**Truy vấn Amazon Aurora PostgreSQL bằng Amazon Bedrock Knowledge Bases với dữ liệu có cấu trúc**

**Amazon Bedrock Knowledge Bases** cung cấp một tính năng **RAG (Retrieval Augmented Generation)** được quản lý hoàn toàn, cho phép **kết nối các mô hình ngôn ngữ lớn (LLM)** với **nguồn dữ liệu nội bộ**. Tính năng này giúp cải thiện đầu ra của các mô hình nền tảng (Foundation Model) bằng cách bổ sung thông tin có ngữ cảnh từ dữ liệu riêng tư của người dùng, từ đó làm cho phản hồi trở nên **chính xác và phù hợp hơn**.

Tại sự kiện **AWS re:Invent 2024**, AWS đã công bố rằng **Amazon Bedrock Knowledge Bases hỗ trợ truy vấn ngôn ngữ tự nhiên** để truy xuất dữ liệu có cấu trúc từ **Amazon Redshift** và **Amazon SageMaker Lakehouse**. Tính năng này cung cấp **một quy trình làm việc có quản lý** để xây dựng các ứng dụng AI sinh ngữ, có thể truy cập và sử dụng dữ liệu từ cả nguồn có cấu trúc lẫn phi cấu trúc.

Thông qua **xử lý ngôn ngữ tự nhiên (NLP)**, Amazon Bedrock Knowledge Bases sẽ **chuyển đổi các câu hỏi ngôn ngữ tự nhiên thành truy vấn SQL**, giúp người dùng truy xuất dữ liệu **mà không cần hiểu cấu trúc cơ sở dữ liệu hoặc cú pháp SQL**.

**Mục tiêu của bài viết**

Trong bài viết này, chúng tôi sẽ hướng dẫn cách **kết nối dữ liệu từ Amazon Aurora PostgreSQL-Compatible Edition** để truy vấn bằng ngôn ngữ tự nhiên thông qua Amazon Bedrock Knowledge Bases **và vẫn đảm bảo dữ liệu luôn cập nhật**.

**Truy xuất dữ liệu có cấu trúc với Amazon Bedrock Knowledge Bases và Amazon Redshift Zero-ETL**

Tính năng **structured data retrieval** trong Amazon Bedrock Knowledge Bases cho phép **tương tác cơ sở dữ liệu bằng ngôn ngữ tự nhiên** bằng cách **chuyển đổi câu hỏi của người dùng thành các truy vấn SQL**.

Khi bạn kết nối một nguồn dữ liệu được hỗ trợ (ví dụ: Amazon Redshift), Bedrock Knowledge Bases sẽ:

* Phân tích **cấu trúc cơ sở dữ liệu**
* Hiểu được **mối quan hệ giữa các bảng**
* Tận dụng **query engine và lịch sử truy vấn**
* Từ đó tạo ra các truy vấn SQL **chính xác, phù hợp ngữ cảnh**

**Tại thời điểm hiện tại**, Bedrock Knowledge Bases **chỉ hỗ trợ trực tiếp Amazon Redshift và SageMaker Lakehouse** cho truy vấn dữ liệu có cấu trúc. Tuy nhiên, bạn **vẫn có thể truy cập dữ liệu từ Aurora PostgreSQL-Compatible** bằng cách sử dụng **tích hợp zero-ETL giữa Aurora và Redshift**.

Tích hợp này giúp **sao chép dữ liệu từ Aurora sang Redshift gần như theo thời gian thực**, mà **không cần phải xây dựng pipeline ETL phức tạp**.

