

P-3: Implementation

Brazilian E-commerce Dataset

Team 4:

Anushka Darade: NUID: 002734159

Yash Panchal NUID: 002771456

Dhriti Kanchan: NUID: 002794620

Vikash Singh: NUID: 002929181

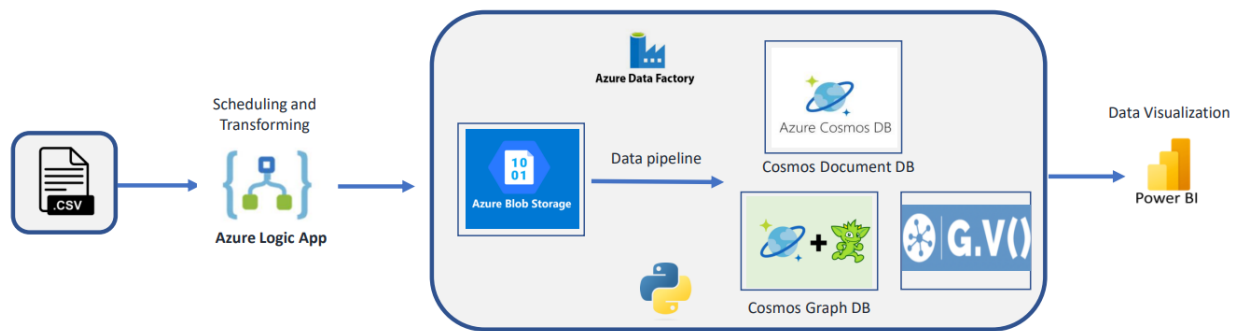
Swapnil Bhasgauri NUID: 002752978

Implementation Description:

An overview of the implementation procedure is as follows:

1. Upon conducting Data Profiling for the Brazilian E-commerce dataset, we observed and addressed issues such as missing values, inconsistent data, and ensured alignment of data types with respect to the data dictionary. Initially, we created a data dictionary specific to the Brazilian E-commerce dataset, outlining column names and corresponding data types.
2. The files from the blob storage are imported into Azure Data Factory in the Dataset section for the Azure Cosmos Document DB implementation. The files are cleansed, integrated, transformed, aggregated as per required hierarchy and nesting and finally, sink it into the destination Azure Cosmos Document DB.
3. The first step in implementing Azure Graph DB is to utilize Azure Data Factory to merge the various.csv files needed for the graph implementation into a single file. We created a Python connection to our Azure Blob Storage, which is where the combined file is stored, so that it can be retrieved immediately. Next, using the Gremlin API, created a Python connection to the Azure Cosmos Graph Database. After the connection was made, we used the Gremlin API to generate the vertices and edges and ingested data into Azure Cosmos Graph DB.

Data Architecture Diagram:



Data Catalog:

olist_customers_dataset

| | | |
|--------------------------|----------|--|
| customer_id | v_string | |
| customer_unique_id | v_string | |
| customer_zip_code_prefix | v_string | |
| customer_city | v_string | |
| customer_state | v_string | |

olist_geolocation_dataset

| | | |
|-----------------------------|----------|--|
| geolocation_zip_code_prefix | v_string | |
| geolocation_lat | v_string | |
| geolocation_lng | v_string | |
| geolocation_city | v_string | |
| geolocation_state | v_string | |

olist_order_items_dataset

| | | |
|---------------------|----------|--|
| order_id | v_string | |
| order_item_id | v_string | |
| product_id | v_string | |
| seller_id | v_string | |
| shipping_limit_date | v_string | |
| price | v_string | |
| freight_value | v_string | |

olist_order_payments_dataset

| | | |
|----------------------|----------|--|
| order_id | v_string | |
| payment_sequential | v_string | |
| payment_type | v_string | |
| payment_installments | v_string | |
| payment_value | v_string | |

olist_order_reviews_dataset

| | | |
|-------------------------|----------|--|
| review_id | v_string | |
| order_id | v_string | |
| review_score | v_string | |
| review_comment_title | v_string | Null (87656) |
| review_comment_message | v_string | Empty and null (58247) Cleaned Null, blank, white spaces, ///////////////, *****, |
| review_creation_date | v_string | Remove the timestamp part from the data |
| review_answer_timestamp | v_string | Remove the timestamp part from the data |

olist_orders_dataset

| | | |
|-------------------------------|----------|---|
| order_id | v_string | |
| customer_id | v_string | |
| order_status | v_string | |
| order_purchase_timestamp | v_string | |
| order_approved_at | v_string | Null (160) |
| order_delivered_carrier_date | v_string | Null (1783) |
| order_delivered_customer_date | v_string | Null (2965), Remove Timestamp from the data and replace with 1900-01-01 |
| order_estimated_delivery_date | v_string | Remove Timestamp from the data and replace with 1900-01-01 |

olist_products_dataset

| | | |
|----------------------------|----------|---|
| product_id | v_string | |
| product_category_name | v_string | Null (610), Replacing Null with "No Value Provided" |
| product_name_lenght | v_string | Null (610), Replacing Null with "No Value Provided" |
| product_description_lenght | v_string | Null (610), Replacing Null with "No Value Provided" |
| product_photos_qty | v_string | Null (610), Replacing Null with "-1" |
| product_weight_g | v_string | Null (2), Replacing Null with "-1" |
| product_length_cm | v_string | Null (2), Replacing Null with "-1" |
| product_height_cm | v_string | Null (2), Replacing Null with "-1" |
| product_width_cm | v_string | Null (2), Replacing Null with "-1" |

olist_sellers_dataset

| | | |
|------------------------|----------|--|
| seller_id | v_string | |
| seller_zip_code_prefix | v_string | |
| seller_city | v_string | |
| seller_state | v_string | |

product_category_name_translation

| | | |
|-------------------------------|----------|--|
| product_category_name | v_string | |
| product_category_name_english | v_string | |

Implementation:

Files in Azure Blob Storage:

Microsoft Azure

Search resources, services, and docs (G+I)

Home > test96 | Containers >

test4296

Container

Search

Upload

Change access level

Refresh

Delete

Change tier

Acquire lease

Break lease

View snapshots

Create snapshot

Give feedback

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

Shared access tokens

Access policy

Properties

Metadata

Authentication method: Access key (Switch to Microsoft Entra user account)

Location: test4296

Search blobs by prefix (case-sensitive)

