

PostgreSQL

Customer Orders Database on AWS RDS

This project involves creating a cloud-hosted customer order database on AWS RDS with PostgreSQL. The database stores customer and order information, with key queries to analyze data. It showcases skills in database design, SQL query execution, and cloud deployment, and is documented on GitHub and Notion for documentation and publication.



Hi, I'm Chris, and thank you for checking out my work! I'm hopeful this project exemplifies my knowledge of SQL and my passion for leveraging this language to optimize both professional and personal endeavors.

"Let's take a look at how I built this out. This was my first time creating my own database, and it turned out to be an incredibly fun and rewarding hands-on learning experience. I enjoyed the process of designing, setting up, and working with SQL in the cloud, and it gave me valuable insights into database management and query optimization."

AWS RDS Configuration

Initializing Aurora PostgreSQL Engine

For this project, I set up an AWS RDS instance using the PostgreSQL engine, selecting the most practical configuration for short-term use. I configured the necessary settings, including but not limited to the database name, username, and password, to establish a cloud-hosted environment for the customer order database.

Engine: Aurora PostgreSQL (v 16.6)

Template: DEV/Test

Cluster Configuration: Aurora Standard (Cost Effective; No Output)

Instance Configuration: db.r7g.large

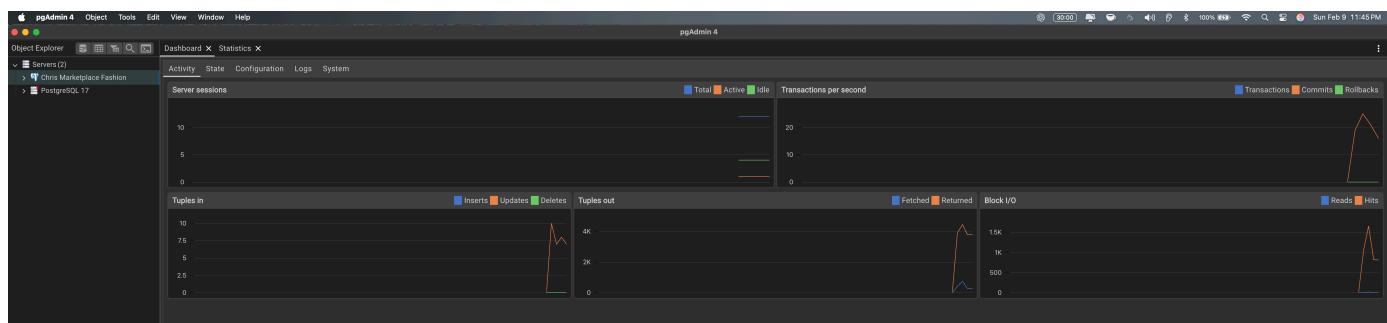
Public Access: Yes (Allowing me to connect to RDS Instance via pgAdmin)

VPC (firewall): Allows inbound traffic on (Port **5432**)

