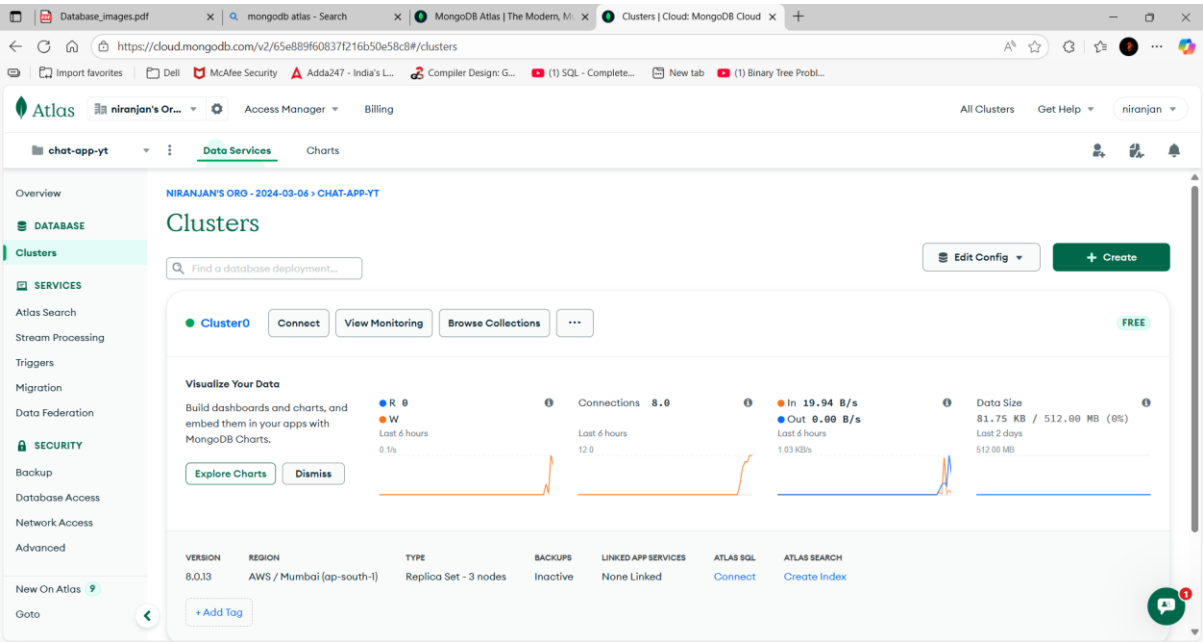


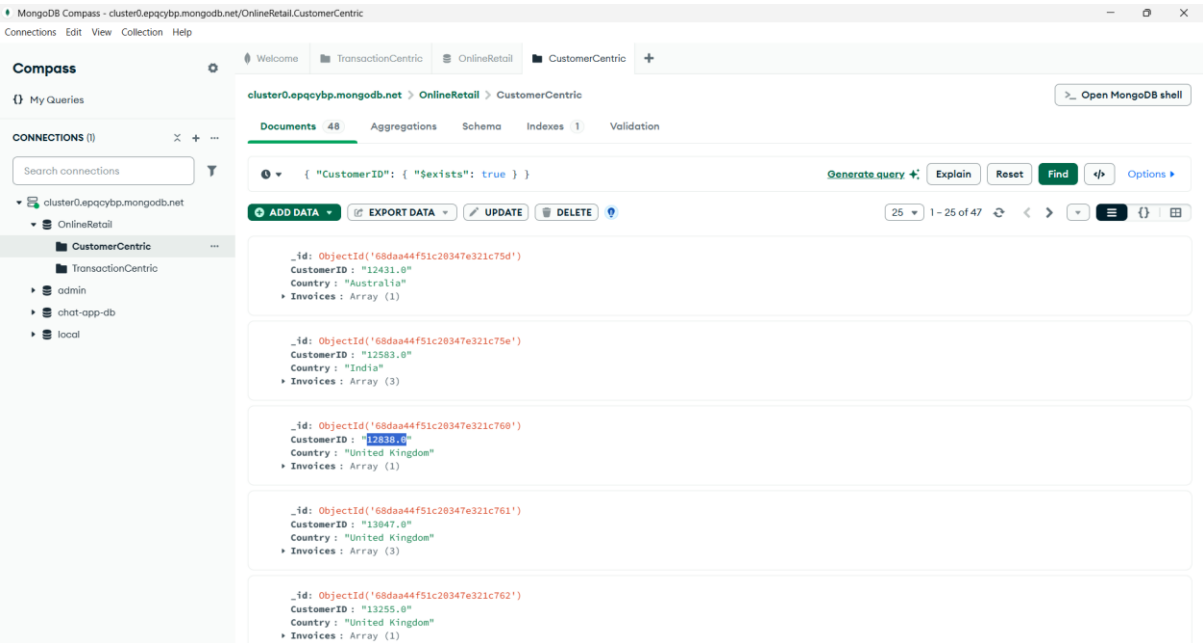
MongoDB Atlas + Compass Connection



MongoDB (OnlineRetail) Database With Collections

CustomerCentric

TransactionCentric



Filter Query in CustomerCentric Component

The screenshot shows the MongoDB Compass interface. The left sidebar displays the database structure with 'CustomerCentric' selected. The main panel shows a filter query: `{ "CustomerID": "12583.0" }`. The results table displays one document:

Document
<pre>{ "_id": ObjectId('68daa44f51c20347e321c75e'), "CustomerID": "12583.0", "Country": "India", "Invoices": Array (3) }</pre>

Filter Query in Transaction Centric:

The screenshot shows the MongoDB Compass interface with 'TransactionCentric' selected. The filter query is: `{ "CustomerID": "17850.0" }`. The results table displays four documents:

Document
<pre>{ "_id": ObjectId('68daa39580b25f67ebac21ff'), "InvoiceNo": 536365, "InvoiceDate": 2010-12-01T08:26:00.000+00:00, "CustomerID": "17850.0", "Country": "United Kingdom", "Items": Array (7) }</pre>
<pre>{ "_id": ObjectId('68daa39580b25f67ebac21fe'), "InvoiceNo": 536366, "InvoiceDate": 2010-12-01T08:28:00.000+00:00, "CustomerID": "17850.0", "Country": "United Kingdom", "Items": Array (2) }</pre>
<pre>{ "_id": ObjectId('68daa39580b25f67ebac21fe'), "InvoiceNo": 536372, "InvoiceDate": 2010-12-01T09:01:00.000+00:00, "CustomerID": "17850.0", "Country": "United Kingdom", "Items": Array (2) }</pre>
