

P-4: Implementation

TOPIC: BRAZIL E-COMMERCE DATA ANALYSIS

TEAM: 3

Abbas Furniturewala - NUID 002193272

Shubham Idekar - NUID 002776415

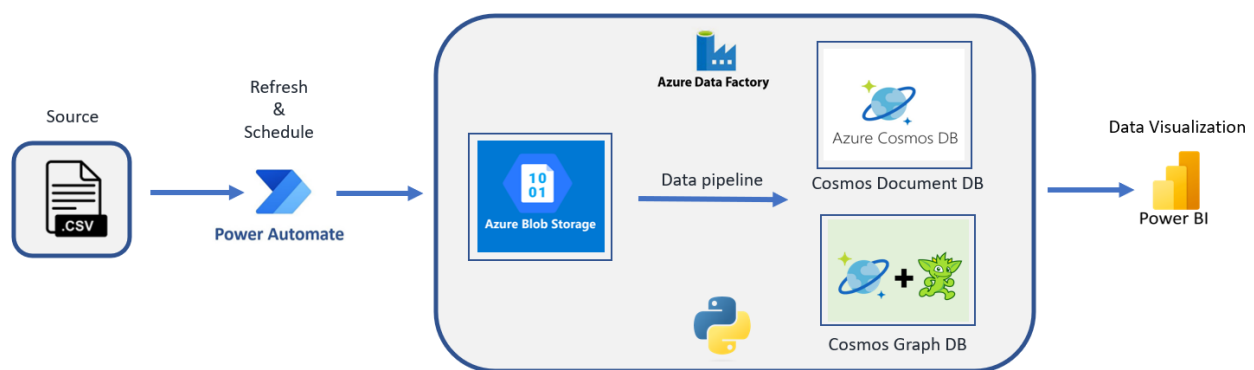
Shrutika Salian - NUID 002142365

Harshit Parikh - NUID 001044838

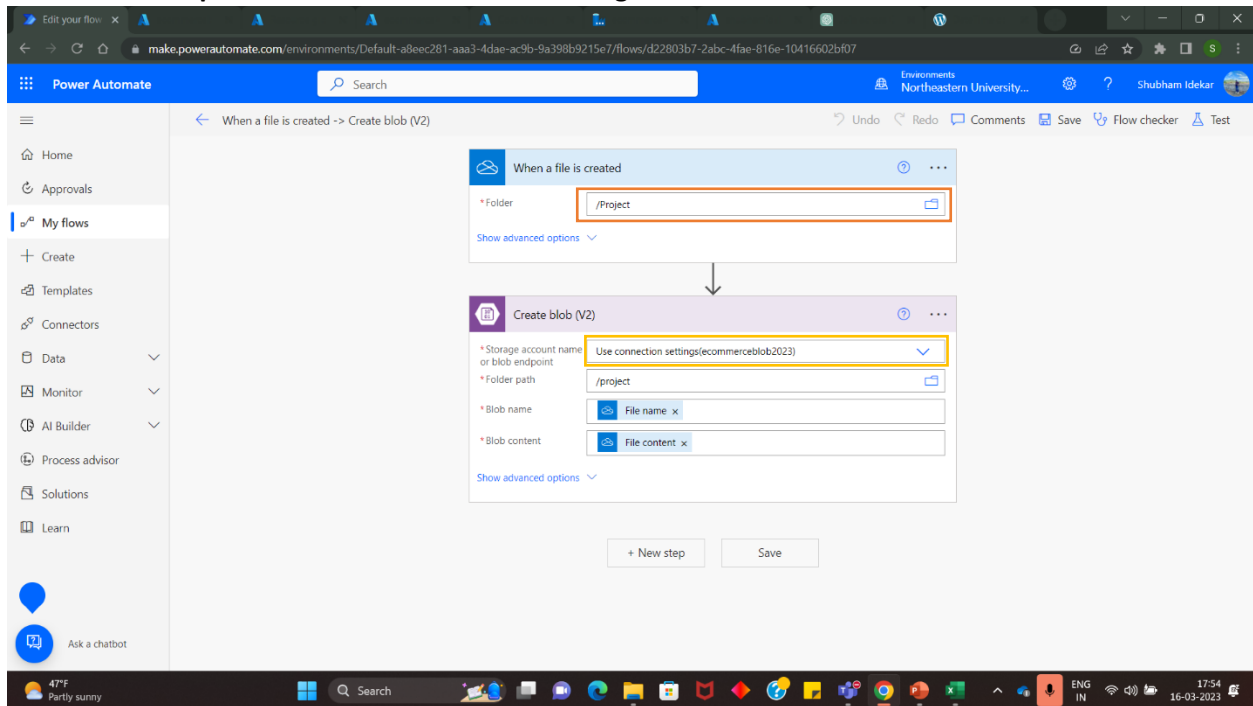
A brief description of the implementation process:

1. Connected Power Automate to Azure Blob Storage using the connection strings. As soon as a file is created in OneDrive folder, the file is fetched by Power Automate and placed into Azure Blob Storage.
2. For Azure Cosmos Document DB implementation, the files from blob are imported into AZURE DATA FACTORY in the Dataset section. 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. For Azure Graph DB implementation, we first use Azure data factory to combine the multiple .csv files required for graph implementation into one file. We established a connection from python to our azure blob storage where the combined file is placed to directly fetch from there. Then, established a connection from python to Azure Cosmos Graph DB using Gremlin API. Once the connection was established, we created the vertices and edges using the Gremlin API, ingesting data into Azure Cosmos Graph DB.