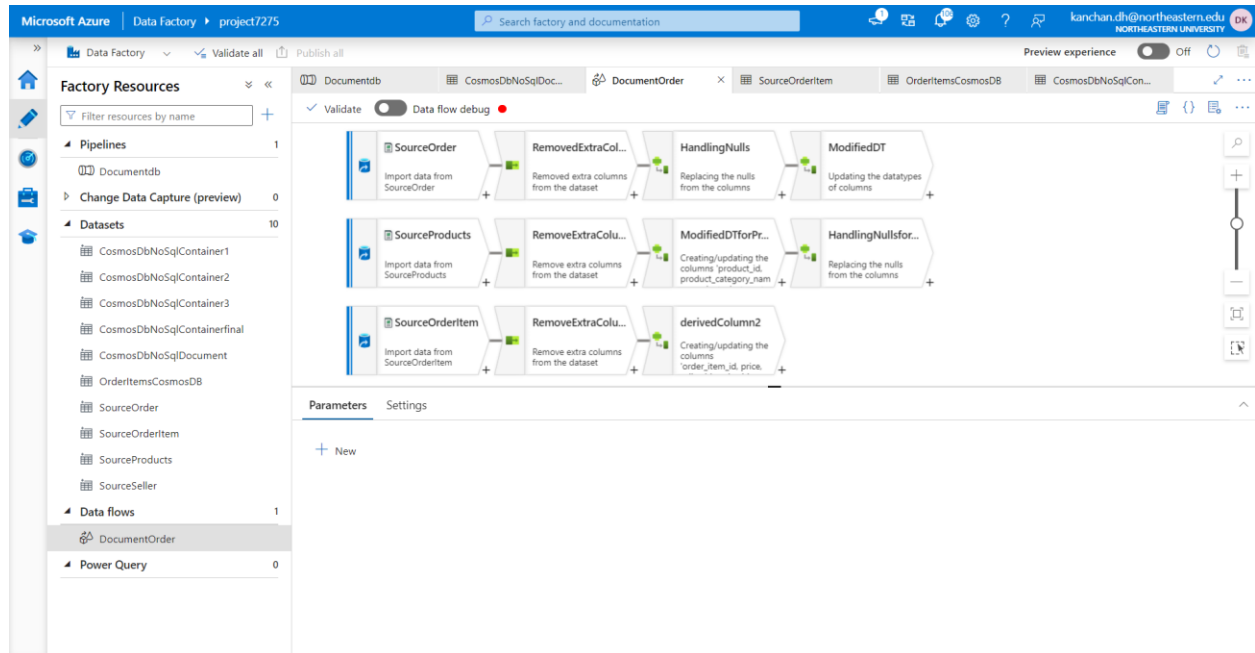
Show deleted blobs

Add filter

| Name | Modified | Access tier | Archive status | Blob type | Size | Lease state | |
|--|-------------------------|----------------|----------------|------------|------------|-------------|-----|
| <input type="checkbox"/> olist_customers_dataset.csv | 11/15/2023, 4:06:09 ... | Hot (Inferred) | | Block blob | 8.62 MiB | Available | ... |
| <input type="checkbox"/> olist_geolocation_dataset.csv | 11/15/2023, 5:24:39 ... | Hot (Inferred) | | Block blob | 58.44 MiB | Available | ... |
| <input type="checkbox"/> olist_order_items_dataset.csv | 11/15/2023, 4:13:06 ... | Hot (Inferred) | | Block blob | 14.72 MiB | Available | ... |
| <input type="checkbox"/> olist_order_payments_dataset.csv | 11/15/2023, 4:15:30 ... | Hot (Inferred) | | Block blob | 5.51 MiB | Available | ... |
| <input type="checkbox"/> olist_order_reviews_dataset.csv | 11/15/2023, 4:17:56 ... | Hot (Inferred) | | Block blob | 13.78 MiB | Available | ... |
| <input type="checkbox"/> olist_orders_dataset.csv | 11/15/2023, 4:19:21 ... | Hot (Inferred) | | Block blob | 16.84 MiB | Available | ... |
| <input type="checkbox"/> olist_products_dataset.csv | 11/15/2023, 4:21:13 ... | Hot (Inferred) | | Block blob | 2.27 MiB | Available | ... |
| <input type="checkbox"/> olist_sellers_dataset.csv | 11/15/2023, 4:08:06 ... | Hot (Inferred) | | Block blob | 170.61 KiB | Available | ... |
| <input type="checkbox"/> product_category_name_translation.csv | 11/15/2023, 4:09:56 ... | Hot (Inferred) | | Block blob | 2.55 KiB | Available | ... |

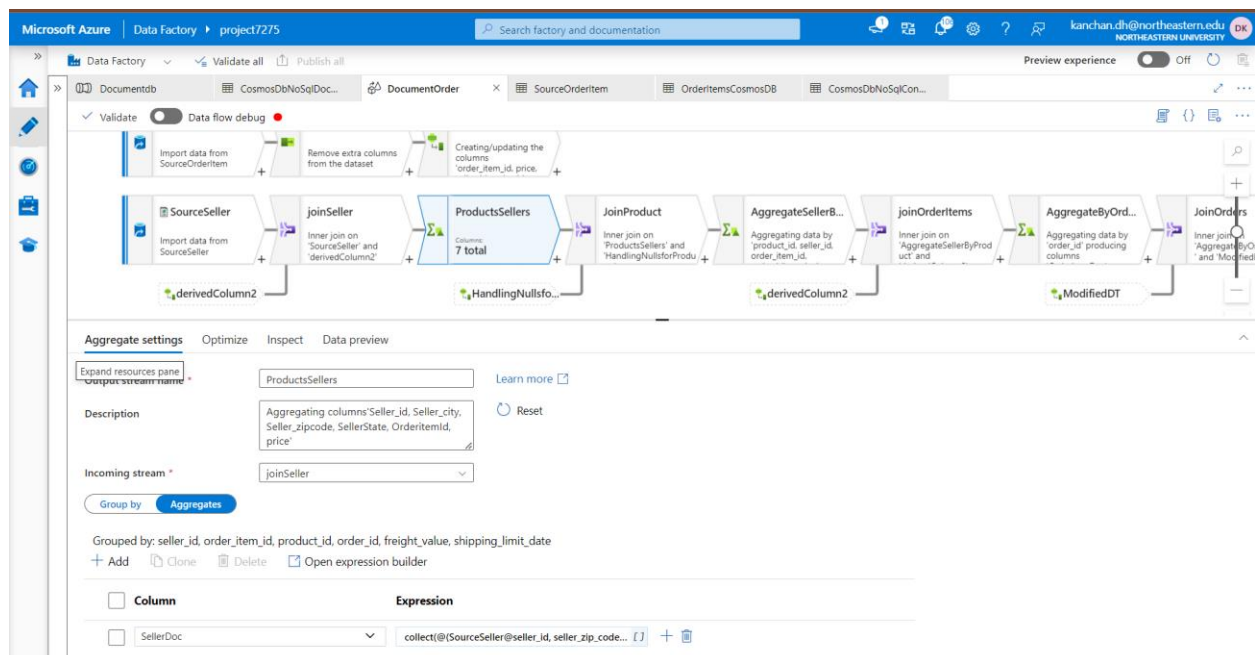
AZURE COSMOS DOCUMENT DB implementation:

In Azure Data Factory, we created dataflow to carry out various data transformation and cleaning functions on.csv files, such as: deleting unnecessary columns, managing null values in columns, and changing datetime, timestamp, and other data types to the appropriate format and data type.