<pre>{ "_id": ObjectId('68daa39580b25f67ebac21ff'), "InvoiceNo": 536373, "InvoiceDate": 2010-12-01T09:02:00.000+00:00, "CustomerID": "17850.0", "Country": "United Kingdom", "Items": Array (2) }</pre>

PostgreSQL : Database (online_retail)

Running Queries

The screenshot shows the pgAdmin 4 interface with the 'online_retail/postgres@PostgreSQL 16*' connection selected. The 'Query' tab is active, displaying a SQL query that counts records in 'Customer', 'Product', and 'InvoiceDetails' tables, and then joins 'Invoice' and 'Customer' tables to retrieve invoice details.

```
1 SELECT COUNT(*) FROM "Customer";
2 SELECT COUNT(*) FROM "Product";
3 SELECT COUNT(*) FROM "Invoice";
4 SELECT COUNT(*) FROM "InvoiceDetails";
5
6 SELECT i."InvoiceNo", i."InvoiceDate", c."CustomerID", c."Country"
7 FROM "Invoice" i
8 JOIN "Customer" c ON i."CustomerID" = c."CustomerID"
9 LIMIT 10;
```

The 'Data Output' tab shows the results of the query, displaying a table with 10 rows and 5 columns: InvoiceNo, InvoiceDate, CustomerID, Country, and a count. The status bar indicates 'Total rows: 10 of 10' and 'Query complete 00:00:00.068'.

InvoiceNo	InvoiceDate	CustomerID	Country	
536407	2010-12-01 11:34:00	17850.0	United Kingdom	
536406	2010-12-01 11:33:00	17850.0	United Kingdom	
536399	2010-12-01 10:52:00	17850.0	United Kingdom	
536396	2010-12-01 10:51:00	17850.0	United Kingdom	
536377	2010-12-01 09:34:00	17850.0	United Kingdom	
536375	2010-12-01 09:32:00	17850.0	United Kingdom	
536373	2010-12-01 09:02:00	17850.0	United Kingdom	
536372	2010-12-01 09:01:00	17850.0	United Kingdom	
536366	2010-12-01 08:28:00	17850.0	United Kingdom	
Total rows: 10 of 10				