**Ý nghĩa kiến trúc**

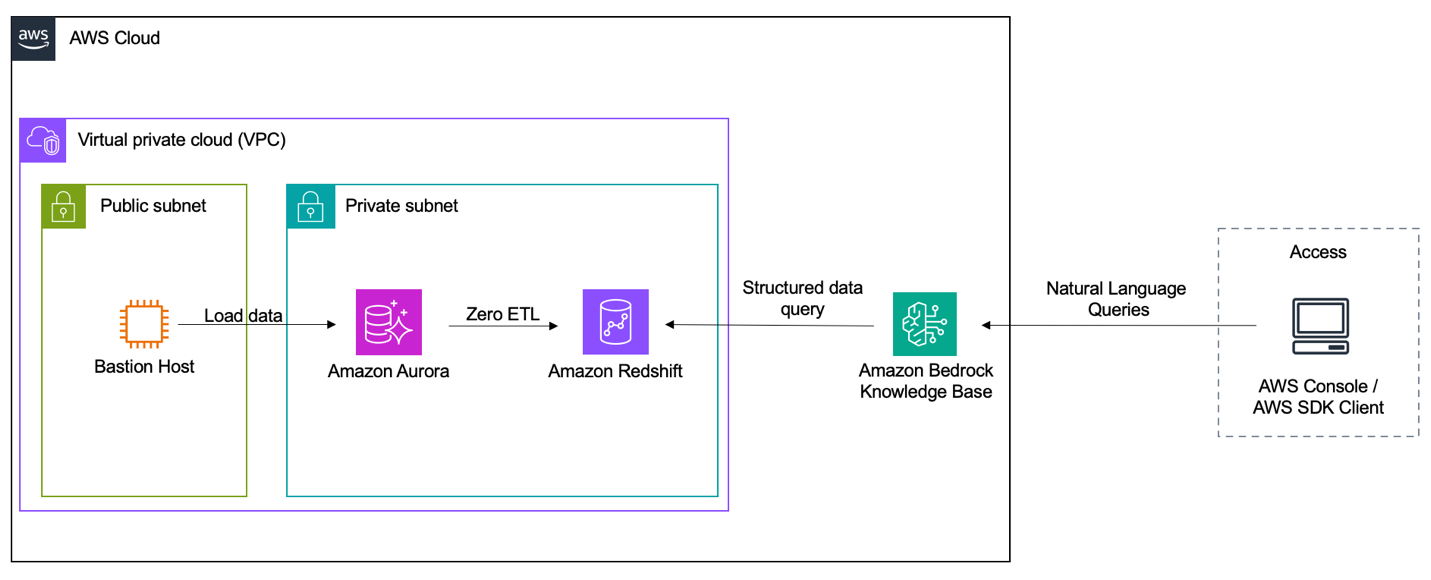
Mô hình kiến trúc này đặc biệt hữu ích đối với các tổ chức muốn **kích hoạt khả năng truy vấn ngôn ngữ tự nhiên trên dữ liệu ứng dụng được lưu trữ trong Amazon Aurora**.

Bằng cách **kết hợp tích hợp zero-ETL với Bedrock Knowledge Bases**, bạn có thể xây dựng các ứng dụng thông minh như **trợ lý AI** – có khả năng cung cấp phản hồi bằng ngôn ngữ tự nhiên dựa trên dữ liệu vận hành trong thời gian thực.

**Tổng quan giải pháp**

Sơ đồ sau minh họa **kiến trúc kết nối Aurora PostgreSQL-Compatible với Amazon Bedrock Knowledge Bases** thông qua **zero-ETL**.

*(Lưu ý: Nếu bạn muốn, mình có thể mô phỏng sơ đồ hoặc giải thích chi tiết các bước triển khai.)*



**Quy trình làm việc gồm các bước sau:**

1. **Dữ liệu được lưu trữ trong Aurora PostgreSQL-Compatible**, nằm trong **private subnet** (mạng con riêng tư).  
   Để truy cập an toàn từ **public subnet**, ta sử dụng một **bastion host** (máy trung gian bảo mật) làm cầu nối kết nối đến cơ sở dữ liệu.
2. **Thông qua tích hợp zero-ETL**, dữ liệu từ Aurora được **đồng bộ hóa gần như theo thời gian thực** sang **Amazon Redshift**, cũng nằm trong private subnet.
3. **Amazon Bedrock Knowledge Bases** sử dụng **Amazon Redshift làm nguồn dữ liệu có cấu trúc** để xử lý và phản hồi các truy vấn.
4. Người dùng có thể tương tác với Amazon Bedrock Knowledge Bases thông qua:
   * **AWS Management Console**
   * Hoặc một **client sử dụng AWS SDK**  
     Người dùng gửi các **truy vấn ngôn ngữ tự nhiên**, sau đó được Bedrock Knowledge Bases **chuyển đổi thành truy vấn SQL** để lấy dữ liệu từ Amazon Redshift — mà bản chất dữ liệu này chính là **được sao chép từ Aurora PostgreSQL**.