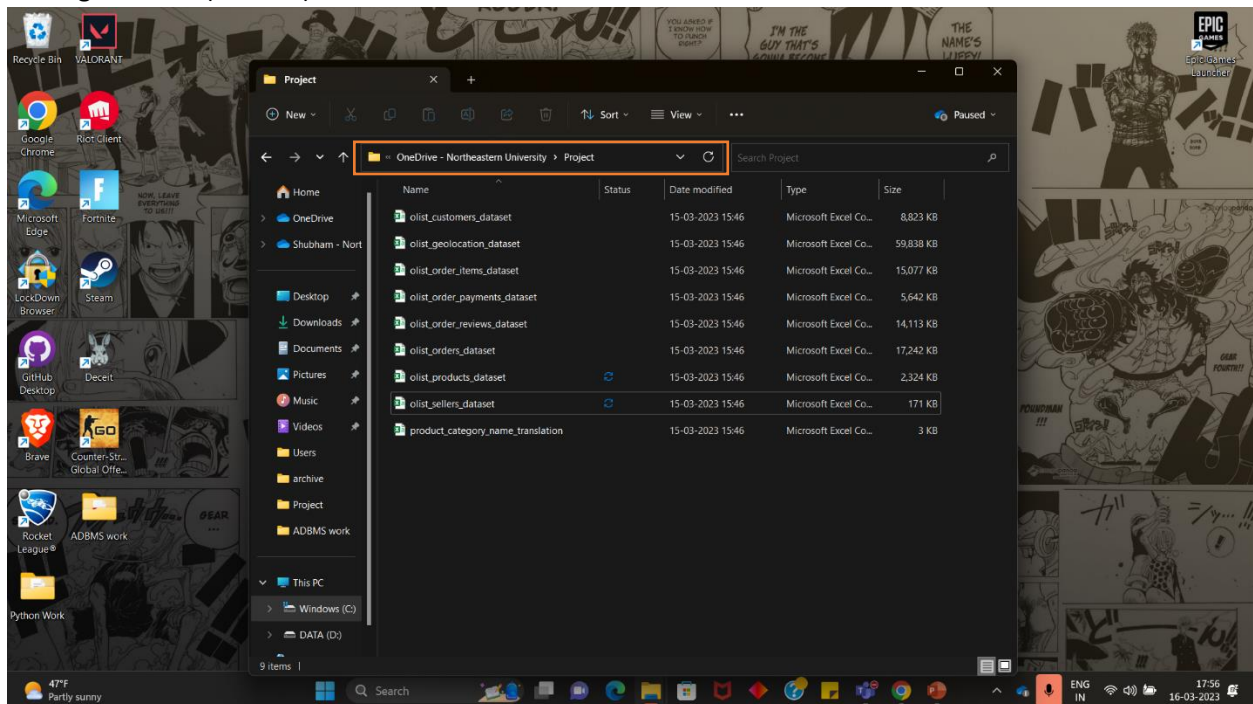
Revised Data Architecture Diagram:



Using **Power Automate Automated Cloud Flow** to **fetch files from OneDrive folder** whenever a **new file is created** and **upload the files to Azure Blob Storage**:



Placing .csv files (dataset) into the OneDrive folder:



Successful execution of Power Automate flow:

The screenshot displays the Microsoft Power Automate web interface. The top navigation bar includes the 'Power Automate' logo, a search bar, and the user's name 'Shubham Idekar'. The left sidebar shows the 'My flows' section, which is expanded to show a list of flows. The main area displays a table of flow runs with the following columns: 'Start time', 'Duration', and 'Status'. All runs are marked as 'Succeeded'.

Start time	Duration	Status
Mar 15, 04:06 PM (1 d ago)	00:00:04	Succeeded
Mar 15, 04:06 PM (1 d ago)	00:00:14	Succeeded
Mar 15, 04:05 PM (1 d ago)	00:00:14	Succeeded
Mar 15, 04:05 PM (1 d ago)	00:00:04	Succeeded
Mar 15, 04:05 PM (1 d ago)	00:00:03	Succeeded
Mar 15, 04:00 PM (1 d ago)	00:00:02	Succeeded
Mar 15, 03:54 PM (1 d ago)	881 ms	Succeeded
Mar 15, 03:49 PM (1 d ago)	777 ms	Succeeded

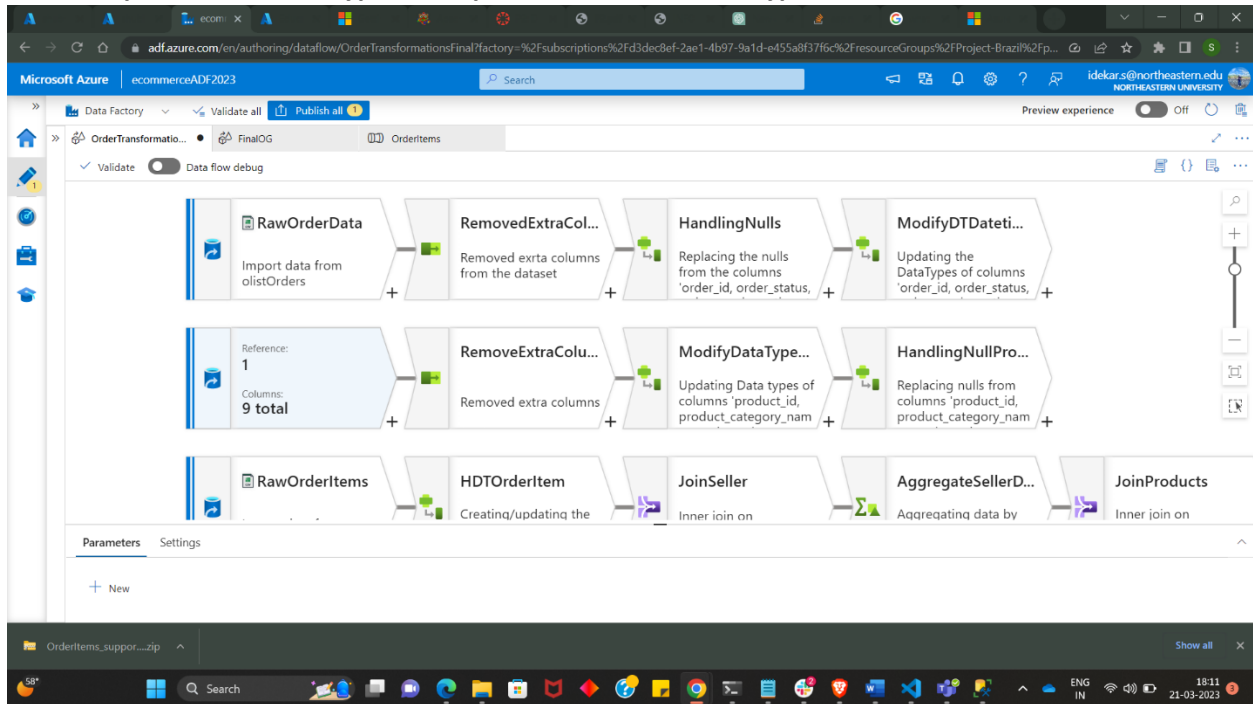
All files placed into Azure Blob Storage automatically once created in OneDrive Folder:

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the 'Microsoft Azure' logo, a search bar, and the user's name 'idekar.s@northeastern.e...'. The left sidebar shows the 'Containers' section, which is expanded to show a list of containers. The main area displays the 'project' container, which contains a list of files. The files are listed in a table with the following columns: 'Name', 'Modified', 'Access tier', 'Archive status', 'Blob type', 'Size', and 'Lease state'. All files are marked as 'Available'.

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
olist_customers_dataset.csv	3/15/2023, 4:06:19 PM	Hot (Inferred)		Block blob	8.62 MiB	Available
olist_geolocation_dataset.csv	3/15/2023, 6:51:16 PM	Hot (Inferred)		Block blob	58.44 MiB	Available
olist_order_items_dataset.csv	3/15/2023, 4:05:37 PM	Hot (Inferred)		Block blob	14.72 MiB	Available
olist_order_payments_dataset.csv	3/15/2023, 4:05:45 PM	Hot (Inferred)		Block blob	5.51 MiB	Available
olist_order_reviews_dataset.csv	3/15/2023, 4:06:20 PM	Hot (Inferred)		Block blob	13.78 MiB	Available
olist_orders_dataset.csv	3/15/2023, 4:06:08 PM	Hot (Inferred)		Block blob	16.84 MiB	Available
olist_products_dataset.csv	3/15/2023, 4:00:07 PM	Hot (Inferred)		Block blob	2.27 MiB	Available
olist_sellers_dataset.csv	3/15/2023, 3:54:23 PM	Hot (Inferred)		Block blob	170.61 KiB	Available
product_category_name_translation.csv	3/15/2023, 3:49:17 PM	Hot (Inferred)		Block blob	2.55 KiB	Available

AZURE COSMOS DOCUMENT DB implementation:

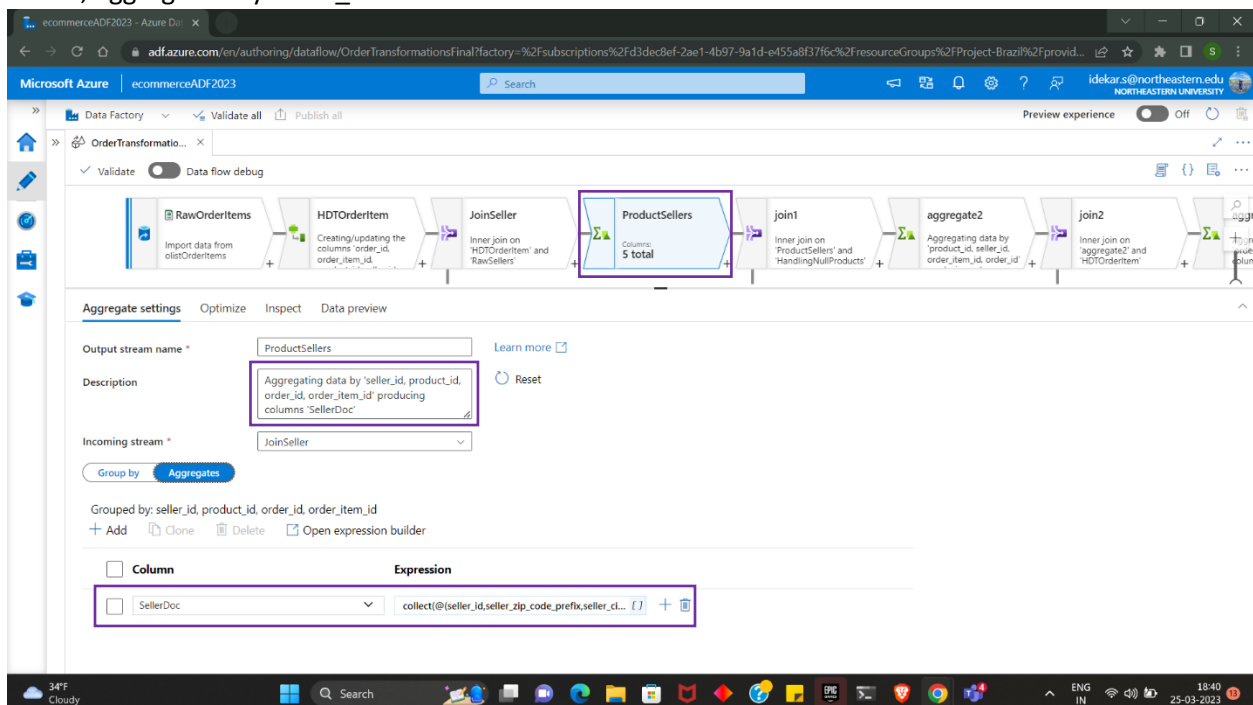
Created a **Dataflow** in **Azure Data Factory** to perform some **data cleansing and transformation processes** on .csv files like: **Removing Extra Columns, Handling Null Values in columns, Modifying the datetime, timestamp and other data types to required format and data type:**



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' interface. On the left, under 'Aggregate Columns', there is a list of columns including 'SellerDoc'. The main 'Expression' field contains the code: `collect(@(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 including 'order_id', 'order_item_id', 'product_id', 'RawOrderItems@seller_id', and 'shipping_limit_date'. At the bottom, there are buttons for 'Save and finish', 'Cancel', and 'Clear contents'.