Applying joins to combine data from multiple. csvs.

After join at each step Aggregating by id, to achieve the format required for Document DB:
Below, aggregated by seller_id:



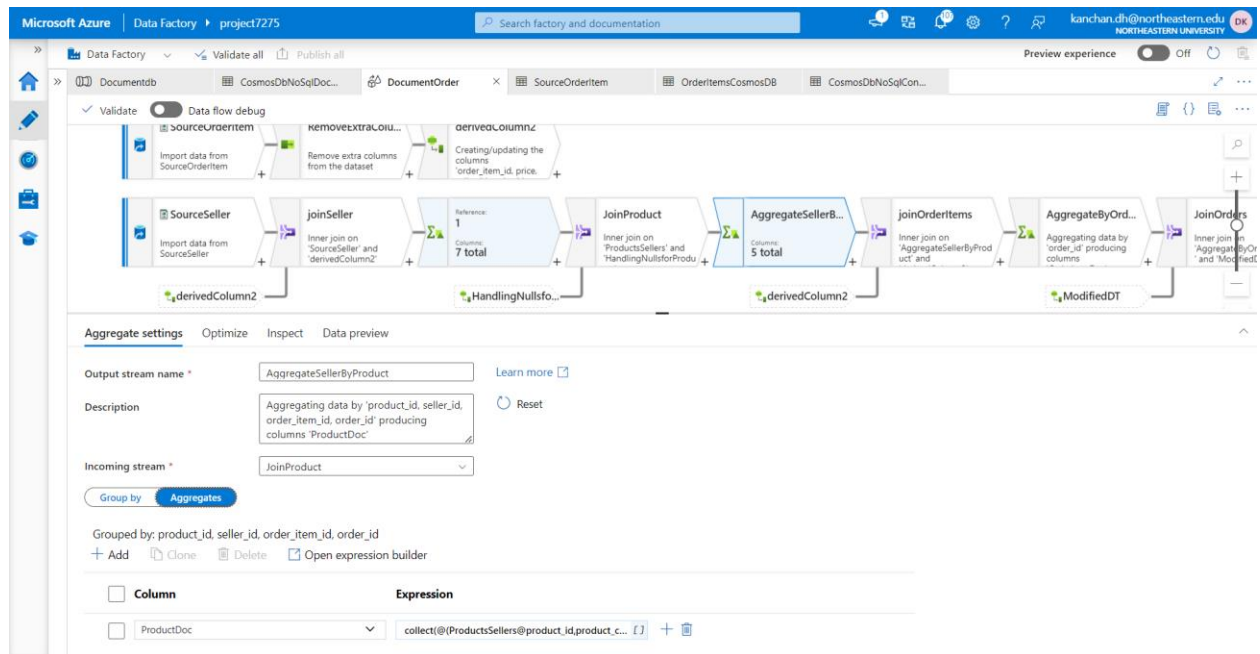
Expression used for Aggregation: `collect(@(id, column_1, column_2, ..., column_N)`

The screenshot shows the 'Dataflow expression builder' in Microsoft Azure Data Factory. The 'Column name' field is set to 'SellerDoc'. The 'Expression' field contains the code: `collect(@(SourceSeller@seller_id, seller_zip_code_prefix, seller_city, seller_state))`. Below the expression field, there are two panels: 'Expression elements' and 'Expression values'. The 'Expression values' panel shows a list of variables: `SourceSeller@seller_id`, `seller_zip_code_prefix`, `seller_city`, `seller_state`, `order_item_id`, and `price`. The interface includes a 'Save and finish' button and a 'Clear contents' link.

Data Preview:

The screenshot shows the 'Data Preview' view of a Dataflow in Microsoft Azure Data Factory. The interface displays a pipeline with several steps: 'Import data from SourceSeller', 'JoinSeller', 'ProductsSellers', 'JoinProduct', 'AggregateSellerB...', 'JoinOrderItems', 'AggregateByOrd...', and 'JoinOrders'. The 'Data preview' tab is active, showing a table with columns: `seller_id`, `order_item_id`, `product_id`, `order_id`, `freight_value`, `shipping_limit_date`, and `SellerDoc`. The table contains 100 rows of data. A dropdown menu is open for the `SellerDoc` column, showing the values: `01c7e3d21494c41...`, `01c7e3d21494c41...`, `0b90b6df587eb83...`, `0b90b6df587eb83...`, `0b90b6df587eb83...`, `0b90b6df587eb83...`, `0b90b6df587eb83...`, `0b90b6df587eb83...`, `0b90b6df587eb83...`, and `0b90b6df587eb83...`. The interface also includes a 'Refresh' button and a 'Map drifted' link.

Aggregated by respective ids from the product table to achieve the nested document db json format:



The screenshot shows the 'Dataflow expression builder' interface for the 'AggregateSellerByProduct' step. The 'Aggregate Columns' section shows a list of columns: 'ProductDoc'. The 'Column name' field is set to 'ProductDoc'. The 'Expression' field contains the following code:

```
collect(@(ProductsSellers@product_id,product_category_name,product_height_cm,product_weight_g,product_photos_qty,product_width_cm,p
```

. The 'Expression elements' section lists various functions and parameters, including 'seller_id', 'order_item_id', 'ProductsSellers@product_id', 'order_id', 'freight_value', and 'shipping_limit_date'. The 'Expression values' section provides a search bar and a list of values to select from.

Data Preview:

Number of rows: INSERT 17, UPDATE 0, DELETE 0, UPSERT 0, LOOKUP 0, ERROR 0, TOTAL 17

| product_id | seller_id | order_item_id | order_id | ProductDoc |
|----------------------|-----------------------|---------------|----------------------|------------|
| e10758160da97891c... | 323ce52b5b81df2cd... | 1 | 006cb7cfc99b2954... | [...] |
| f9471562478eba876... | 3785b653b1b82de8... | 1 | 014872cb0c4fc7695... | [...] |
| f9471562478eba876... | 3785b653b1b82de8... | 2 | 014872cb0c4fc7695... | [...] |
| f9471562478eba876... | 3785b653b1b82de8... | 3 | 014872cb0c4fc7695... | [...] |
| f9471562478eba876... | 3785b653b1b82de8... | 4 | 014872cb0c4fc7695... | [...] |
| bbf920aa6ac72007a... | 391fc6631aebcf3004... | 1 | 01dcb39823b4cda... | [...] |
| bbf920aa6ac72007a... | 391fc6631aebcf3004... | 2 | 01dcb39823b4cda... | [...] |
| 5e21d5cab5d33e77... | 6560211a19b47992c... | 1 | 00418a49a685c6bb... | [...] |

ProductDoc details (from expanded view):

- product_id: e10758160da97891c2fdbcb35f0f031d
- product_category_name: No Value Provided
- product_height_cm: 2
- product_weight_g: 2280
- product_photos_qty: -1