**Điều kiện tiên quyết**

Trước khi bắt đầu, bạn cần đảm bảo đang đăng nhập bằng **một IAM role (vai trò người dùng)** có đủ quyền để thực hiện các tác vụ sau:

* **Tạo cơ sở dữ liệu Aurora**
* Chạy các câu lệnh **DDL** (CREATE, ALTER, DROP, RENAME)
* Chạy các câu lệnh **DML** (SELECT, INSERT, UPDATE, DELETE)
* **Tạo cơ sở dữ liệu Redshift**
* **Thiết lập tích hợp zero-ETL giữa Aurora và Redshift**
* **Tạo một knowledge base trong Amazon Bedrock**

**Thiết lập cơ sở dữ liệu Aurora PostgreSQL**

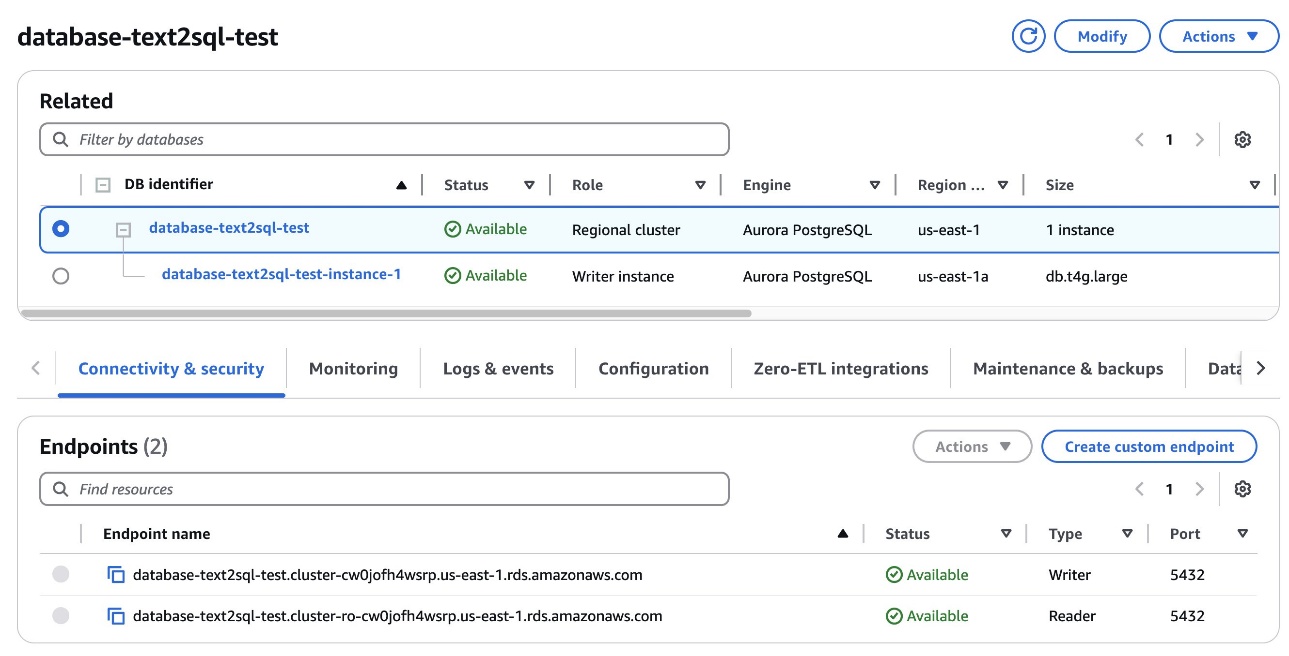
Trong phần này, chúng tôi sẽ hướng dẫn cách tạo và cấu hình một cơ sở dữ liệu Aurora PostgreSQL với một lược đồ (schema) mẫu để minh họa. Chúng tôi sẽ tạo ba bảng có liên kết với nhau: products, customers và orders.

**Khởi tạo cơ sở dữ liệu**

Hãy bắt đầu bằng cách thiết lập môi trường cơ sở dữ liệu.  
Tạo một cụm cơ sở dữ liệu Aurora PostgreSQL mới và khởi chạy một phiên bản Amazon Elastic Compute Cloud (Amazon EC2), phiên bản này sẽ đóng vai trò là điểm truy cập để quản lý cơ sở dữ liệu.