Data Preview:

The screenshot shows the 'Data Preview' interface. At the top, there is a data flow diagram with several steps: 'JoinSeller', 'ProductSellers', 'JoinProduct', 'AggregateSellers...', 'JoinOrderItems', 'AggregateProduct...', and 'JoinOrders'. The 'Data preview' tab is active, showing a table with columns: 'seller_id', 'product_id', 'order_id', 'order_item_id', and 'SellerDoc'. The table contains 100 rows of data. The 'SellerDoc' column is highlighted, showing a detailed view of the data for a specific row.

Similarly, aggregated by respective ids to achieve the nested document db json format:

The screenshot shows the 'Aggregate settings' tab for the 'AggregateProductByOrderItems' activity in an Azure Data Factory pipeline. The 'Output stream name' is 'AggregateProductByOrderItems'. The 'Description' is 'Aggregating data by 'order_id' producing columns 'OrderItemDoc''. The 'Incoming stream' is 'JoinOrderItems'. The 'Grouped by' is 'order_id'. The 'Column' 'OrderItemDoc' is selected, and its 'Expression' is 'collect(@(order_item_id,shipping_limit_date,price,...))'. The pipeline diagram above shows the flow from 'JoinSeller' to 'ProductSellers', 'JoinProduct', 'AggregateSellers...', 'JoinOrderItems', 'AggregateProduct...', 'Join3', and finally 'Join'.

Data Preview after last join and aggregation:

The screenshot shows the 'Data preview' tab for the 'AggregateProductByOrderItems' activity. It displays a table with 13 rows. The first column is 'order_id'. The second column, 'OrderItemDoc', contains a JSON array of objects. One object is highlighted, showing 'abc_order_item_id: 1', 'shipping_limit_date: 1519306528000', 'price: 56', and 'freight_value: 14'. Below this, there is a 'ProductDoc' field with an empty array. The pipeline diagram above shows the flow from 'JoinSeller' to 'ProductSellers', 'JoinProduct', 'AggregateSellers...', 'JoinOrderItems', 'AggregateProduct...', 'Join3', and finally 'Join'.

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

The screenshot displays the Azure Data Factory (ADF) pipeline editor interface. The pipeline is named 'OrderTransformationsFinal' and is in the 'Sink' tab. The pipeline flow is as follows:

- AggregateSellersByProduct**: Aggregating data by 'product_id', 'seller_id', 'order_item_id', and 'order_id'.
- JoinOrderItems**: Inner join on 'AggregateSellersByProduct' and 'HDTOrderItem'.
- AggregateProductByOrderItem**: Aggregating data by 'order_id' producing columns 'OrderItemDoc'.
- JoinOrders**: Inner join on 'AggregateProductByOrderItems' and 'ModifyDTDateTime'.
- LoadInCosmosDB**: Sink activity with 'Columns: 8 total'.

The 'Sink' configuration for 'LoadInCosmosDB' is shown below:

- Output stream name**: LoadInCosmosDB
- Description**: Export data to OrderItemsCosmosDB
- Incoming stream**: JoinOrders
- Sink type**: Dataset
- Dataset**: OrderItemsCosmosDB
- Options**: ☒ Allow schema drift, ☐ Validate schema

Running the data pipeline:

The screenshot displays the Azure Data Factory (ADF) pipeline editor interface. The pipeline is named 'OrderItems' and is in the 'Data flow' tab. The data flow is named 'FinalDocModel'.

The 'Settings' tab for the 'FinalDocModel' data flow is shown below:

- Data flow**: OrderTransformationsFinal
- Run on (Azure IR)**: AutoResolveIntegrationRuntime
- Compute size**: Small
- Logging level**: ☒ Verbose, ☐ Basic, ☐ None

Successful execution of the data pipeline:

The screenshot displays the 'Pipeline runs' section in the Microsoft Azure portal. The left sidebar shows the navigation menu with 'Pipeline runs' selected. The main area shows a table of pipeline runs. The first run, named 'OrderItems', started on 3/21/2023 at 12:29:47 AM and ended at 12:55:36 AM, with a duration of 00:25:49. The status is 'Succeeded' and it was triggered by a 'Manual trigger'. The 'Run' column shows 'Original'.

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Error	Run
OrderItems	3/21/2023, 12:29:47 AM	3/21/2023, 12:55:36 AM	00:25:49	Manual trigger	Succeeded		Original

Count of Data loaded in Azure Cosmos Document DB:

The screenshot displays the 'Data Explorer' section in the Microsoft Azure portal for an Azure Cosmos DB account. The left sidebar shows the navigation menu with 'Data Explorer' selected. The main area shows a query editor with the query 'SELECT count(1) total_count FROM c'. The results tab shows a single result: 'total_count': 98666.

```
SELECT count(1) total_count FROM c
```

Results
{ "total_count": 98666 }

Document DB loaded in Azure:

Microsoft Azure

Search resources, services, and docs (G+)

Home > Azure Cosmos DB > shubhamidekar

shubhamidekar | Data Explorer

Azure Cosmos DB account

NOSQL API

DATA

▼ commerceCosmosDB

Scale

OrderItems

Items

Settings

Stored Procedures

User Defined Functions

Triggers

NOTEBOOKS

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

Triggers

NOTEBOOKS

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

Triggers

NOTEBOOKS

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

Home

OrderItems - It...