Aggregated by respective ids from the Order Item table to achieve the nested document db json format and creating a column named 'OrderItemDoc':

Output stream name: AggregateByOrderItem

Description: Aggregating data by 'order_id' producing columns 'OrderItemDoc'

Incoming stream: joinOrderItems

Group by: Aggregate

Grouped by: order_id

| Column | Expression |
|--------------|---|
| OrderItemDoc | collect(@(AggregateSellerByProduct@order_ite... |

Microsoft Azure | Data Factory | project7275 | Search factory and documentation | kanchan.dh@northeastern.edu

Dataflow expression builder

Aggregate Columns

+ Create new

// OrderItemDoc []

Column name *

OrderItemDoc

Expression

collect(@(AggregateSellerByProduct@order_item_id,freight_value,shipping_limit_date,price, ProductDoc))

Expression elements

- All
- Functions
- Input schema
- Parameters
- Cached lookup
- Data flow library functions
- Locals

Expression values

Filter by keyword

+ Create new

- AggregateSellerByProduct@product_id
- AggregateSellerByProduct@seller_id
- AggregateSellerByProduct@order_item_id
- AggregateSellerByProduct@order_id
- ProductDoc []
- derivedColumn2@order_item_id

Save and finish Cancel Clear contents

Data Preview:

Microsoft Azure | Data Factory | project7275 | Search factory and documentation | kanchan.dh@northeastern.edu

Data Factory | Validate all | Publish all | Preview experience Off

Documentdb CosmosDbNoSqlDoc... DocumentOrder SourceOrderItem OrderItemsCosmosDB CosmosDbNoSqlCon...

Validate Data flow debug Debug Settings

Columns set

ProductsSellers

JoinProduct

AggregateSeller8...

JoinOrderItems

AggregateByOrd...

JoinOrders

LoadingCosmo...

Export data to Ordercontainer

Aggregate settings Optimize Inspect Data preview

Number of rows INSERT 13 UPDATE 0 DELETE 0 UPSERT 0 LOOKUP 0 ERROR 0 TOTAL 13

Refresh Typecast Modify Map drifted Statistics Remove Export to CSV

| order_id | OrderItemDoc |
|----------------------|------------------------------------|
| 006cb7c9c99b2954... | 123 order_item_id: 1 |
| 014872cb0c4fc7695... | abc freight_value: 14.14 |
| 01dccb39823b4cda... | shipping_limit_date: 1519257600000 |
| 00418a49a685c6bb... | price: 56.00 |
| 0166bc333d66b414b... | ProductDoc [] |
| 01f66e58769f84129... | |
| 00404fa7a687c8c44... | |
| 011c899816ea29773... | |

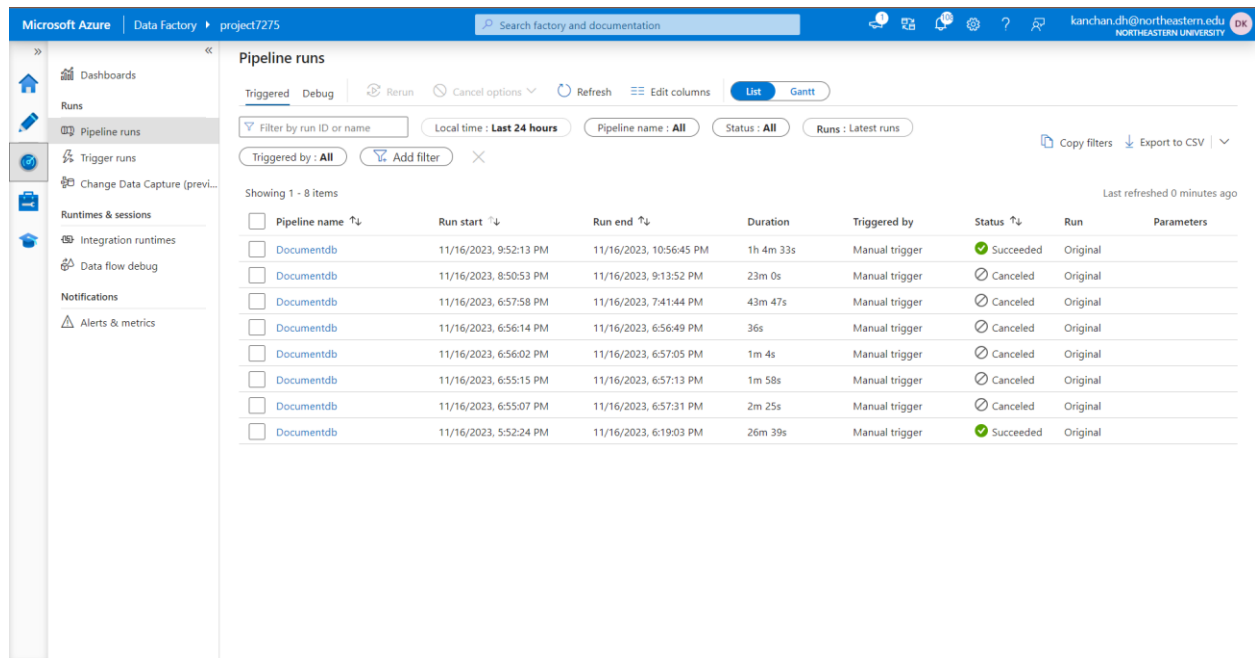
Last step to sink the data into Azure Cosmos Document DB:

The screenshot displays the Microsoft Azure Data Factory interface for a project named 'project7275'. The top navigation bar includes the 'Data Factory' tab, a search bar, and user information. The main area shows a data pipeline graph with several activities: 'Remove extra columns from the dataset', 'Creating/updating the columns', 'JoinSeller', 'ProductsSellers', 'JoinProduct', 'AggregateSellerB...', 'JoinOrderItems', 'AggregateByOrd...', 'JoinOrders', and 'LoadingCosmosdocdb'. The 'LoadingCosmosdocdb' sink is highlighted, and its properties are shown in the bottom pane. The 'Output stream name' is 'LoadingCosmosdocdb', the 'Description' is 'Export data to Ordercontainer', and the 'Incoming stream' is 'JoinOrders'. The 'Sink type' is set to 'Dataset', and the 'Dataset' is 'CosmosDbNoSqlContainerfinal'. The 'Options' section includes 'Allow schema drift' (checked) and 'Validate schema' (unchecked).

Running the data pipeline:

The screenshot shows the Microsoft Azure Data Factory interface with the 'Data flow' task selected. The 'Data flow' task is highlighted in the main area, and its properties are shown in the bottom pane. The 'Data flow' is 'DocumentOrderfinal', and the 'Run on (Azure IR)' is 'AutoResolveIntegrationRuntime'. The 'Compute size' is set to 'Small'. The 'Logging level' is set to 'Verbose'. The 'Sink properties' and 'Staging' sections are also visible.