Caption

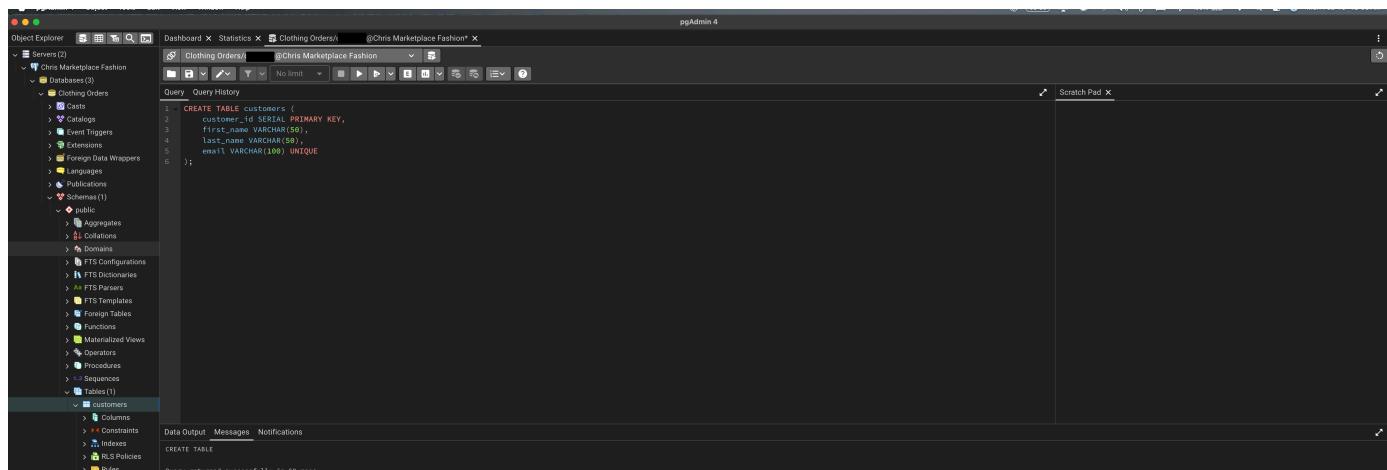
Connecting RDS Instance via pgAdmin

During the setup of my AWS RDS PostgreSQL instance, I faced a “connection timeout expired” error in pgAdmin. After confirming that the instance and public access were set up correctly, I realized my IP wasn’t added to the security group’s inbound rules. To troubleshoot, I used Hyper (*terminal application*) to install Homebrew and Telnet. By running Telnet to test the connection to the RDS endpoint, I identified the issue. Once I added my public IP (verified via curl ***ifconfig.me***) to the security group, I successfully connected to the database through pgAdmin and proceeded with my work.

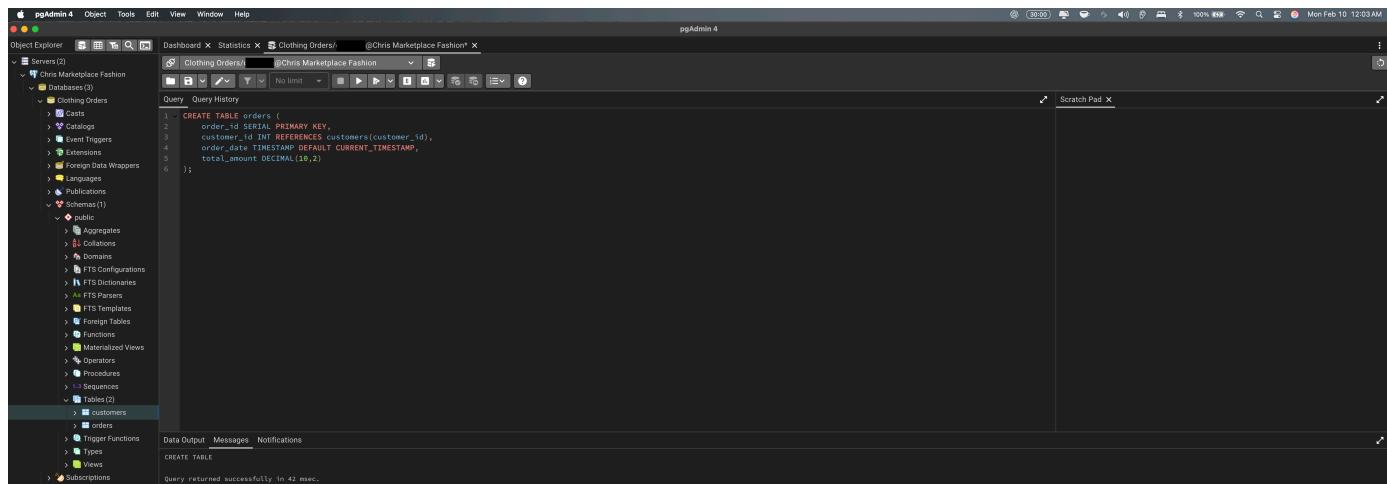


Configuring Database Schema

Once connected, I created the database schema with two tables: customers and orders. The customers table tracks customer information, while the orders table links customer orders to their respective details. After setting up the schema, I inserted sample data into both tables, representing two customers and their corresponding orders. Finally, I used an INNER JOIN query to retrieve and display related data from both tables, ensuring everything was correctly linked and functional.