Query 1 x

1 SELECT * FROM c

Results

Query Stats

1 - 100 | Load more

```
{
  "order_status": "delivered",
  "order_delivered_customer_date": 1500655425000,
  "order_purchase_timestamp": 1499366730000,
  "order_estimated_delivery_date": 1501804000000,
  "id": "df9b0ba6-779b-4c0a-bd8e-28a5f0bb21d8",
  "order_id": "A6alac51daefad9fb8e65caacd6435339",
  "OrderItemDoc": [
    {
      "order_item_id": "1",
      "price": 50,
      "ProductDoc": [
        {
          "product_category_name": "utilidades_domesticas",
          "product_height_cm": 32,
          "product_id": "078680744a44896ee0a59373edfcc50",
          "product_weight_g": 3500,
          "product_photos_qty": 1,
          "product_width_cm": 40,
          "SellerDoc": [
            {
              "seller_city": "anaguari",
              "seller_state": "MG",
              "seller_zip_code_prefix": "38442",
              "seller_id": "8ebcc67478bedae941d9bd9eb7f8d7fa"
            }
          ],
          "product_length_cm": 40
        }
      ]
    },
    {
      "freight_value": 19,
      "shipping_limit_date": 1499914266000
    }
  ],
  "order_item_id": "2",
  "price": 51,
  "ProductDoc": [
    {
      "product_category_name": "utilidades_domesticas",
      "product_height_cm": 35,
      "product_id": "3ff490455b099053cba94a2dab6f814d",

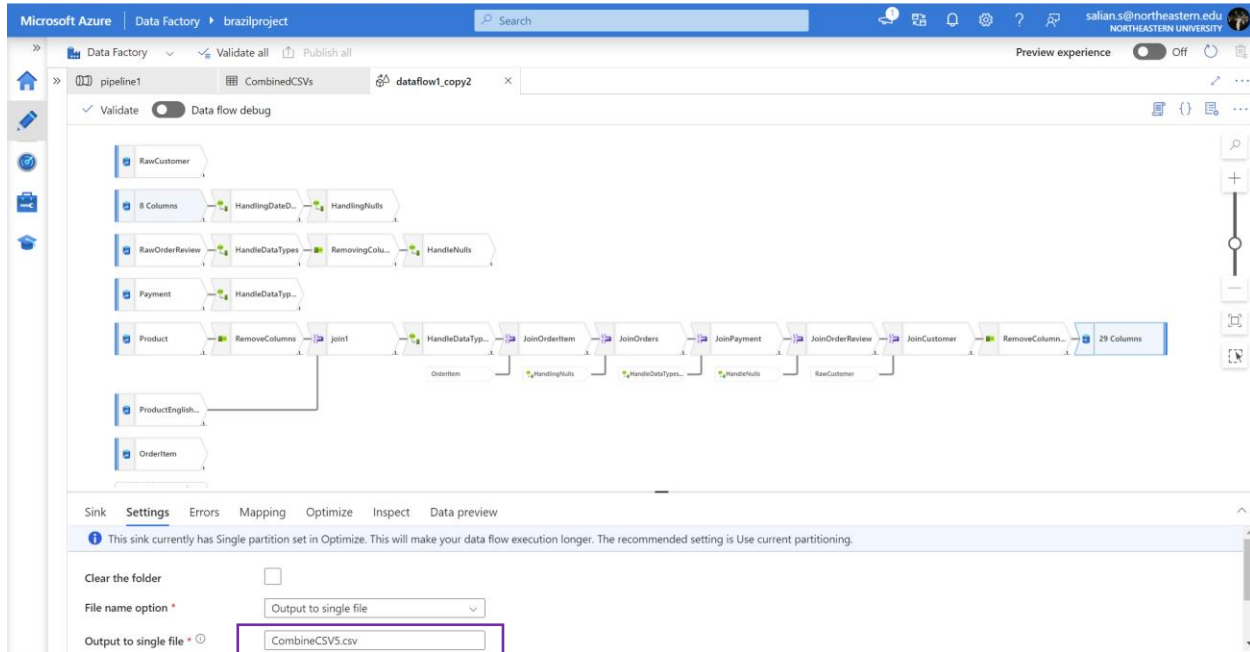
```

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

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

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:**



Combined .csv file stored in Azure blob storage:

The screenshot shows the Microsoft Azure portal interface for the 'brazilproject7275' container. The table lists various CSV files stored in the container. The 'CombineCSV5.csv' file is highlighted with a red box.

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
.._SUCCESS	3/24/2023, 6:21:51 PM	Hot (Inferred)		Block blob	0 B	Available
CombineCSV5.csv	3/25/2023, 8:16:02 PM	Hot (Inferred)		Block blob	56.79 MiB	Available
olist_customers_dataset.csv	3/21/2023, 4:59:43 PM	Hot (Inferred)		Block blob	8.62 MiB	Available
olist_geolocation_dataset.csv	3/21/2023, 4:59:50 PM	Hot (Inferred)		Block blob	58.44 MiB	Available
olist_order_items_dataset.csv	3/21/2023, 4:59:36 PM	Hot (Inferred)		Block blob	14.72 MiB	Available
olist_order_payments_dataset.csv	3/21/2023, 4:59:31 PM	Hot (Inferred)		Block blob	5.51 MiB	Available
olist_order_reviews_dataset - Copy.xlsx	3/21/2023, 6:01:23 PM	Hot (Inferred)		Block blob	9.69 MiB	Available
olist_order_reviews_dataset.csv	3/21/2023, 4:59:45 PM	Hot (Inferred)		Block blob	13.78 MiB	Available
olist_order_reviews_dataset.tsv	3/21/2023, 5:55:02 PM	Hot (Inferred)		Block blob	13.37 MiB	Available
olist_orders_dataset.csv	3/21/2023, 4:59:43 PM	Hot (Inferred)		Block blob	16.84 MiB	Available
olist_products_dataset.csv	3/21/2023, 4:59:37 PM	Hot (Inferred)		Block blob	2.27 MiB	Available
olist_sellers_dataset.csv	3/21/2023, 4:59:37 PM	Hot (Inferred)		Block blob	170.61 KiB	Available

