Lab DynamoDB - Deliverable 1

Marino Oliveros Blanco NIU:1668563
Pere Mayol Carbonell NIU:1669503

Github Link: https://github.com/marinocom/Cloud-Lab-DM-GIA

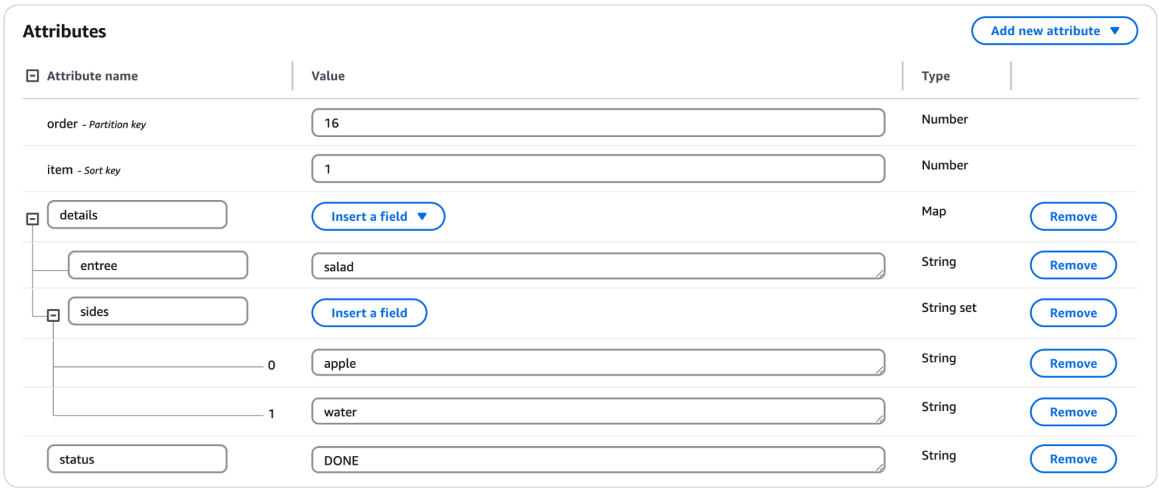
Session 1:

Create a dynamoDB database following the ppt file called “exampleDynamoDB.ppt”.
Use orders.txt file to create a database and apply the queries to the database.

Deliverable: A document explaining the work carried out, demonstrating a clear understanding of each step in the tutorial. The document must also include the results of the queries to confirm that the database is accurate.

Set-up

After launching the AWS Learning Lab and setting up the DynamoDB we insert the orders from the ‘orders.txt’ file by hand. We do it both by inserting the attributes in the Add items toggle in the console and also by adding them as json text. I recall some of the items we added by the attributes some by json text/view to check if it was working successfully, of course not adding them twice.



Example of adding an item to the DynamoDB table

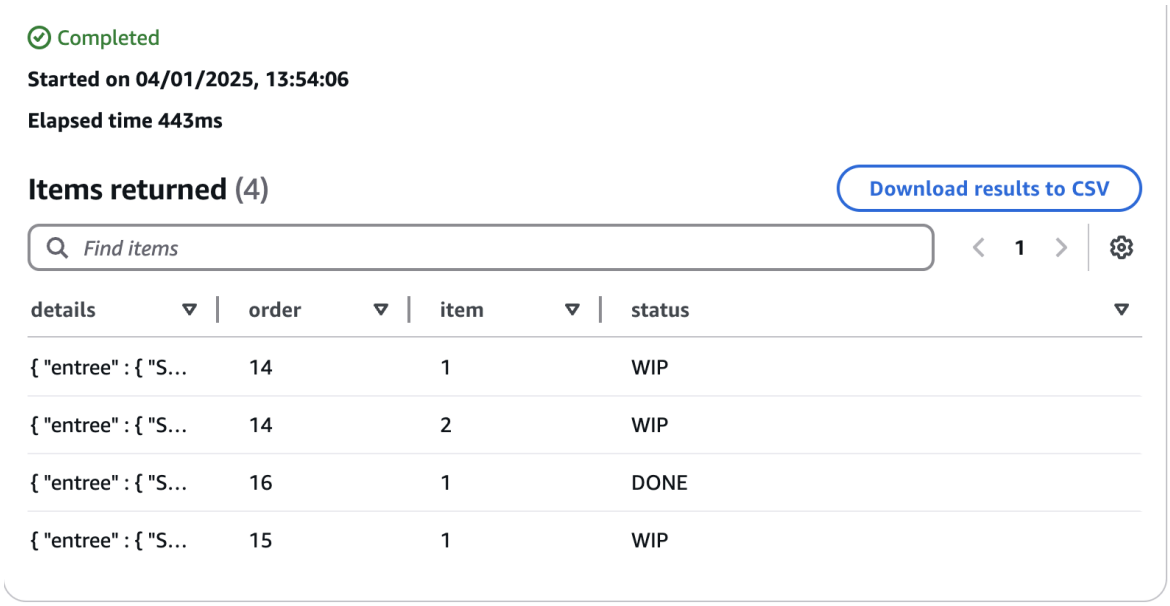
Example of adding an item to the DynamoDB table in Json view