EC2 instance này sẽ giúp bạn dễ dàng tạo bảng và quản lý dữ liệu trong suốt bài viết này.

Hình ảnh dưới đây hiển thị chi tiết cụm cơ sở dữ liệu và phiên bản EC2 của chúng tôi.



Để biết hướng dẫn thiết lập cơ sở dữ liệu, hãy tham khảo [**Creating and connecting to an Aurora PostgreSQL DB cluster**](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Connecting.html).

**Tạo schema cho cơ sở dữ liệu**

Sau khi bạn kết nối với cơ sở dữ liệu bằng SSH thông qua phiên bản EC2 (được mô tả trong phần *Creating and connecting to an Aurora PostgreSQL DB cluster*), bước tiếp theo là tạo cấu trúc dữ liệu.

Chúng tôi sử dụng các câu lệnh DDL sau để tạo ba bảng:

-- Create Product table

CREATE TABLE product (

product\_id SERIAL PRIMARY KEY,

product\_name VARCHAR(100) NOT NULL,

price DECIMAL(10, 2) NOT NULL

);

-- Create Customer table

CREATE TABLE customer (

customer\_id SERIAL PRIMARY KEY,

customer\_name VARCHAR(100) NOT NULL,

pincode VARCHAR(10) NOT NULL

);

-- Create Orders table

CREATE TABLE orders (

order\_id SERIAL PRIMARY KEY,

product\_id INTEGER NOT NULL,

customer\_id INTEGER NOT NULL,

FOREIGN KEY (product\_id) REFERENCES product(product\_id),

FOREIGN KEY (customer\_id) REFERENCES customer(customer\_id)

);

**Chèn dữ liệu vào các bảng**

Sau khi bạn đã tạo xong các bảng, bước tiếp theo là **chèn dữ liệu mẫu** vào chúng.  
Khi chèn dữ liệu vào bảng orders, hãy nhớ **duy trì tính toàn vẹn tham chiếu** bằng cách đảm bảo những điều sau:

* product\_id phải tồn tại trong bảng products
* customer\_id phải tồn tại trong bảng customers

Chúng tôi sử dụng đoạn mã ví dụ sau để chèn dữ liệu vào các bảng:

INSERT INTO product (product\_id, product\_name, price) VALUES (1, 'Smartphone X', 699.99);

INSERT INTO product (product\_id, product\_name, price) VALUES (2, 'Laptop Pro', 1299.99);

INSERT INTO product (product\_id, product\_name, price) VALUES (3, 'Wireless Earbuds', 129.99);

INSERT INTO customer (customer\_id, customer\_name, pincode) VALUES (1, 'John Doe', '12345');

INSERT INTO customer (customer\_id, customer\_name, pincode) VALUES (2, 'Jane Smith', '23456');

INSERT INTO customer (customer\_id, customer\_name, pincode) VALUES (3, 'Robert Johnson', '34567');

INSERT INTO orders (order\_id, product\_id, customer\_id) VALUES (1, 1, 1);

INSERT INTO orders (order\_id, product\_id, customer\_id) VALUES (2, 1, 2);

INSERT INTO orders (order\_id, product\_id, customer\_id) VALUES (3, 2, 3);

INSERT INTO orders (order\_id, product\_id, customer\_id) VALUES (4, 2, 1);

INSERT INTO orders (order\_id, product\_id, customer\_id) VALUES (5, 3, 2);

INSERT INTO orders (order\_id, product\_id, customer\_id) VALUES (6, 3, 3);

**Đảm bảo toàn vẹn tham chiếu**

Khi bạn chèn dữ liệu vào bảng orders, hãy đảm bảo **duy trì tính toàn vẹn tham chiếu** để tránh vi phạm **ràng buộc khóa ngoại (foreign key constraint violations)**.

Bạn cũng có thể sử dụng các ví dụ tương tự để xây dựng schema của riêng mình và thêm dữ liệu cho phù hợp.

**Thiết lập Redshift cluster và cấu hình zero-ETL**

Sau khi đã thiết lập cơ sở dữ liệu Aurora PostgreSQL, bạn có thể tiến hành **thiết lập tích hợp zero-ETL với Amazon Redshift**.  
Tích hợp này sẽ **tự động đồng bộ dữ liệu giữa Aurora PostgreSQL-Compatible và Amazon Redshift**.