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

```
def generate_gremlin_code(df, vertex_label: str, pk_column: str) -> List[str]:
    """
    Generate Gremlin code to insert vertices for each row in a pandas DataFrame

    Args:
    - df: pandas DataFrame containing the data to insert
    - vertex_label: str, the label of the vertex to create
    - pk_column: str, the name of the column containing the primary key

    Returns:
    - List of strings, each string contains the Gremlin code to insert a single vertex
    """
    gremlin_code = []
    for i, row in df.iterrows():
        pk_value = str(row[pk_column])
        properties = []
        for col, value in row.iteritems():
            if col != pk_column:
                properties.append(f".property('{col}', '{value}')"")
        properties = "".join(properties)
        vertex_code = f"g.addV('{vertex_label}').property('id', '{pk_value}').property('_id', '{pk_value}'){{properties}}"
        gremlin_code.append(vertex_code)
    return gremlin_code
```

Function for inserting Vertices:

```
def insert_vertices(client, all_vertices):
    """
    The function "insert_vertices" takes two arguments: "client" and "all_vertices".

    Args:
    - client: client object used to submit queries to a graph database.
    - all_vertices: list of queries that insert vertices into the graph database.

    The function iterates over each query in the "all_vertices" list and submits it to the database using the client object.

    Returns:
    - The result of each query is checked for errors and if successful, the inserted vertex is printed.
    - The function also calls a helper function "print_status_attributes" to print the status and attributes of the inserted vertex.
    """
    for query in all_vertices:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this vertex:\n\t{0}".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query: {0}".format(query))
        print("\n")
        print_status_attributes(callback.result())
        print("\n")

    print("\n")
```

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

```
def create_gremlin_query(df, col1, col2):
    """
    Generate Gremlin code to insert edges for each row in a pandas DataFrame

    Args:
    - df: pandas DataFrame containing the edges to insert
    - col1: Primary Key 1 of Table which is the source edge
    - col2: Primary Key 2 of Table which is the destination edge

    Returns:
    - List of strings, each string contains the Gremlin code to insert a single edge
    """
    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
```

Function for inserting Edges:

```
def insert_edges(client, all_edges):
    """
    The function "insert_edges" takes two arguments: "client" and "all_edges".
    Args:
    - client: client object used to submit queries to a graph database.
    - all_edges: list of queries that insert edges into the graph database.

    The function iterates over each query in the "all_edges" list and submits it to the database using the client object.

    Returns:
    - The result of each query is checked for errors and if successful, the inserted vertex is printed.
    - The function also calls a helper function "print_status_attributes" to print the status and attributes of the inserted vertex.
    """
    for query in all_edges:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this edge:\n\t{0}\n".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query:\n\t{0}\n".format(query))
            print_status_attributes(callback.result())
            print("\n")

    print("\n")
```

Establishing connection to Azure Blob Storage from python to fetch the combined csv dataset file:

```
# Connect to Azure Blob storage and download the incoming file from Azure Blob
blob = BlobClient(account_url="https://azureblob7275.blob.core.windows.net",
                  container_name="brazilproject7275",
                  blob_name="CombineCSV5.csv",
                  credential="t9cuuj3ZECg==")

with open("CombineCSV5.csv", "wb") as f:
    data = blob.download_blob()
    data.readinto(f)

# Rename the file
os.rename("CombineCSV5.csv", "gremlin_project.csv")
```

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

```
try:
    client = client.Client('wss://azure-gremlin-hp.gremlin.cosmos.azure.com:443/', 'g',
                           username="/dbs/gremlin_test/colls/Person1",
                           password="vor028",
                           message_serializer=serializer.GraphSONSerializersV2d0()
                           )

    print("Welcome to Azure Cosmos DB + Gremlin on Python!")

    print("Let's drop everything first!")
    cleanup_graph(client)
```

Calling the functions to insert Vertices and Edges:

```
insert_vertices(client, generate_gremlin_code(order, 'order', 'order_id'))
insert_vertices(client, generate_gremlin_code(product, 'product', 'product_id'))
insert_vertices(client, generate_gremlin_code(order_review, 'order_review', 'order_review_SK'))
insert_vertices(client, generate_gremlin_code(customer, 'customer', 'customer_id'))
insert_vertices(client, generate_gremlin_code(payment, 'payment', 'payment_SK'))

insert_edges(client, create_gremlin_query(order_product, 'order_id', 'product_id')) # Order and Product
insert_edges(client, create_gremlin_query(order_payment, 'order_id', 'payment_SK')) # Order and Payment
insert_edges(client, create_gremlin_query(order_customer, 'order_id', 'customer_id')) # Order and Customer
insert_edges(client, create_gremlin_query(order_order_review, 'order_id', 'order_review_SK')) # Order and Order Review
```

Executing the script in command prompt:

Vertices being successfully created:

```
C:\Windows\System32\cmd.exe
Response status_attributes:
{'x-ms-status-code': 200, 'x-ms-activity-id': '724f6823-5caa-43f5-ac6c-ff5b2507ae66', 'x-ms-request-charge': 16.95, 'x-ms-total-request-charge': 16.95, 'x-ms-server-time-ms': 7.1836, 'x-ms-total-server-time-ms': 7.1836}

> g.addV('order').property('id', '04bef5ebaab368e72d66decdad88fddf').property('id', '04bef5ebaab368e72d66decdad88fddf').property('customer_id', '55ca07ff32dc30f00ded4c99f8ea9124').property('order_status', 'delivered').property('order_purchase_timestamp', '7/6/2017 16:54').property('order_approved_at', '7/6/2017 17:10').property('order_delivered_carrier_date', '7/19/2017 12:57').property('order_delivered_customer_date', '7/19/2017 12:57').property('order_estimated_delivery_date', '8/1/2017 0:00')

Inserted this vertex:
[{'id': '04bef5ebaab368e72d66decdad88fddf', 'label': 'order', 'type': 'vertex', 'properties': {'_id': [{'id': '04bef5ebaab368e72d66decdad88fddf_id', 'value': '04bef5ebaab368e72d66decdad88fddf'}], 'customer_id': [{'id': '291a624-7357-4de9-9905-8c6d05c55ab6', 'value': '55ca07ff32dc30f00ded4c99f8ea9124'}], 'order_status': [{'id': 'e04b00e1-90a4-4324-9e45-866a47553511', 'value': 'delivered'}], 'order_purchase_timestamp': [{'id': 'f5c21864-30f4-4b81-938b-de4aceec8d6d', 'value': '7/6/2017 16:54'}], 'order_approved_at': [{'id': '5465cbe6-4f02-472d-95a0-21af74142a13', 'value': '7/6/2017 17:10'}], 'order_delivered_carrier_date': [{'id': '72ac7fbb-cf2b-4169-b97-cd17603c4766', 'value': '7/19/2017 12:57'}], 'order_delivered_customer_date': [{'id': '20d95f80-4ed0-43d3-a3b8-b6f8aab57800', 'value': '7/19/2017 12:57'}], 'order_estimated_delivery_date': [{'id': 'c097019e-c2fb-4f6b-91a7-a5a9e1813b2f', 'value': '8/1/2017 0:00'}]}]]

Response status_attributes:
{'x-ms-status-code': 200, 'x-ms-activity-id': 'b1c628f0-dd61-458d-9d65-95676d08fa88', 'x-ms-request-charge': 16.95, 'x-ms-total-request-charge': 16.95, 'x-ms-server-time-ms': 6.547, 'x-ms-total-server-time-ms': 6.547}

> g.addV('order').property('id', '04e00ba23c33890eae39b02e8185cc2').property('id', '04e00ba23c33890eae39b02e8185cc2').property('customer_id', '291fd0287b7cfd75702b9be21f23f1e3').property('order_status', 'delivered').property('order_purchase_timestamp', '7/25/2018 21:36').property('order_approved_at', '7/25/2018 21:45').property('order_delivered_carrier_date', '7/30/2018 19:03').property('order_delivered_customer_date', '7/30/2018 19:03').property('order_estimated_delivery_date', '8/22/2018 0:00')

Inserted this vertex:
[{'id': '04e00ba23c33890eae39b02e8185cc2', 'label': 'order', 'type': 'vertex', 'properties': {'_id': [{'id': '04e00ba23c33890eae39b02e8185cc2_id', 'value': '04e00ba23c33890eae39b02e8185cc2'}], 'customer_id': [{'id': '0e58ec54-e76d-41e5-8c6b-9eb7fae31769', 'value': '291fd0287b7cfd75702b9be21f23f1e3'}], 'order_status': [{'id': '06695dd6-8f39-481c-b1c1-616bf3c18ec8', 'value': 'delivered'}], 'order_purchase_timestamp': [{'id': '47fb80c4-80bf-4c58-8c2d-e71cf9aad06b', 'value': '7/25/2018 21:36'}], 'order_approved_at': [{'id': 'f6c0baef-d1bc-48fd-bc24-4eb18a9046c', 'value': '7/25/2018 21:45'}], 'order_delivered_carrier_date': [{'id': '4f1d8bb7-fe08-4ba5-bd24-10186f06c88b', 'value': '7/30/2018 19:03'}], 'order_delivered_customer_date': [{'id': 'a3e2c7ff-be7d-4fae-85ca-d6eb51b6109f', 'value': '7/30/2018 19:03'}], 'order_estimated_delivery_date': [{'id': '4ce4a4d3-17c5-4356-a60d-dcf57e672f55', 'value': '8/22/2018 0:00'}]}]]

Response status_attributes:
{'x-ms-status-code': 200, 'x-ms-activity-id': '026cc0c4-101c-40ed-860e-a464e5686e83', 'x-ms-request-charge': 16.95, 'x-ms-total-request-charge': 16.95, 'x-ms-server-time-ms': 6.5956, 'x-ms-total-server-time-ms': 6.5956}
```

