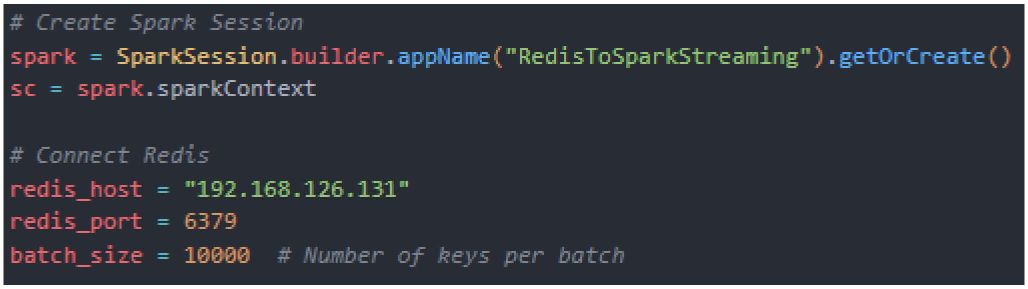
**Step 1: Import library**

Import the libraries needed for uploading data



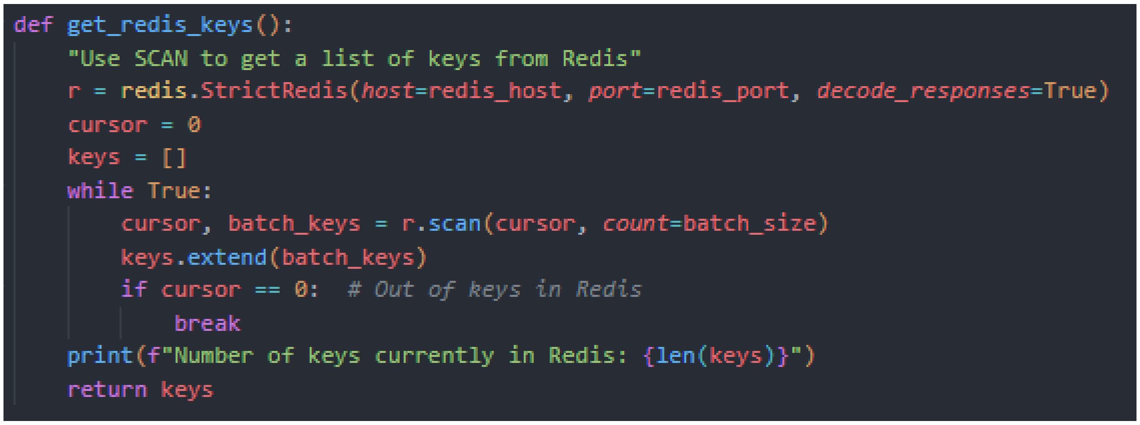
```
import time
from pyspark.sql import SparkSession
import redis
from pyspark import SparkContext
```

**Step 2: Create Spark Session and Connect Redis**



```
# Create Spark Session
spark = SparkSession.builder.appName("RedisToSparkStreaming").getOrCreate()
sc = spark.sparkContext

# Connect Redis
redis_host = "192.168.126.131"
redis_port = 6379
batch_size = 10000  # Number of keys per batch
```