**Thiết lập Amazon Redshift**

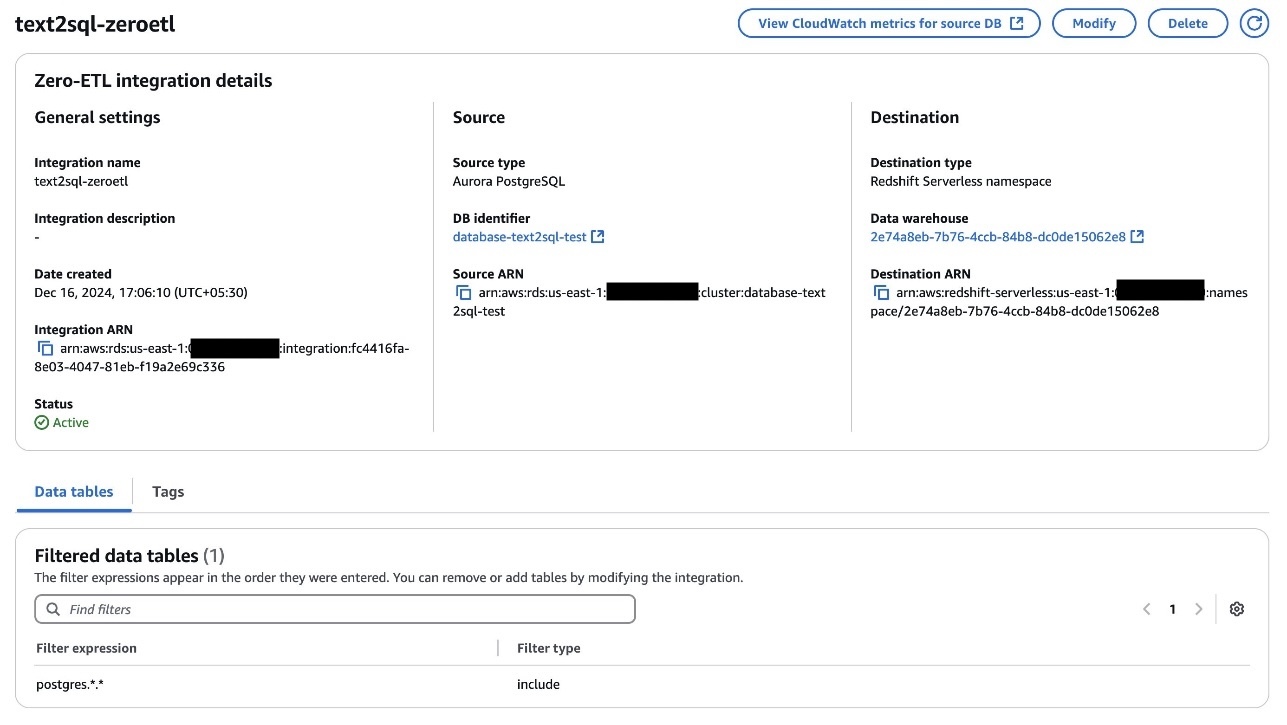
Trước tiên, hãy tạo một **Redshift Serverless workgroup** và **namespace**.  
Để biết hướng dẫn chi tiết, hãy tham khảo [**Creating a data warehouse with Amazon Redshift Serverless**](https://docs.aws.amazon.com/redshift/latest/gsg/serverless-create-workgroup.html).

**Tạo tích hợp zero-ETL**

Quá trình tích hợp zero-ETL gồm hai bước chính:

1. **Tạo tích hợp zero-ETL** từ cơ sở dữ liệu Aurora PostgreSQL của bạn đến Redshift Serverless.
2. Sau khi thiết lập tích hợp từ phía Aurora, **tạo cơ sở dữ liệu ánh xạ tương ứng (mapping database)** trong Amazon Redshift.  
   Đây là bước quan trọng để đảm bảo dữ liệu được đồng bộ hóa chính xác giữa hai dịch vụ.