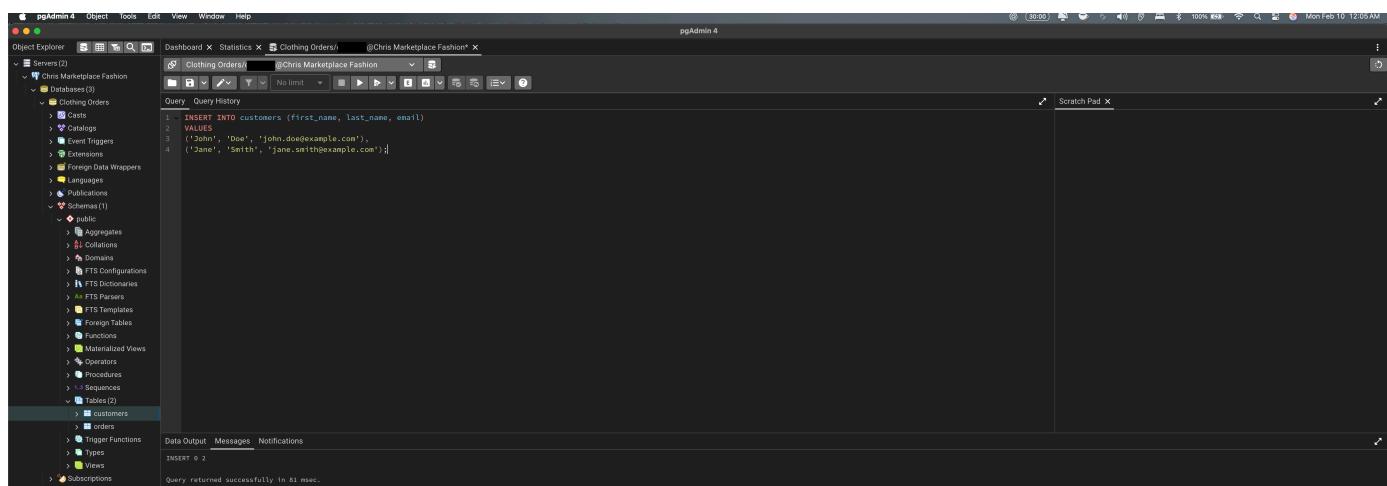
Customer Orders Database on AWS RDS (PostgreSQL)



The screenshot shows the pgAdmin 4 interface for a PostgreSQL database named 'Clothing Orders'. In the Object Explorer, under the 'Tables (2)' section of the 'Clothing Orders' schema, there is a new table named 'orders'. The 'Query' tab in the main pane displays the SQL code for creating the 'orders' table:

```
1. CREATE TABLE orders (
2.     order_id SERIAL PRIMARY KEY,
3.     customer_id INT REFERENCES customers(customer_id),
4.     order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
5.     total_amount DECIMAL(10,2)
6. );
```

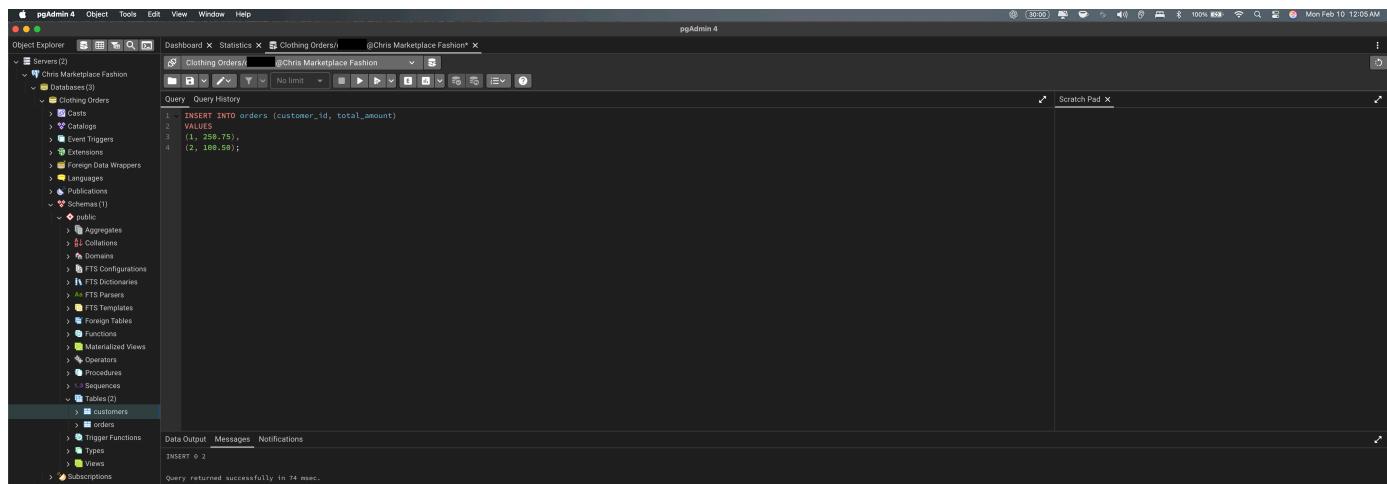
The status bar at the bottom indicates 'Query returned successfully in 42 msec.'



The screenshot shows the pgAdmin 4 interface for the same PostgreSQL database. In the Object Explorer, under the 'Tables (2)' section of the 'Clothing Orders' schema, there is a table named 'customers'. The 'Query' tab in the main pane displays the SQL code for inserting data into the 'customers' table:

```
1. INSERT INTO customers (first_name, last_name, email)
2. VALUES
3. ('John', 'Doe', 'john.doe@example.com'),
4. ('Jane', 'Smith', 'jane.smith@example.com');
```

The status bar at the bottom indicates 'Query returned successfully in 81 msec.'