Queries & PartiQL

We continued our task by resolving the 8 queries posed in the presentation 'dynamo-DB-intro.2425', for which we used the PartiQL editor. Simple query input output using a SQL query. The query results are shown as a screenshot of the PartiQL editor result in table view.

Query 1: Show all data items for orders

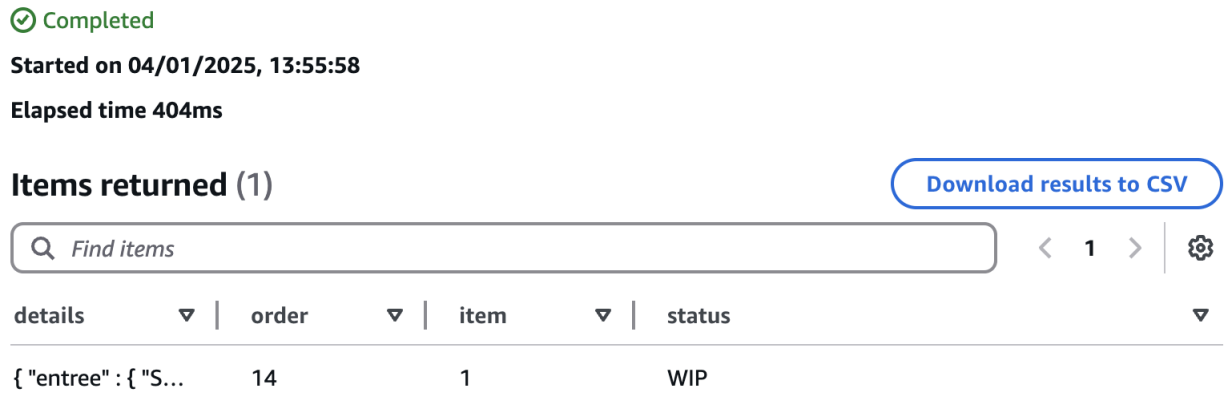
```
select * from "orders";
```



details	order	item	status
{ "entree" : { "S...	14	1	WIP
{ "entree" : { "S...	14	2	WIP
{ "entree" : { "S...	16	1	DONE
{ "entree" : { "S...	15	1	WIP