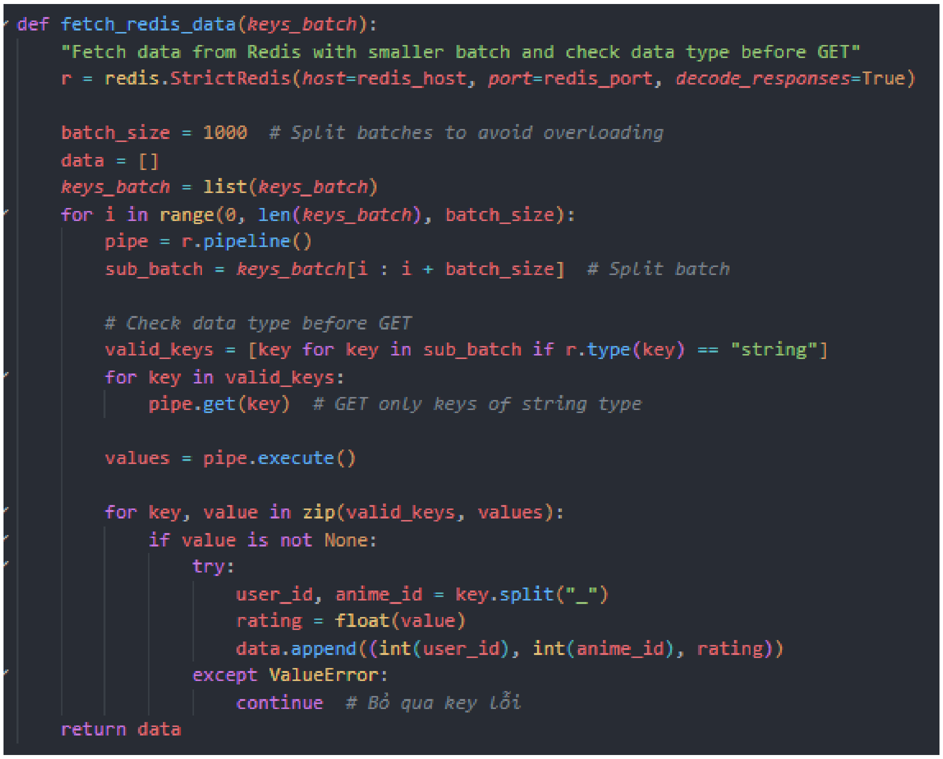
**Step 3: Get Redis keys**



```
def get_redis_keys():
    "Use SCAN to get a list of keys from Redis"
    r = redis.StrictRedis(host=redis_host, port=redis_port, decode_responses=True)
    cursor = 0
    keys = []
    while True:
        cursor, batch_keys = r.scan(cursor, count=batch_size)
        keys.extend(batch_keys)
        if cursor == 0:  # Out of keys in Redis
            break
    print(f"Number of keys currently in Redis: {len(keys)}")
    return keys
```

- Create a Redis connection using StrictRedis with information from redis_host and redis_port.
- decode_responses=True: Makes the data returned from Redis be strings instead of bytes.
- Initialize variables.
- Use scan() to get a batch of keys from Redis.
- cursor: Redis uses cursor to iterate through the data piece by piece.
- If cursor=0, it means start from the beginning.
- If scan() returns cursor=0 after one iteration, it means all keys have been scanned.
- Save the retrieved keys to the keys list.
- When cursor == 0, it means there is no more data to scan, exit the loop.
- Return the keys list.