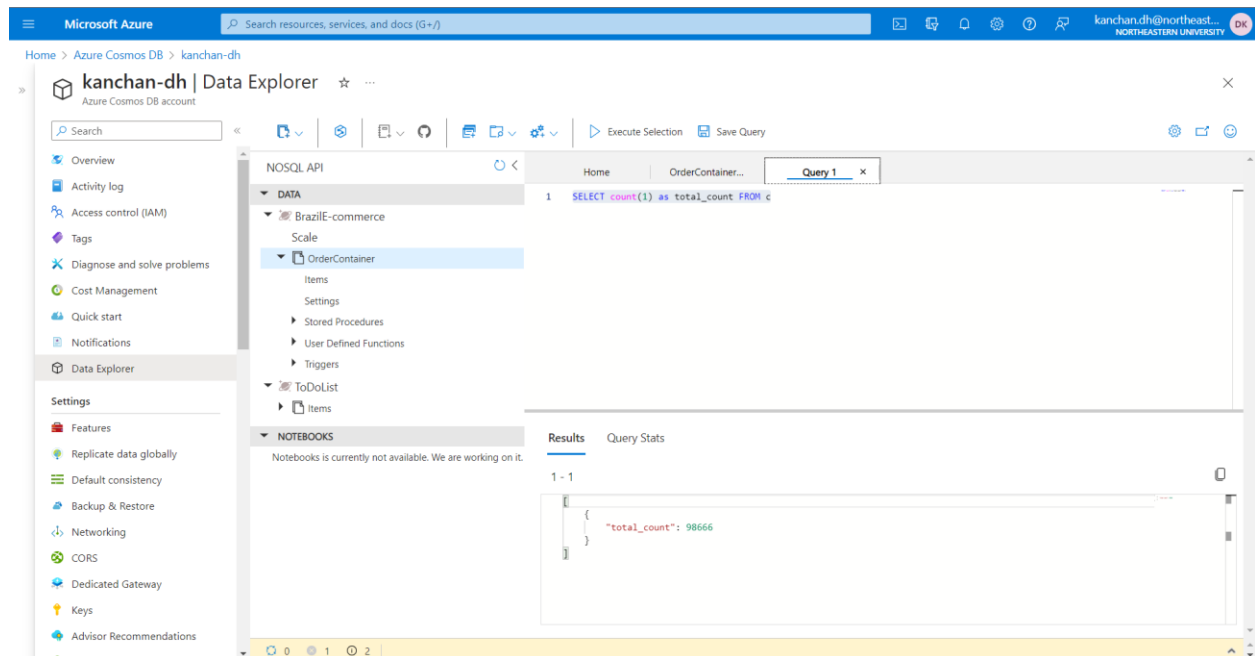
Successful execution of the data pipeline:



The screenshot shows the 'Pipeline runs' page in Microsoft Azure Data Factory. The left sidebar contains navigation options: Dashboards, Runs, Pipeline runs (selected), Trigger runs, Change Data Capture (previous), Runtimes & sessions, Integration runtimes, Data flow debug, Notifications, and Alerts & metrics. The main area displays a table of pipeline runs for 'Documentdb'. The table has columns: Pipeline name, Run start, Run end, Duration, Triggered by, Status, Run, and Parameters. The runs are filtered by 'Local time: Last 24 hours', 'Pipeline name: All', and 'Status: All'. The table shows 8 items, with the first 7 being 'Canceled' and the last one 'Succeeded'.

| Pipeline name | Run start | Run end | Duration | Triggered by | Status | Run | Parameters |
|---------------|------------------------|-------------------------|-----------|----------------|-----------|----------|------------|
| Documentdb | 11/16/2023, 9:52:13 PM | 11/16/2023, 10:56:45 PM | 1h 4m 33s | Manual trigger | Succeeded | Original | |
| Documentdb | 11/16/2023, 8:50:53 PM | 11/16/2023, 9:13:52 PM | 23m 0s | Manual trigger | Canceled | Original | |
| Documentdb | 11/16/2023, 6:57:58 PM | 11/16/2023, 7:41:44 PM | 43m 47s | Manual trigger | Canceled | Original | |
| Documentdb | 11/16/2023, 6:56:14 PM | 11/16/2023, 6:56:49 PM | 36s | Manual trigger | Canceled | Original | |
| Documentdb | 11/16/2023, 6:56:02 PM | 11/16/2023, 6:57:05 PM | 1m 4s | Manual trigger | Canceled | Original | |
| Documentdb | 11/16/2023, 6:55:15 PM | 11/16/2023, 6:57:13 PM | 1m 58s | Manual trigger | Canceled | Original | |
| Documentdb | 11/16/2023, 6:55:07 PM | 11/16/2023, 6:57:31 PM | 2m 25s | Manual trigger | Canceled | Original | |
| Documentdb | 11/16/2023, 5:52:24 PM | 11/16/2023, 6:19:03 PM | 26m 39s | Manual trigger | Succeeded | Original | |

Count of Data loaded in Azure Cosmos Document DB:



The screenshot shows the Microsoft Azure Data Explorer interface. The left sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Cost Management, Quick start, Notifications, Data Explorer (selected), and Settings. The main area displays a query result for 'Query 1'. The query is: `SELECT count(1) as total1_count FROM c`. The result shows a single row with the value '98666' for 'total1_count'.

| total1_count |
|--------------|
| 98666 |

Document DB loaded in Azure:

The screenshot displays the Microsoft Azure Data Explorer interface. The top navigation bar shows the Microsoft Azure logo and a search bar. The main header indicates the user is logged in as 'kanchan.dh@northeast...' from 'NORTHEASTERN UNIVERSITY'. The breadcrumb trail shows 'Home > Azure Cosmos DB > kanchan-dh'. The left sidebar shows the 'kanchan-dh | Data Explorer' view with a tree structure of the database. The 'DATA' section is expanded, showing 'BrazilE-commerce' > 'Scale' > 'OrderContainer' > 'Items'. The 'NOTEBOOKS' section is collapsed with a message: 'Notebooks is currently not available. We are working on it.' The main pane shows a table of data with columns 'id' and '/ord...'. The table contains 20 rows of data. The right pane shows a JSON document for one of the items, with a query editor above it showing a SELECT statement: 'SELECT * FROM c'. The JSON document is a complex nested structure representing an order item.

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB > kanchan-dh

kanchan-dh | Data Explorer

Azure Cosmos DB account

DATA

BrazilE-commerce

Scale

OrderContainer

Items

Settings

Stored Procedures

User Defined Functions

Triggers

ToDoList

Items

NOTEBOOKS

Notebooks is currently not available. We are working on it.