The screenshot shows the pgAdmin 4 interface for the same PostgreSQL database. In the Object Explorer, under the 'Tables (2)' section of the 'Clothing Orders' schema, there is a table named 'orders'. The 'Query' tab in the main pane displays the SQL code for inserting data into the 'orders' table:

```
1. INSERT INTO orders (customer_id, total_amount)
2. VALUES
3. (1, 258.75),
4. (2, 108.50);
```

The status bar at the bottom indicates 'Query returned successfully in 74 msec.'

Customer Orders Database on AWS RDS (PostgreSQL)

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (2)', there is one entry for 'Chris Marketplace Fashion'. Under 'Chris Marketplace Fashion', there are several categories: 'Database (0)', 'Tables (2)', 'Views (0)', 'Triggers (0)', 'Extensions (0)', 'Foreign Data Wrappers (0)', 'Languages (0)', 'Publications (0)', 'Schemas (1)', and 'Tables (2)'. The 'Tables (2)' section contains two entries: 'customers' and 'orders'. The 'customers' table is selected, and its schema is displayed in the main pane. A query history tab is open, showing a single query:

```

1 SELECT c.customer_id, c.first_name, c.last_name, c.email, o.order_id, o.order_date, o.total_amount
2 FROM customers c
3 INNER JOIN orders o ON c.customer_id = o.customer_id;

```

The main pane also displays a table of customer data:

customer_id	first_name	last_name	email	order_id	order_date	total_amount
1	John	Doe	john.doe@example.com	1	2025-02-10 05:05:45.896537	250.75
2	Jane	Smith	jane.smith@example.com	2	2025-02-10 05:05:45.896537	100.50

The screenshot shows the AWS RDS console for the 'customer-order-db' database. On the left, there's a sidebar with options like 'Dashboard', 'Databases', 'Query Editor', 'Performance insights', 'Snapshots', 'Exports in Amazon S3', 'Advanced backup', 'Reserved instances', 'Proxies', 'Subnet groups', 'Parameter groups', 'Option groups', 'Custom engine versions', 'Zero-ETL Integrations', 'Events', 'Event subscriptions', 'Recommendations', and 'Certificate update'. The main area is titled 'customer-order-db-instance-1'.

The 'Logs & events' tab is active. It shows a table of recent events:

Time	System notes
February 09, 2025, 22:56 UTC-05:00	DB instance restarted
February 09, 2025, 22:56 UTC-05:00	The parameter max_wal_senders was set to a value incompatible with replication. It has been adjusted from 10 to 20.
February 09, 2025, 22:56 UTC-05:00	DB instance shutdown
February 09, 2025, 22:56 UTC-05:00	DB instance restarted
February 09, 2025, 22:41 UTC-05:00	The parameter max_wal_senders was set to a value incompatible with replication. It has been adjusted from 10 to 20.
February 09, 2025, 22:41 UTC-05:00	DB instance shutdown
February 09, 2025, 21:47 UTC-05:00	Performance metrics have been enabled
February 09, 2025, 21:47 UTC-05:00	Monitoring interval changed to 60
February 09, 2025, 21:47 UTC-05:00	DB instance created
February 09, 2025, 21:40 UTC-05:00	The parameter max_wal_senders was set to a value incompatible with replication. It has been adjusted from 10 to 20.
February 09, 2025, 21:38 UTC-05:00	Cluster topology is updated

Below the event log is a 'Logs' section with a table of log files:

Name	Last written	Size
February 09, 2025, 22:56 UTC-05:00	8.1 kB	
February 09, 2025, 21:40 UTC-05:00	10.1 kB	
February 09, 2025, 22:09 UTC-05:00	3 kB	
February 09, 2025, 22:40 UTC-05:00	8 kB	
February 09, 2025, 22:58 UTC-05:00	1.3 kB	
February 09, 2025, 23:58 UTC-05:00	7.2 kB	
February 10, 2025, 00:01 UTC-05:00	645 B	
February 09, 2025, 21:39 UTC-05:00	0 bytes	

Caption

Harnessing the Power of PostgreSQL in AWS

This project has not only solidified my foundational understanding of cloud databases and AWS, but it has also sharpened my problem-solving and technical troubleshooting abilities, which are critical in support and operations roles. The experience gained will be invaluable as I pursue more complex challenges, both in AWS cloud architecture and technical support. I am eager to apply these skills in real-world environments, where I can optimize cloud infrastructures and provide comprehensive support solutions to organizations. By advancing my knowledge further with more intricate database projects, testing new features, and exploring integrations with various AWS services, I am confident this expertise will significantly enhance my ability to contribute to the ongoing success of any technical team. This journey is just beginning, and I look forward to refining these skills and embracing new opportunities within the world of AWS and technical support.