**Step 4: Fetches data from Redis**

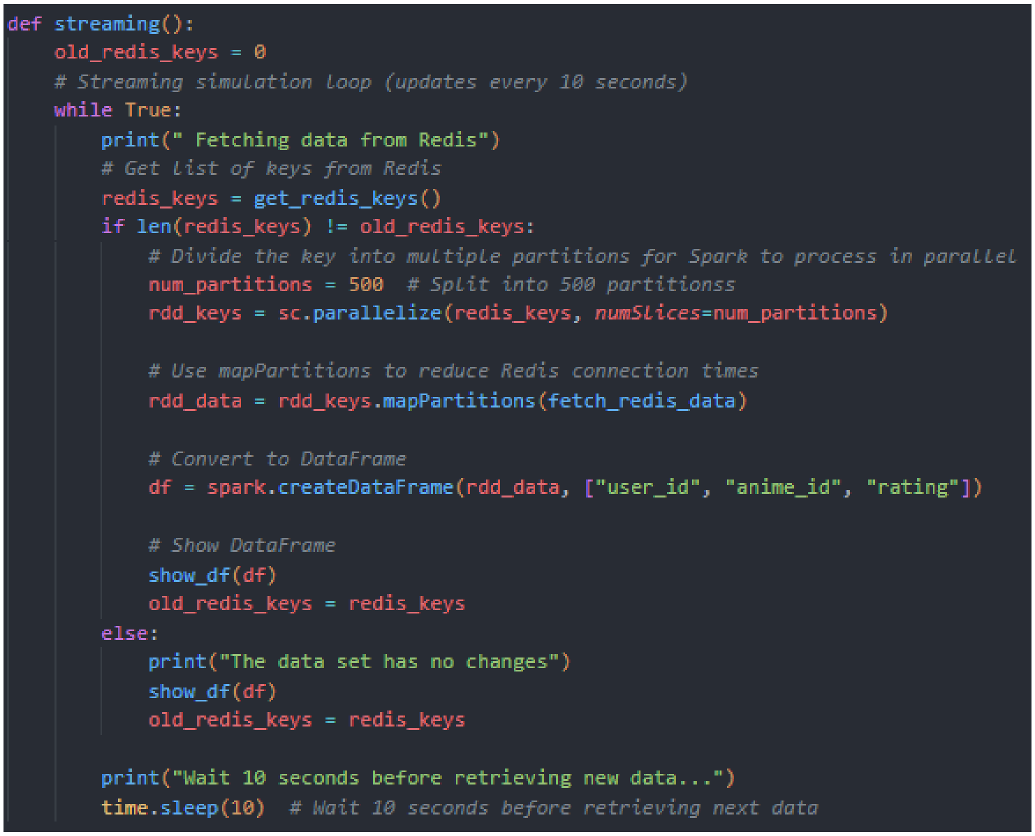


```python
def fetch_redis_data(keys_batch):
    "Fetch data from Redis with smaller batch and check data type before GET"
    r = redis.StrictRedis(host=redis_host, port=redis_port, decode_responses=True)

    batch_size = 1000  # Split batches to avoid overloading
    data = []
    keys_batch = list(keys_batch)
    for i in range(0, len(keys_batch), batch_size):
        pipe = r.pipeline()
        sub_batch = keys_batch[i : i + batch_size]  # Split batch

        # Check data type before GET
        valid_keys = [key for key in sub_batch if r.type(key) == "string"]
        for key in valid_keys:
            pipe.get(key)  # GET only keys of string type

        values = pipe.execute()

        for key, value in zip(valid_keys, values):
            if value is not None:
                try:
                    user_id, anime_id = key.split("_")
                    rating = float(value)
                    data.append((int(user_id), int(anime_id), rating))
                except ValueError:
                    continue  # Bỏ qua key lỗi
    return data
```

- This function fetches data from Redis while optimizing performance by using batch processing.

- A new Redis connection r is established using StrictRedis, with decode_responses=True to ensure values are returned as strings instead of bytes.
- batch_size = 1000: The function processes keys in chunks of 1000 to prevent overwhelming Redis.
- data = []: A list to store the final processed results.
- keys_batch = list(keys_batch): Ensures that keys_batch is a list, in case it was provided as another iterable (e.g., a generator).
- Looping through the keys in chunks of batch_size to fetch data in smaller portions.
- r.pipeline(): A Redis pipeline is used to reduce the number of network requests, improving efficiency.
- sub_batch = keys_batch[i : i + batch_size]: Extracts a smaller subset of keys for processing.
- Filters out keys that are not of type string before calling GET.
- This prevents unnecessary operations on incompatible key types (e.g., lists, sets, hashes).
- Adds GET commands for all valid keys to the Redis pipeline.
- Executes all GET commands in one batch to minimize network overhead.
- Iterates through the fetched values and their corresponding keys.
- Parses the key into user_id and anime_id (assuming they are separated by _).
- Converts the value to float since it represents a rating.
- Appends the parsed data as a tuple (user_id, anime_id, rating) to the data list.
- If an error occurs during conversion (e.g., improperly formatted keys or values), the function skips the key instead of crashing.
- Returns the final list of parsed (user_id, anime_id, rating) tuples.

**Step 5: Streaming data**



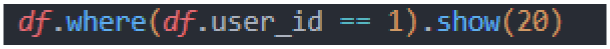
```python
def streaming():
    old_redis_keys = 0
    # Streaming simulation loop (updates every 10 seconds)
    while True:
        print(" Fetching data from Redis")
        # Get list of keys from Redis
        redis_keys = get_redis_keys()
        if len(redis_keys) != old_redis_keys:
            # Divide the key into multiple partitions for Spark to process in parallel
            num_partitions = 500  # Split into 500 partitionss
            rdd_keys = sc.parallelize(redis_keys, numSlices=num_partitions)

            # Use mapPartitions to reduce Redis connection times
            rdd_data = rdd_keys.mapPartitions(fetch_redis_data)

            # Convert to DataFrame
            df = spark.createDataFrame(rdd_data, ["user_id", "anime_id", "rating"])

            # Show DataFrame
            show_df(df)
            old_redis_keys = redis_keys
        else:
            print("The data set has no changes")
            show_df(df)
            old_redis_keys = redis_keys

        print("Wait 10 seconds before retrieving new data...")
        time.sleep(10)  # Wait 10 seconds before retrieving next data
```