SELECT * FROM c

| id | /ord... |
|-------------|-------------|
| b925996... | 3e8a0bc... |
| c9ba830... | 4076d3e... |
| 685e9eaf... | 41c4425... |
| 69e6427... | 41eb344f... |
| 63dddad... | 41fe5122... |
| 457d569... | 424cd6d... |
| e139219... | 42d34c5... |
| 5d1d3f4... | 43aacfee... |
| 6575120a... | 4410b68... |
| 1e132e7... | 442651e... |
| d288d56... | 445e644... |
| b429ee3... | 44f3d1f0... |
| f115455c... | 451f91bc... |
| 7a9e4a9... | 457dbdc... |
| f1a26ade... | 4620a3e... |

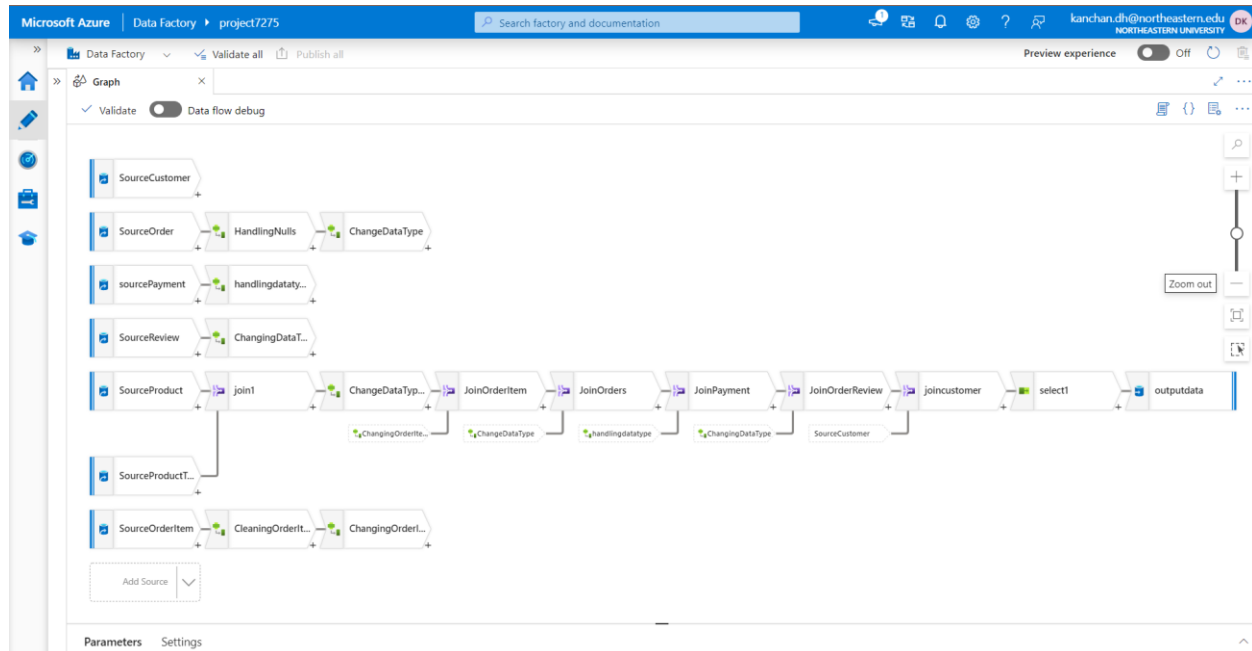
```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

```
{
  "order_id": "3e8a0bcdd82e790370f54bc991291d82",
  "customer_id": "c354ab5a93962e093ef3f3cde03b721",
  "order_status": "delivered",
  "order_purchase_timestamp": 1490400000000,
  "order_delivered_customer_date": 1491264000000,
  "order_estimated_delivery_date": 1493078400000,
  "OrderItemDoc": [
    {
      "order_item_id": 1,
      "freight_value": "26.69",
      "shipping_limit_date": 1490832000000,
      "price": 279,
      "ProductDoc": [
        {
          "product_id": "245269defe4105aea93d92bc62fb983d",
          "product_category_name": "automotivo",
          "product_height_cm": 25,
          "product_weight_g": 2900,
          "product_photos_qty": 1,
          "product_width_cm": 60,
          "product_length_cm": 33,
          "SellerDoc": [
            {
              "seller_id": "d2572f31e9023e9850ef986a636ff9bf",
              "seller_zip_code_prefix": "30640",
              "seller_city": "belo horizonte",
              "seller_state": "MG"
            }
          ]
        }
      ]
    }
  ]
}
```

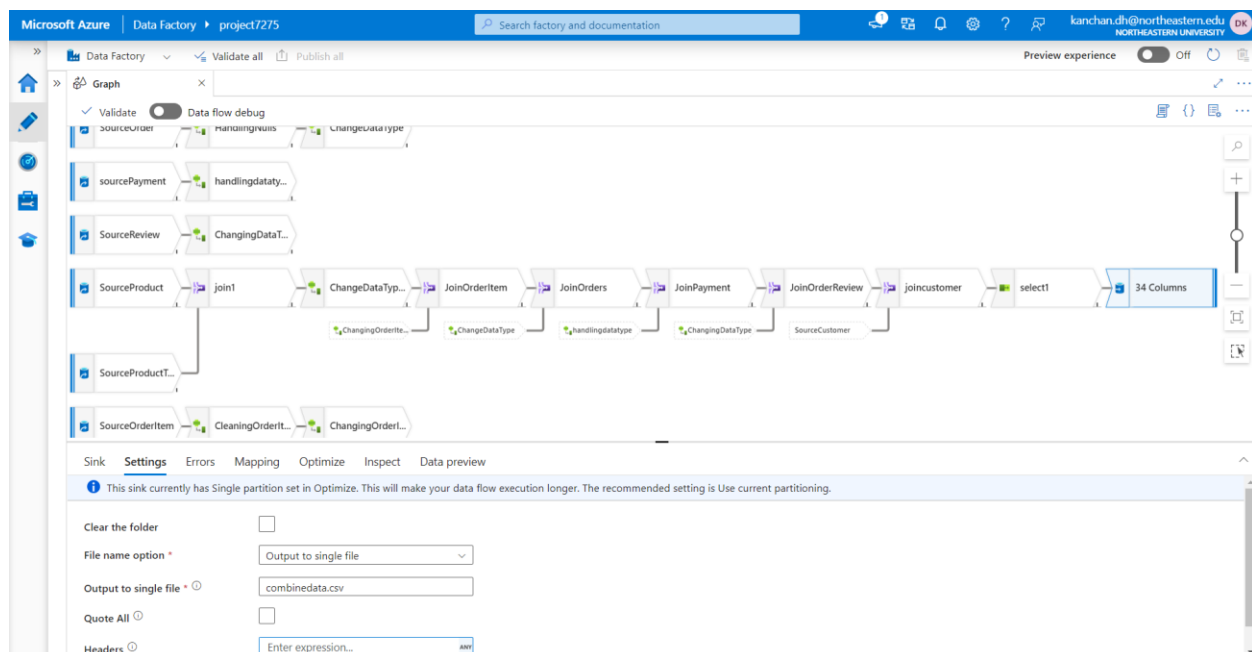
Revised Document DB Data model: (Combined Order, OrderItems, Products, Seller and Geolocation into one Document DB as per querying needs)