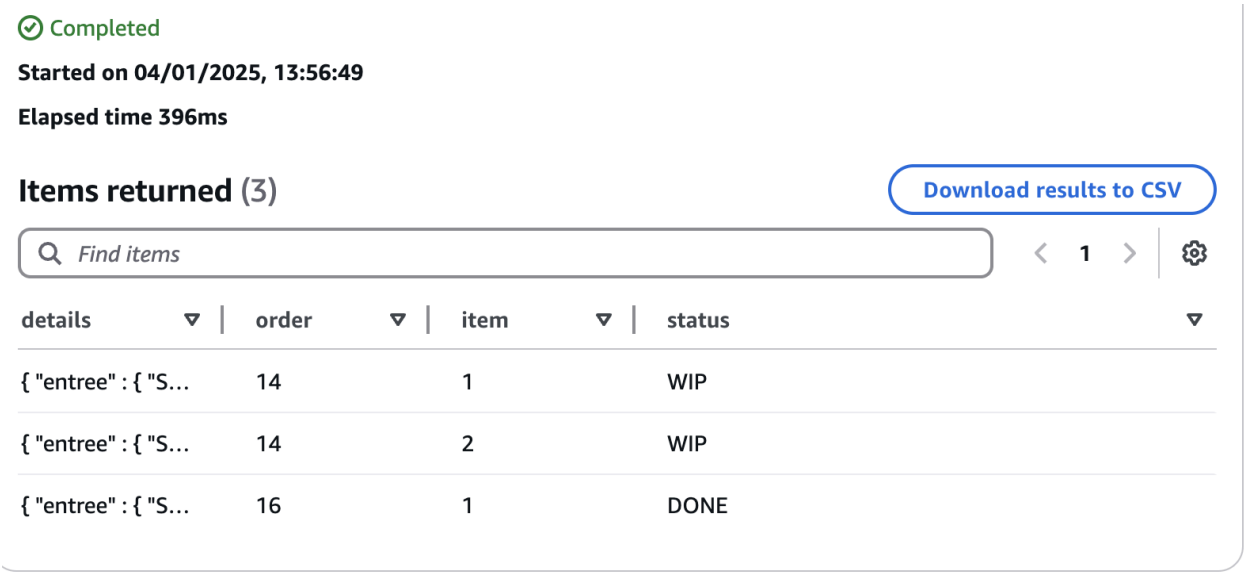
Query 1 result

Query 2: Find the first item in order 14, item 1
select * from "orders" where "order" = 14 and "item" = 1;



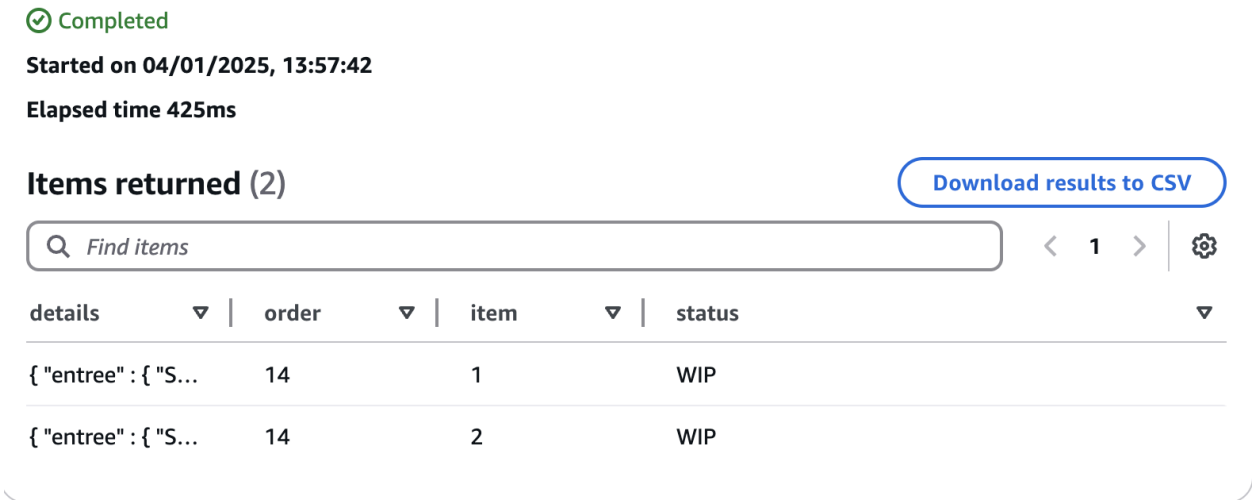
Query 2 result

Query 3: Find all the order lines that include a side of water
SELECT * FROM "orders"
WHERE contains(details.sides, 'water')