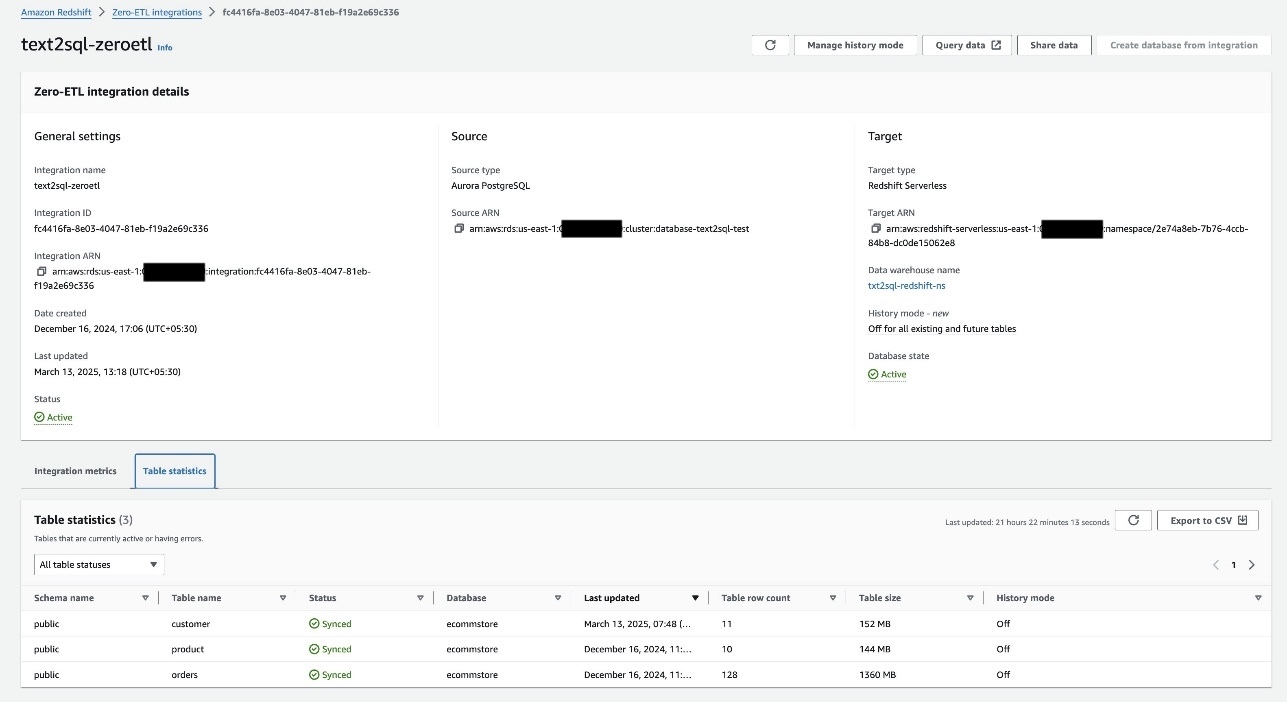
Hình ảnh dưới đây hiển thị thông tin chi tiết về cấu hình zero-ETL của chúng tôi.



**Xác minh tích hợp**

Sau khi bạn hoàn tất quá trình tích hợp, có thể xác minh tính thành công của nó thông qua một số cách kiểm tra sau:

Trước hết, bạn có thể kiểm tra **thông tin chi tiết về tích hợp zero-ETL** trong **bảng điều khiển Amazon Redshift**.  
Tại đây, bạn nên thấy **trạng thái là “Active”** cho tích hợp của mình, cùng với các thông tin về **nguồn (source)** và **đích (destination)**, như được hiển thị trong ảnh chụp màn hình sau.