Edges being sucessfully created:

```

> g.V('cd725c256dd60e25a721e1fe8cccd6aa6').addE('has').to(g.V('dc9674016642e9b975996058421c2976_4.0'))

Inserted this edge:
[{'id': '14590dc2-041f-4977-acd2-72fd2d3e8335', 'label': 'has', 'type': 'edge', 'inVLabel': 'order_review', 'outVLabel': 'order', 'inV': 'dc9674016642e9b975996058421c2976_4.0', 'outV': 'cd725c256dd60e25a721e1fe8cccd6aa6'}]

Response status_attributes:
{'x-ms-status-code': 200, 'x-ms-activity-id': '05642959-9c74-4eb4-853c-49c7753c5abe', 'x-ms-request-charge': 14.420000000000002, 'x-ms-total-request-charge': 14.420000000000002, 'x-ms-server-time-ms': 13.2569, 'x-ms-total-server-time-ms': 13.2569}

> g.V('ce08f40b106df255b226e0cf2df6a11f').addE('has').to(g.V('464df9bea60d90a4b8eae55e60759b20_5.0'))

Inserted this edge:
[{'id': '77c46049-e339-49d4-a332-6d880f036df1', 'label': 'has', 'type': 'edge', 'inVLabel': 'order_review', 'outVLabel': 'order', 'inV': '464df9bea60d90a4b8eae55e60759b20_5.0', 'outV': 'ce08f40b106df255b226e0cf2df6a11f'}]

Response status_attributes:
{'x-ms-status-code': 200, 'x-ms-activity-id': '4e42a54e-c920-4964-abc1-03893f21cf73', 'x-ms-request-charge': 14.420000000000002, 'x-ms-total-request-charge': 14.420000000000002, 'x-ms-server-time-ms': 13.359, 'x-ms-total-server-time-ms': 13.359}

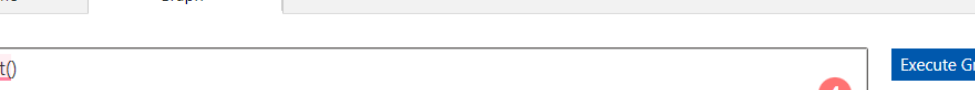
> g.V('ce11340d59020c4c840f12096c81b609').addE('has').to(g.V('006862652d256edde8f62c5d7792ee50_5.0'))

Inserted this edge:
[{'id': 'a4ed3729-3d22-44df-b34e-652f8a1b2042', 'label': 'has', 'type': 'edge', 'inVLabel': 'order_review', 'outVLabel': 'order', 'inV': '006862652d256edde8f62c5d7792ee50_5.0', 'outV': 'ce11340d59020c4c840f12096c81b609'}]

Response status_attributes:

```

Count of Vertices:



The screenshot shows the Gremlin console interface. At the top, there are tabs for 'Home' and 'Graph'. The 'Graph' tab is active. Below the tabs is a text input field containing the query `g.V().count()`. To the right of the input field is a red circle with the number '4' inside, indicating the number of results. To the right of the input field is a blue button labeled 'Execute Gremlin Query'. Below the input field, there are two tabs: 'JSON' and 'Query Stats'. The 'JSON' tab is selected. Below the 'JSON' tab, there is a green box highlighting the result `[391515]`.

Home Graph x

`g.V().count()`

4

Execute Gremlin Query x

JSON Query Stats

[391515]

Count of Edges:

Home

Graph x

`g.E().count()`

4

Execute Gremlin Query x

JSON

Query Stats

403161

Azure Cosmos Graph DB:

An Order having multiple nodes: Customer, Product Payment, OrderReview

JSON

Graph

Query Stats

`g.V().has("label", "order").has("id", "8272b63d03f5f79c56e9e4120aec44ef")`

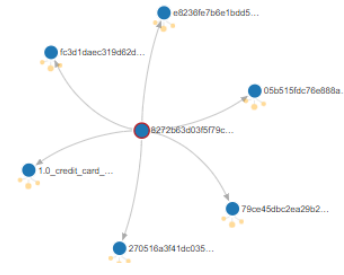
7

Execute Gremlin Query x

Results

8272b63d03f5f79c56e9e4...

Graph




> 8272b63d03f5f79c56e9e4120aec44ef

Properties

id	8272b63d03f5f79c56e9e4120aec44ef
label	order
_id	8272b63d03f5f79c56e9e4120aec44ef
customer_id	fc3d1daec319d62d49bfb5e1f83123e9
order_status	delivered
order_purchase_ti...	7/16/2017 18:19
order_approved_at	7/17/2017 18:25
order_delivered_c...	7/31/2017 18:03

The same Customer:

Results
fc3d1daec319d62d49bfb5...

Graph


> fc3d1daec319d62d49bfb5e1f83123e9
Properties
id: fc3d1daec319d62d49bfb5e1f83123e9
label: customer
_id: **fc3d1daec319d62d49bfb5e1f83123e9**
customer_zip_cod...: 5882
customer_city: sao paulo
customer_state: SP
Sources

Source	Edge label
8272b63d03f5f79c56e9e4120aec44ef	has

Targets
No targets found


Same Customer having the Order:

Home Graph X

g.V().has("label", "customer").has("id", "fc3d1daec319d62d49bfb5e1f83123e9")

JSON Graph Query Stats

Results
fc3d1daec319d62d49bfb5...

Graph


> 8272b63d03f5f79c56e9e4120aec44ef
Properties
id: 8272b63d03f5f79c56e9e4120aec44ef
label: order
_id: **8272b63d03f5f79c56e9e4120aec44ef**
customer_id: **fc3d1daec319d62d49bfb5e1f83123e9**
order_status: delivered
order_purchase_ti...: 7/16/2017 18:19
order_approved_at: 7/17/2017 18:25
order_delivered_c...: 7/31/2017 18:03
order_delivered_c...: 7/31/2017 18:03
order_estimated_...: 7/28/2017 0:00

A Product having Multiple Orders:

```
g.V().has("label", "product").has("id", "001b72dfd63e9833e8c02742adf472e3")
```

Execute Gremlin Query

JSON


Graph

Query Stats

Results

001b72dfd63e9833e8c027...

Graph



> 001b72dfd63e9833e8c02742adf472e3

Source	Edge label
7c644ea34ca017627bdfc5a5465a14a3	has
c214276ccd69c3953f880b487209f47e	has
70ed857e24fd6bf1e25a9bc791a2f6b9	has
4ce789d7705c17767436b6931e654637	has
82de2fedb028c674daa827d7984fea98	has
544317d6ff28d37495220e2bf024a1ae	has
a70705507f1747c0201350acad9d1408	has
15f18fdc945de59e900faa4d7fab8f15	has
9de93c000ca70a1ffac63cefe715bdf2	has

In this way, we have successfully implemented Document DB and Graph DB using NoSQL API and Gremlin API, respectively.