Query 3 result

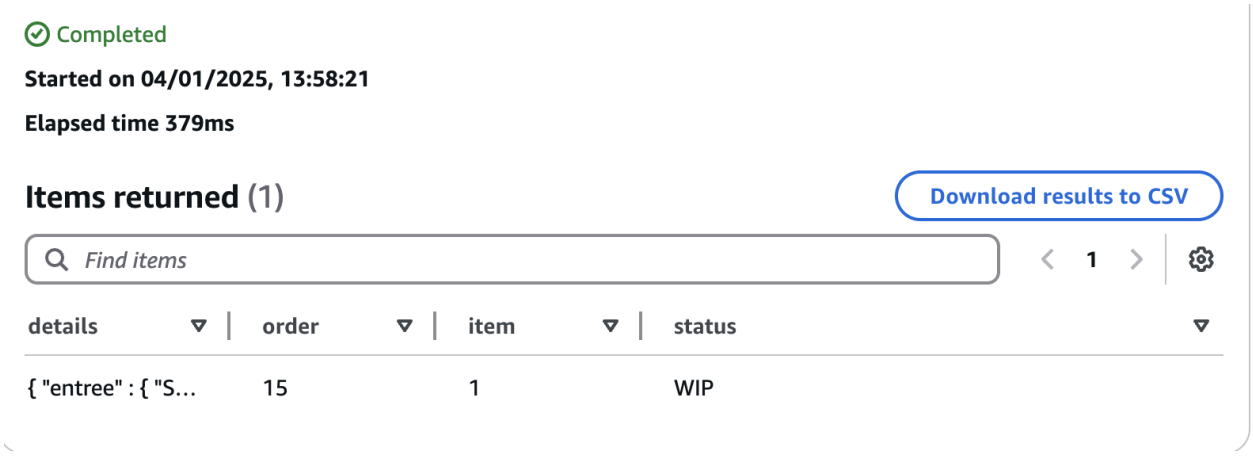
Query 4: Find all the order lines that include a side of water that aren't yet complete:
SELECT * FROM "orders"
WHERE status!='DONE' and
contains(details.sides, 'water');



Query 4 result

Query 5: an order of fries is ready, and we want to know which order should it be sent to:

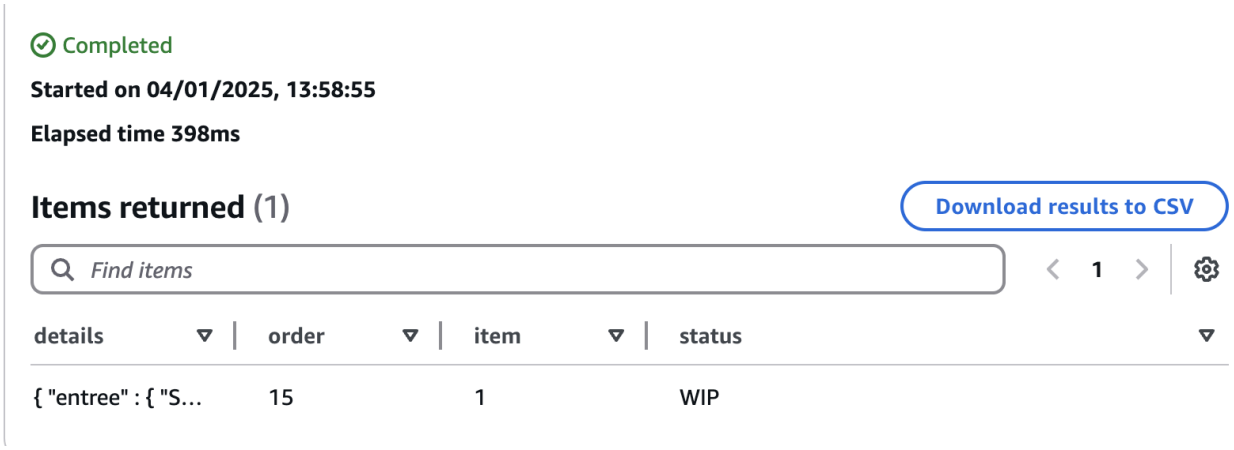
```
select *
from "orders”
where status = 'WIP' and contains(details.sides, 'fries')
```



Query 5 result

Query 6: If a burger is ready, we can determine which order should it be attached to:

```
select *
from "orders"
where details.entree = 'burger' and status = 'WIP';
```