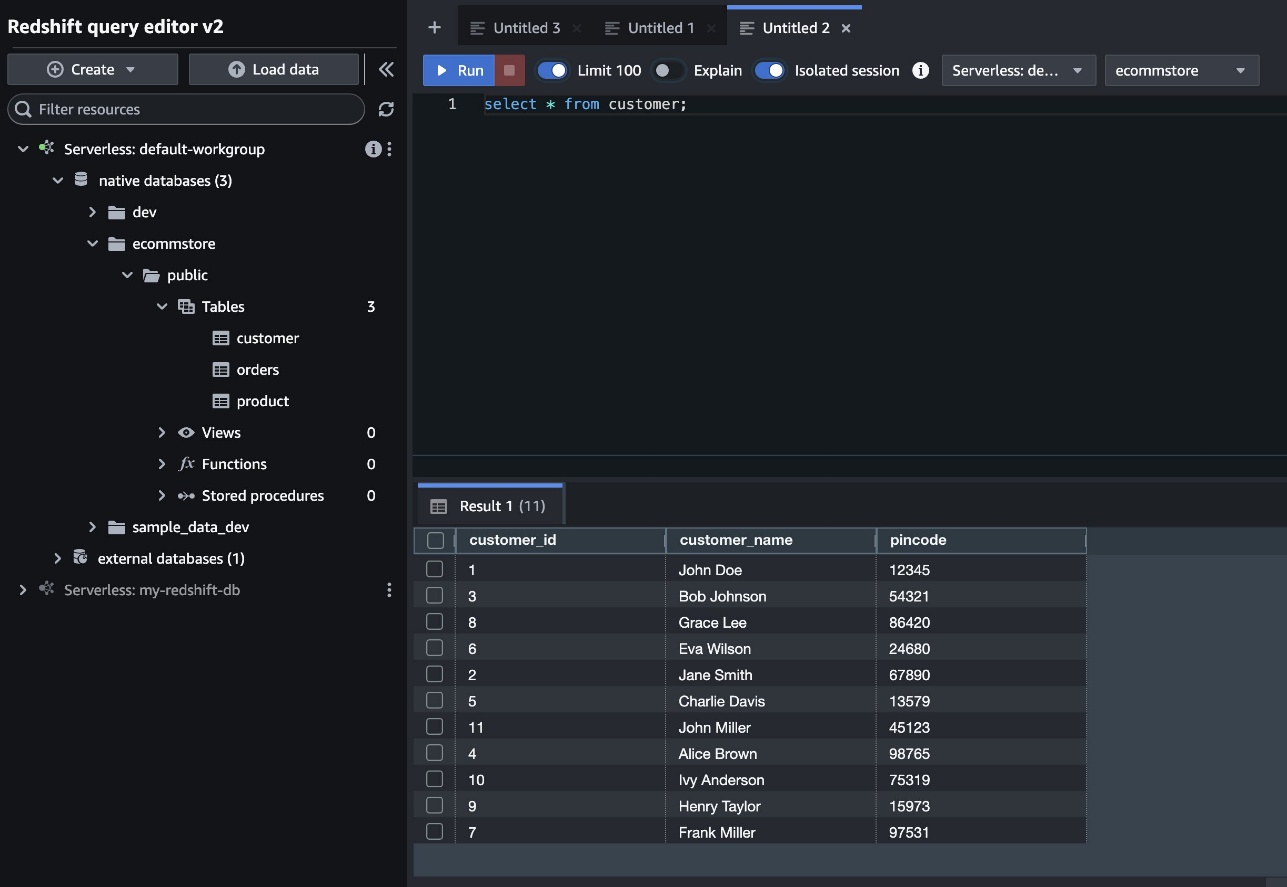


Ngoài ra, bạn có thể sử dụng **Redshift Query Editor v2** để xác minh rằng dữ liệu của bạn đã được đồng bộ thành công.

Một truy vấn đơn giản như:

SELECT \* FROM customer;

sẽ trả về dữ liệu đã được đồng bộ từ cơ sở dữ liệu **Aurora PostgreSQL**, như được hiển thị trong ảnh chụp màn hình dưới đây.



**Thiết lập Amazon Bedrock Knowledge Base với dữ liệu có cấu trúc**

Bước cuối cùng là tạo một **Amazon Bedrock knowledge base**, giúp bạn có thể **truy vấn dữ liệu bằng ngôn ngữ tự nhiên**.

**Tạo Amazon Bedrock Knowledge Base**

Tạo một **knowledge base mới trong Amazon Bedrock**, chọn tùy chọn **dữ liệu có cấu trúc (structured data)**.  
Để biết hướng dẫn chi tiết, hãy tham khảo: [**Build a knowledge base by connecting to a structured data store**](https://docs.aws.amazon.com/bedrock/latest/userguide/knowledge-base-create.html).

Sau đó, bạn cần **đồng bộ hóa query engine** để kích hoạt quyền truy cập dữ liệu.

**Cấu hình quyền truy cập dữ liệu**

Trước khi quá trình đồng bộ có thể diễn ra thành công, bạn cần [**AWS Identity and Access Management**](https://aws.amazon.com/iam/identity-center/)mà Amazon Bedrock Knowledge Bases sử dụng.  
Việc này bao gồm thực thi các lệnh **GRANT SELECT** cho từng bảng trong cơ sở dữ liệu Redshift của bạn.