```
{
  "order_id":
  "customer_id":
  "order_status":
  "order_purchase_timestamp":
  "order_delivered_customer_date":
  "order_estimated_delivery_date":
  "OrderItemDoc": [
    {
      "order_item_id":
      "freight_value":
      "shipping_limit_date":
      "price":
      "ProductDoc": [
        {
          "product_id":
          "product_category_name":
          "product_height_cm":
          "product_weight_g":
          "product_photos_qty":
          "product_width_cm":
          "product_length_cm":
          "SellerDoc": [
            {
              "seller_id":
              "seller_zip_code_prefix":
              "seller_city":
              "seller_state":
            }
          ]
        }
      ]
    }
  ]
}
```

AZURE COSMOS GRAPH DOCUMENT DB implementation using GREMLIN API: Created a Dataflow in Azure Data Factory to perform some data cleansing and transformation processes on .csv files like: Removing Extra Columns, Handling Null Values in columns, Modifying the datetime, timestamp and other data types to required format and data type, combining all .csvs files into one file:



Generating combinedata.csv file:



Combined .csv file stored in Azure blob storage:

Home > storageproject12 | Containers >

source ...

Container

Search

Upload | Change access level | Refresh | Delete | Change tier | Acquire lease | Break lease | View snapshots | Create snapshot | Give feedback

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

Shared access tokens

Access policy

Properties

Metadata

Authentication method: Access key (Switch to Microsoft Entra user account)

Location: source

Search blobs by prefix (case-sensitive)

Show deleted blobs

Add filter

| | Name | Modified | Access tier | Archive status | Blob type | Size | Lease state | |
|--------------------------|---------------------------------------|-------------------------|----------------|----------------|------------|------------|-------------|-----|
| <input type="checkbox"/> | combinedata.csv | 11/25/2023, 12:01:12... | Hot (Inferred) | | Block blob | 43.57 MiB | Available | *** |
| <input type="checkbox"/> | olist_customers_dataset.csv | 11/14/2023, 3:06:46 ... | Hot (Inferred) | | Block blob | 8.62 MiB | Available | *** |
| <input type="checkbox"/> | olist_products_dataset.csv | 11/14/2023, 9:13:58 ... | Hot (Inferred) | | Block blob | 2.27 MiB | Available | *** |
| <input type="checkbox"/> | olist_sellers_dataset.csv | 11/14/2023, 9:13:58 ... | Hot (Inferred) | | Block blob | 170.61 KiB | Available | *** |
| <input type="checkbox"/> | product_category_name_translation.csv | 11/14/2023, 4:23:52 ... | Hot (Inferred) | | Block blob | 2.55 KiB | Available | *** |

Function for automatically identifying primary key, properties and creating Vertices with it's properties

```
def generate_gremlin_code(df, vertex_label: str, pk_column: str): #List[str]
    gremlin_code = []
    for i, row in df.iterrows():
        pk_value = str(row[pk_column])
        properties = []
        for col, value in row.items():
            if col != pk_column:
                value = value.replace("'", '"')
                properties.append(f'"{col}": "{value}"')
        properties = "".join(properties)
        vertex_code = f'g.addV("{vertex_label}").property("seller_id", "1").property("{pk_value}").property("_id", "{pk_value}")({properties})'
        gremlin_code.append(vertex_code)
    return gremlin_code
```

Python

Function for inserting Vertices:

```
warnings.filterwarnings('ignore')
def insert_vertices(client, all_vertices):
    for idx, query in enumerate(all_vertices):
        if idx % 800 == 0:
            time.sleep(3)
        try:
            client.submit(query)
        except protocol.GremlinServerError as e:
            print("Gremlin Server Error:", e.status_code)
        except Exception as e:
            print("An unexpected error occurred:====> ", query)
```

Python

Function for automatically identifying source & destination and creating Edges from dataset:

Function for automatically identifying source & destination and creating Edges from dataset:

```
def create_gremlin_query(df, col1, col2):
    queries = []
    for index, row in df.iterrows():
        source = row[col1]
        dest = row[col2]
        query = f'g.V('{source}').addE('has').to(g.V('{dest}'))'
        queries.append(query)
    return queries
```

Python

Function for inserting Edges:

```
def insert_edges(client, all_edges):
    for idx, query in enumerate(all_edges):
        if idx % 800 == 0:
            time.sleep(3)
        try:
            client.submit(query)
        except protocol.GremlinServerError as e:
            print("Gremlin Server Error:", e.status_code)
        except Exception as e:
            print("An unexpected error occurred:==> ", query)
```

Python

Establishing connection to Azure Cosmos DB from python using Gremlin API:

▼ Establishing connection with the azure cosmos db gremlin database(specified database)

```
def getClient():
    gremlin_client = client.Client(cosmos_db_endpoint, 'g',
                                   username=f'//dbs/{database_name}/colls/{graph_name}',
                                   password=cosmos_db_password,
                                   message_serializer=serializer.GraphSONSerializersV2d0())
    return gremlin_client
```

Python

Creating Dataframe for vertices and dropping duplicate values in the columns

```
order = dataFrame[['order_id', 'order_estimated_delivery_date', 'order_delivered_carrier_date', 'order_approved_at', 'order_purchase_timestamp', 'order_status']].copy()
print(f'Ord Loading... unique check: {len(order)}')
order = order.drop_duplicates(subset=['order_id'])
print(f'Order after unique check: {len(order)}')

product = dataFrame[['product_id', 'product_height_cm', 'product_length_cm', 'product_weight_g', 'product_photos_qty', 'product_category_name', 'product_width_cm']].copy()
print(f'Product before unique check: {len(product)}')
product = product.drop_duplicates(subset=['product_id'])
print(f'Product after unique check: {len(product)}')

review = dataFrame[['review_id', 'review_creation_date', 'review_score', 'review_answer_timestamp']].copy()
print(f'Review before unique check full-data: {len(review)}')
print(f'Review before unique check: {len(review['review_id'].unique())}')
review = review.drop_duplicates(subset=['review_id'])
print(f'Review before unique check full-data: {len(review)}')
print(f'Review before unique check: {len(review['review_id'].unique())}')

customer = dataFrame[['customer_id', 'customer_city', 'customer_state']].copy()
print(f'Customer before unique check full-data: {len(customer)}')
print(f'Customer before unique check: {len(customer['customer_id'].unique())}')
customer = customer.drop_duplicates(subset=['customer_id'])
print(f'Customer after unique check full-data: {len(customer)}')
print(f'Customer after unique check: {len(customer['customer_id'].unique())}')

payment = dataFrame[['order_id', 'payment_value', 'payment_sequential', 'payment_type', 'payment_installments']].copy()
payment['p_order_id'] = payment['order_id'] + payment['payment_sequential']
print(f'Payment before unique check full-data: {len(payment)}')
print(f'Payment before unique check: {len(payment['p_order_id'].unique())}')
payment = payment.drop_duplicates(subset=['p_order_id'])
print(f'Payment after unique check full-data: {len(payment)}')
print(f'Payment after unique check: {len(payment['p_order_id'].unique())}')
```

Python

Output:

```
payment = payment.drop_duplicates(subset=['p_order_id'])
print(f'Payment after unique check full-data: {len(payment)}')
print(f'Payment after unique check: {len(payment['p_order_id'].unique())}')

... Order before unique check: 115609
Order after unique check: 96516
Product before unique check: 115609
Product after unique check: 32171
Review before unique check full-data: 115609
Review before unique check: 96319
Review before unique check full-data: 96319
Review before unique check: 96319
Customer before unique check full-data: 115609
Customer before unique check: 96516
Customer after unique check full-data: 96516
Customer after unique check: 96516
Payment before unique check full-data: 115609
Payment before unique check: 100792
Payment after unique check full-data: 100792
Payment after unique check: 100792
```

creating dataframe for edge and dropping duplicate values

```
▷ order_product = dataframe[['order_id', 'product_id']].copy()
print(f'Order Product before unique check: {len(order_product)}')
order_product = order_product.drop_duplicates()
print(f'Order Product after unique check: {len(order_product)}')

order_review = dataframe[['review_id', 'order_id']].copy()
print(f'Order - Review before unique check full-data: {len(order_review)}')
order_review = order_review.drop_duplicates()
print(f'Order - Review before unique check full-data: {len(order_review)}')

order_customer = dataframe[['customer_id', 'order_id']].copy()
print(f'Order - Customer before unique check full-data: {len(order_customer)}')
order_customer = order_customer.drop_duplicates()
print(f'Order - Customer after unique check full-data: {len(order_customer)}')

order_payment = payment[['order_id', 'p_order_id']].copy()
print(f'Order - Payment before unique check full-data: {len(order_payment)}')
order_payment = order_payment.drop_duplicates()
print(f'Order - Payment after unique check full-data: {len(order_payment)}')

... Order Product before unique check: 115609
Order Product after unique check: 100157
Order - Review before unique check full-data: 115609
Order - Review before unique check full-data: 97055
Order - Customer before unique check full-data: 115609
Order - Customer after unique check full-data: 96516
Order - Payment before unique check full-data: 100792
Order - Payment after unique check full-data: 100792
```

Inserting vertices in Azure cosmos db

```
vertex_dfs = [{
    "df": order,
    "v_label": 'order',
    "v_key": 'order_id'
},
{
    "df": customer,
    "v_label": 'customer',
    "v_key": 'customer_id'
},
{
    "df": product,
    "v_label": 'product',
    "v_key": 'product_id'
},
{
    "df": review,
    "v_label": 'review',
    "v_key": 'review_id'
},
{
    "df": payment,
    "v_label": 'payment',
    "v_key": 'p_order_id'
}]

def vertex_worker(df, g_client, v_label, v_key):
    insert_vertices(g_client, generate_gremlin_code(df, v_label, v_key))

threads = []
for v_info in vertex_dfs:
    gremlin_client = getClient()
    t = threading.Thread(target=vertex_worker, args=(v_info["df"], gremlin_client, v_info["v_label"], v_info["v_key"]))
    threads.append(t)
    t.start()

for t in threads:
    t.join()
```

Inserting edges in Azure Cosmos DB

```
edges_dfs = [{
    "df": order_payment,
    "col1": 'order_id',
    "col2": 'p_order_id'
},
{
    "df": order_product,
    "col1": 'order_id',
    "col2": 'product_id'
},
{
    "df": order_review,
    "col1": 'order_id',
    "col2": 'review_id'
},
{
    "df": order_customer,
    "col1": 'order_id',
    "col2": 'customer_id'
}]

def edge_worker(df, g_client, col1, col2):
    insert_edges(g_client, create_gremlin_query(df, col1, col2))

threads = []
for e_info in edges_dfs:
    gremlin_client = getClient()
    t = threading.Thread(target=edge_worker, args=(e_info["df"], gremlin_client, e_info["col1"], e_info["col2"]))
    threads.append(t)
    t.start()

for t in threads:
    t.join()
```

Count of Vertices:

APACHE GREMLIN API

Home Graph x

DATA

- gremlinproject
 - Scale
 - test
 - Graph
 - Settings
 - Stored Procedures
 - User Defined Functions
 - Triggers

NOTEBOOKS

Notebooks is currently not available. We are working on it.

g.V().count()

Execute Gremlin Query x

JSON Query Stats

422314

Count of Edges:

APACHE GREMLIN API

Home Graph x

DATA

- gremlinproject
 - Scale
 - test
 - Graph
 - Settings
 - Stored Procedures
 - User Defined Functions
 - Triggers

NOTEBOOKS

Notebooks is currently not available. We are working on it.

g.E().count()

Execute Gremlin Query x

JSON Query Stats

394520

Plotting of graphs:

g.V()

Execute Gremlin Query x

JSON Graph Query Stats

Results

- 0433830caca22b01a0f477...
- 04808b9886b62b9c06b4c4...
- 060f0122cdc8f25db4a0d7...
- 0ad48bfc4a5619823e2b0f...
- 0c104824e973388d487c67...
- 0c4e43e25f781898363a83f...
- 0789be671fe3bd93cabfbb...
- 135fb255f9873811c6aa6eb...
- 13da041c237c948a2d42da...
- 16ff579c7dd128062ca085e...
- 1a20dc4f6c828ec4836526c...

Graph

0433830caca22b01a0f477d31307b043

Properties

- id: 0433830caca22b01a0f477d31307b043
- label: product
- seller_id: 1
- _id: 0433830caca22b01a0f477d31307b043
- product_height_cm: 10
- product_length_cm: 40
- product_weight_g: 7450
- product_photos_...: 7
- product_category_...: portateis_casa_forno_e_cafe
- product_width_cm: 40

Sources

Home

Graph

g.E()

Execute Gremlin Query

JSON

Graph

Query Stats

Results

0433830caca22b01a0f477...

04808b9886b62b9c06b4c4...

060f0122cdc8f25db4a0d7...

0ad48bfc4a5619823e2b0f...

0c104824e973388d487c67...

0c4e43e25f781898363a83f...

0f789be671fe3bd93cabfbb...

135fb255f9873811c6aa6eb...

13da041c237c948a2d42da...

16ff579c7dd128062ca085e...

1a20dc46c828ec4836526c...

Graph

0433830caca22b01a0f477d31307b043

Properties

id

0433830caca22b01a0f477d31307b043

label

product

seller_id

1

_id

0433830caca22b01a0f477d31307b043

product_height_cm

10

product_length_cm

40

product_weight_g

7450

product_photos_...

7

product_category...

portateis_casa_forno_e_cafe

product_width_cm

40

Sources

We have successfully implemented Document DB and Graph DB using NoSQL API and Gremlin API, respectively.