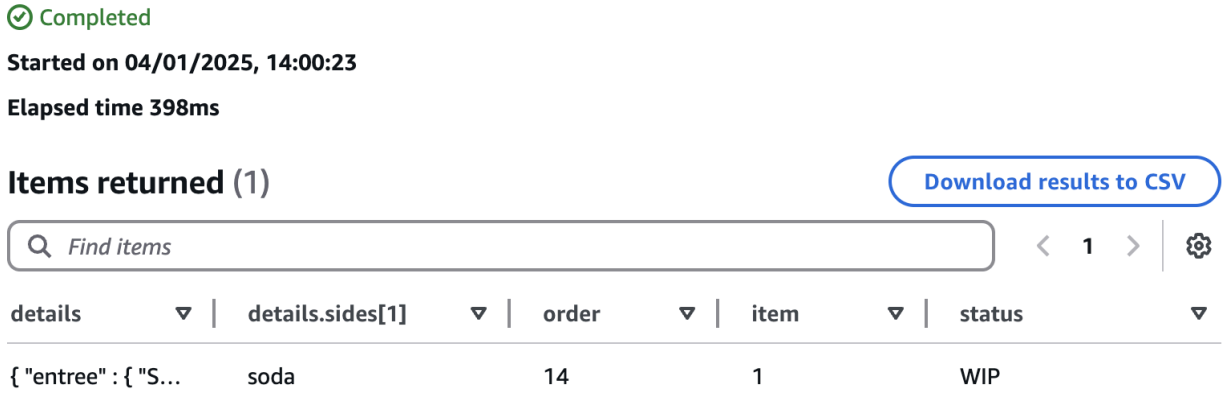


Query 6 result

Update orders

Query 7: The customer who placed order 14, item 1, changed their mind and instead of water would like a soda:

select * from "orders" where "order"=14 and "item"=1
update "orders" set "details.sides[1]"='soda' where "order"=14 and "item"=1
select * from "orders" where "order"=14 and "item"=1



Query 7 result

Add more attributes to an order

Query 8: Add attributes to an embedded object. If a customer has a special request, you can add it to the order:

update "orders" set "details.notes" = 'extra dressing' where "order" = 14 and "item" = 1
select * from "orders" where "order"=14

Completed

Started on 04/01/2025, 14:01:39

Elapsed time 408ms

Items returned (2) Download results to CSV

Find items 1

details.notes	details	details.sides[1]	order	item
extra dressing	{ "entree" : ...	soda	14	1
	{ "entree" : ...		14	2

Query 8 result

Connect to the database using python

We use a library named boto3 and write 3 cells of code, this task was done in a python notebook. And it is available in our github under Deliverable 1.

The first cell creates the DynamoDB instance, we connect it by adding certain keys obtained by writing 'cat ~/.aws/credentials' in our lab session terminal. These keys are as follows: access key, secret access key, session token and region name. With these values we are able to connect the order table using python.

```
import boto3

# Creating a DynamoDB Instance
dynamodb = boto3.resource('dynamodb',
    aws_access_key_id='',
    aws_secret_access_key='',
    aws_session_token='',
    region_name=''
)

table_name = 'orders'
table = dynamodb.Table(table_name)

print("Table connected: ", table.table_status)
```

```
Table connected:  ACTIVE
```

First cell of the notebook along with output

The second cell is used to read all the values of the table, and print the items out so we can get an overview of the items we have and what they contain. We would like to add that these results are after the queries have been executed on the PartiQL.

```
# Reading all the values of the table
response = table.scan()

for item in response['Items']:
    print(item)
```

```
{'details.notes': 'extra dressing', 'details.sides[0]': 'soda', 'details': {'entree': 'salad', 'sides': {'apple', 'water'}}, 'order': Dec
{'details': {'entree': 'BLT sandwich', 'sides': {'water'}}, 'order': Decimal('14'), 'item': Decimal('2'), 'status': 'WIP'}
{'details': {'entree': 'salad', 'sides': {'apple', 'water'}}, 'order': Decimal('16'), 'item': Decimal('1'), 'status': 'DONE'}
{'details': {'entree': 'burger', 'sides': {'fries', 'soda'}}, 'order': Decimal('15'), 'item': Decimal('1'), 'status': 'WIP'}
```

Second cell of the notebook along with output

Our last cell of code is an example of how to make queries in our python program.

```
from boto3.dynamodb.conditions import Key
# Making queryies

response = table.query(
    KeyConditionExpression=Key('order').eq(14)
    )

response.get('Items', [])
```

```
[{'details.notes': 'extra dressing',
  'details.sides[0]': 'soda',
  'details': {'entree': 'salad', 'sides': {'apple', 'water'}},
  'order': Decimal('14'),
  'item': Decimal('1'),
  'status': 'WIP'},
 {'details': {'entree': 'BLT sandwich', 'sides': {'water'}},
  'order': Decimal('14'),
  'item': Decimal('2'),
  'status': 'WIP'}]
```

Third cell of the notebook along with output