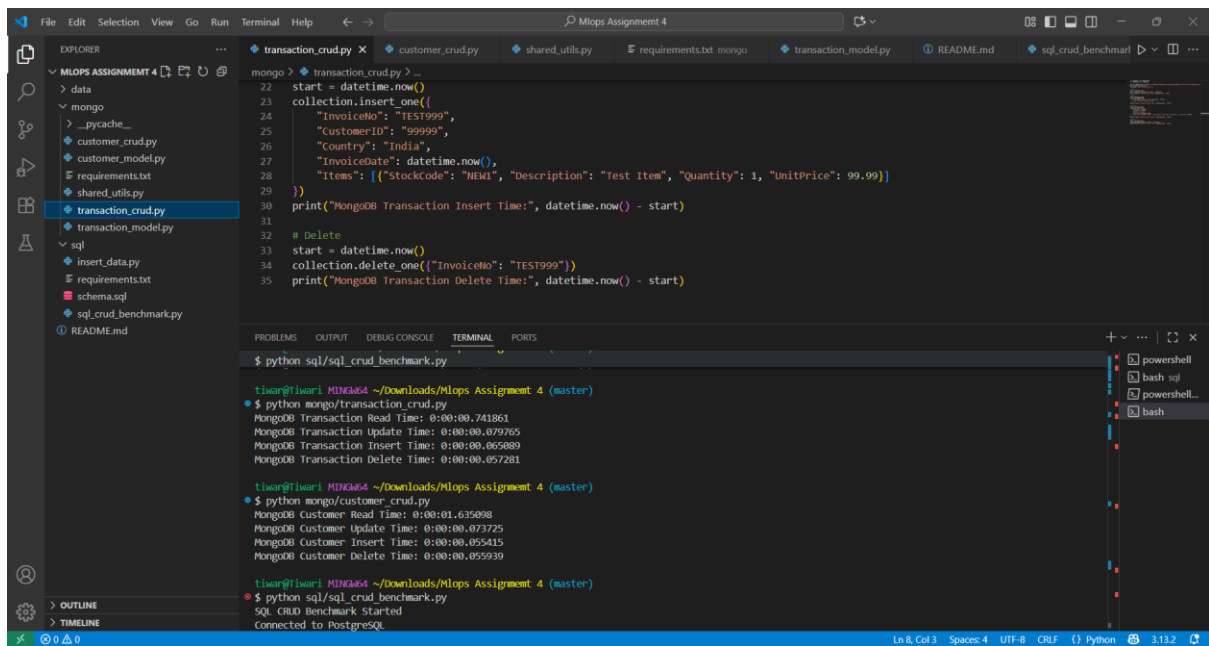
The screenshot shows the pgAdmin 4 interface with the 'online_retail/postgres@PostgreSQL 16*' connection selected. The 'Query' tab is active, displaying a SQL query that counts records in 'Customer', 'Product', and 'InvoiceDetails' tables.

```
1 SELECT COUNT(*) FROM "Customer";
2 SELECT COUNT(*) FROM "Product";
3 SELECT COUNT(*) FROM "Invoice";
4 SELECT COUNT(*) FROM "InvoiceDetails";
```

The 'Data Output' tab shows the results of the query, displaying a table with 1 row and 1 column: count bigint. The status bar indicates 'Total rows: 1 of 1' and 'Query complete 00:00:00.063'.

count bigint
1000

MongoDB: Execution Time (TransactionCentric and CustomerCentric)