- The streaming() function is designed to continuously fetch data from Redis, process it using Apache Spark, and display the results every 10 seconds. This simulates a real-time data ingestion pipeline.
- old_redis_keys = 0: This variable keeps track of the number of previously fetched Redis keys. It is used to detect if new data has been added to Redis, avoiding unnecessary processing.
- Runs an infinite loop that fetches data every 10 seconds. Acts as a real-time data ingestion pipeline where Redis is polled for updates.
- Calls get_redis_keys(), which retrieves all keys stored in Redis. The number of keys will determine whether data processing is needed.
- Compares the new key count with old_redis_keys, if the number of keys has changed, new data is available for

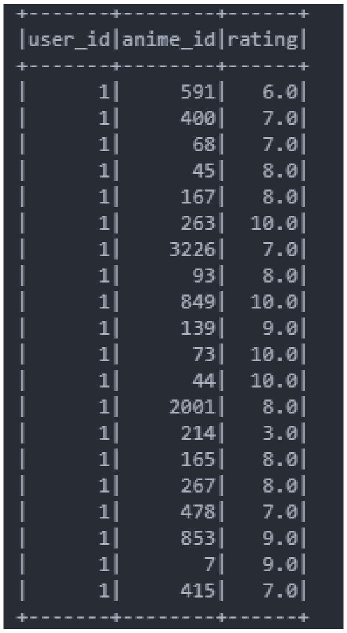
processing. This avoids unnecessary computation when the data remains the same.

- Redis keys are divided into 500 partitions for parallel processing using Spark.
- sc.parallelize(redis_keys, numSlices=500) creates an RDD (Resilient Distributed Dataset) to distribute data across multiple executors. This helps scale the processing across multiple nodes.
- mapPartitions(fetch_redis_data) applies fetch_redis_data() to each partition of keys.
- Instead of opening a new Redis connection for each key, it processes keys in batches, reducing the number of Redis calls. Optimized for performance when dealing with large datasets.
- The processed RDD is converted into a Spark DataFrame. The DataFrame has three columns: user_id, anime_id, and rating. This DataFrame is now ready for further analytics or machine learning models.
- Updates old_redis_keys to store the new state of Redis keys. This ensures the function only processes newly added data in the next iteration.
- If no new data is found, it skips reprocessing and just displays the existing DataFrame.
- This prevents unnecessary computation, making the function efficient.

**Result:**

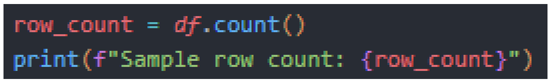


```
df.where(df.user_id == 1).show(20)
```

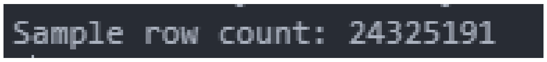


```
+-------+--------+------+
|user_id|anime_id|rating|
+-------+--------+------+
|      1|     591|   6.0|
|      1|     400|   7.0|
|      1|      68|   7.0|
|      1|      45|   8.0|
|      1|     167|   8.0|
|      1|     263|  10.0|
|      1|    3226|   7.0|
|      1|      93|   8.0|
|      1|     849|  10.0|
|      1|     139|   9.0|
|      1|      73|  10.0|
|      1|      44|  10.0|
|      1|    2001|   8.0|
|      1|     214|   3.0|
|      1|     165|   8.0|
|      1|     267|   8.0|
|      1|     478|   7.0|
|      1|     853|   9.0|
|      1|       7|   9.0|
|      1|     415|   7.0|
+-------+--------+------+
```

Show first 20 rows are reviews of user with id 1 for anime in dataframe



```
row_count = df.count()
print(f"Sample row count: {row_count}")
```



```
Sample row count: 24325191
```

Count the number of rows in the current Dataframe