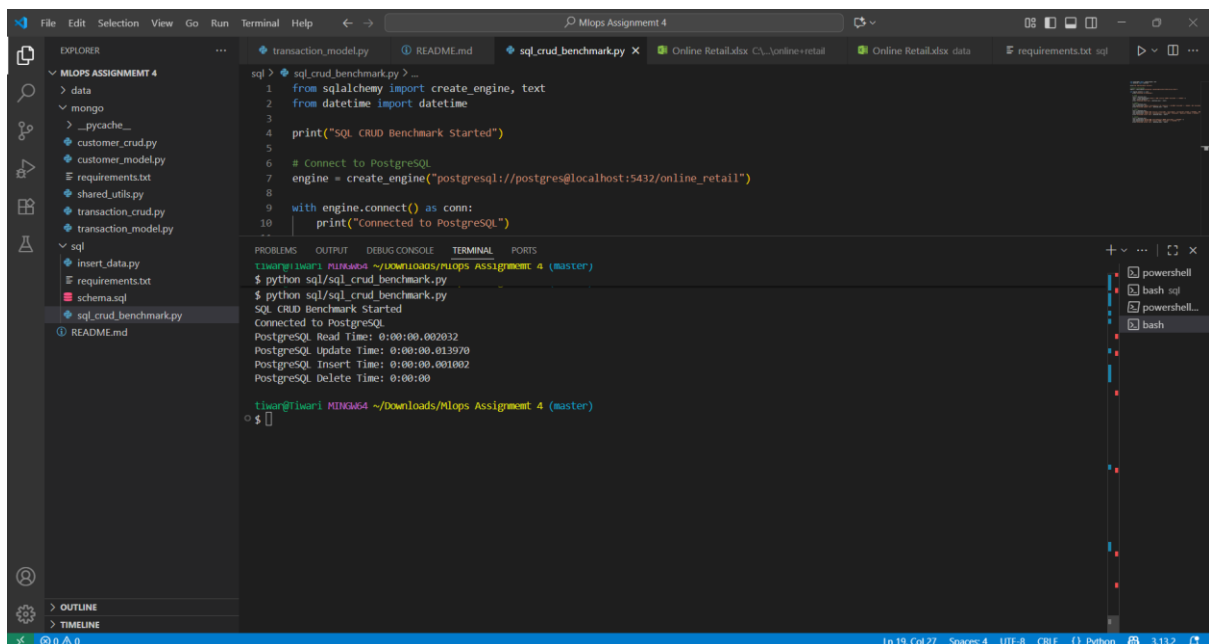
The screenshot shows a VS Code editor with a file explorer on the left containing files like `transaction_crud.py`, `customer_crud.py`, and `sql_crud_benchmark.py`. The main editor displays the code for `transaction_crud.py`, which includes functions for inserting, updating, and deleting a transaction record in MongoDB. The terminal at the bottom shows the execution of these functions, with output indicating the execution time for each operation.

```
mongo> python mongo/transaction_crud.py
MongoDB Transaction Read Time: 0:00:00.741861
MongoDB Transaction Update Time: 0:00:00.079765
MongoDB Transaction Insert Time: 0:00:00.065080
MongoDB Transaction Delete Time: 0:00:00.057281

mongo> python mongo/customer_crud.py
MongoDB Customer Read Time: 0:00:01.635098
MongoDB Customer Update Time: 0:00:00.072725
MongoDB Customer Insert Time: 0:00:00.055415
MongoDB Customer Delete Time: 0:00:00.055939

mongo> python sql/sql_crud_benchmark.py
SQL CRUD Benchmark Started
Connected to PostgreSQL
```

PostgreSQL: Execution Time of CRUD Operations



The screenshot shows a VS Code editor with a file explorer on the left containing files like `sql_crud_benchmark.py` and `requirements.txt`. The main editor displays the code for `sql_crud_benchmark.py`, which includes functions for connecting to PostgreSQL and performing CRUD operations. The terminal at the bottom shows the execution of these functions, with output indicating the execution time for each operation.

```
sql> python sql/sql_crud_benchmark.py
SQL CRUD Benchmark Started
Connected to PostgreSQL
PostgreSQL Read Time: 0:00:00.002032
PostgreSQL Update Time: 0:00:00.013970
PostgreSQL Insert Time: 0:00:00.001002
PostgreSQL Delete Time: 0:00:00
```

Summary Table of Execution time

Operati on	MongoDB (Transaction- Centric)	MongoDB (Customer- Centric)	PostgreSQL
Read	"0:00:00.741861"	"0:00:01.635098"	"0:00:00.002032"
Update	"0:00:00.079765"	"0:00:00.073725"	"0:00:00.013970"
Insert	"0:00:00.065089"	"0:00:00.055415"	"0:00:00.001002"
Delete	"0:00:00.057281"	"0:00:00.055939"	"0:00:00.000000"

Analysis of CRUD Performance

The benchmark results reveal clear differences in execution time across MongoDB and PostgreSQL for each CRUD operation:

Read Operation

- **PostgreSQL** is significantly faster (~0.002s) due to indexed relational access and flat schema.
- **MongoDB Transaction-Centric** takes longer (~0.74s) but is still efficient for flat documents.
- **MongoDB Customer-Centric** is slowest (~1.63s) due to nested document traversal and larger payloads.

Update Operation

- All models perform well, with **MongoDB** slightly faster (~0.07s) due to direct document targeting.
- **PostgreSQL** update (~0.013s) is fast but involves table-level locking and constraint checks.

Insert Operation

- **PostgreSQL** is fastest (~0.001s) due to optimized bulk insert paths.
- **MongoDB** performs well (~0.05–0.06s) but includes overhead for document validation and indexing.

Delete Operation

- **PostgreSQL** deletion is near-instant (~ 0.000 s) for indexed rows.
- **MongoDB** deletion (~ 0.05 s) is consistent across models.

Insight: SQL's transactional engine handles deletes efficiently; MongoDB's performance is stable but slightly slower.