Chạy lệnh sau trong **Redshift Query Editor v2** cho mỗi bảng bạn muốn truy vấn:

GRANT SELECT ON <table\_name> TO "IAMR:<tên IAM role của Knowledge Base>";

Ví dụ:

GRANT SELECT ON customer TO "IAMR:AmazonBedrockExecutionRoleForKnowledgeBase\_ej0f0";

**Đối với môi trường sản xuất (production)**

Để tích hợp **danh tính người dùng cuối** vào quy trình kiểm soát truy cập dữ liệu, bạn cần sử dụng **cơ chế liên kết danh tính (identity federation)**.

Vui lòng tham khảo tài liệu [AWS documentation on structured database access](https://docs.aws.amazon.com/bedrock/latest/userguide/knowledge-base-prereq-structured.html) để triển khai mô hình truy cập dựa trên vai trò (role-based access model).

Nếu ứng dụng web của bạn cần liên kết danh tính từ trình duyệt người dùng, bạn có thể cần:

* **,**[**Amazon Cognito**](https://aws.amazon.com/cognito/)
* [**SAML federation**](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_saml.html) **kết hợp với** [**AWS Security Token Service**](https://docs.aws.amazon.com/STS/latest/APIReference/welcome.html)**(AWS STS)**

Việc chọn giải pháp phụ thuộc vào kiến trúc hệ thống của bạn.

**Xác minh cấu hình**

Sau khi hoàn tất các bước cấu hình, Knowledge Base của bạn sẽ hiển thị các thông tin sau:

* **Trạng thái:** Available
* **Query engine:** đã đồng bộ thành công với **Amazon Redshift**
* **Database synchronization:** hiển thị trạng thái COMPLETE

Bây giờ, bạn đã có thể **bắt đầu truy vấn dữ liệu của mình bằng ngôn ngữ tự nhiên**.

**Ví dụ về các truy vấn ngôn ngữ tự nhiên**

Giờ đây khi bạn đã thiết lập xong **Amazon Bedrock Knowledge Base**, bạn có thể bắt đầu kiểm thử khả năng của nó bằng cách chạy các **truy vấn ngôn ngữ tự nhiên** trên dữ liệu có cấu trúc của mình.

**Amazon Bedrock Knowledge Bases (với dữ liệu có cấu trúc)** sẽ **chuyển đổi các câu hỏi tiếng Anh thông thường thành câu lệnh SQL**, sau đó sử dụng **mô hình nền tảng (Foundation Models - FMs)** để **tạo ra phản hồi dễ hiểu cho con người**.

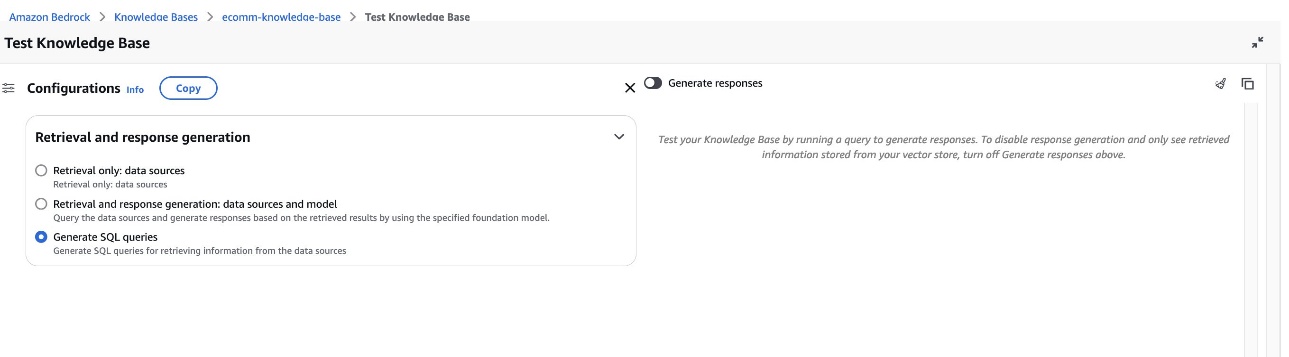
**Bạn có thể kiểm thử Knowledge Base theo hai cách:**

1. **Amazon Bedrock Console**
   * Trong bảng điều khiển Amazon Bedrock, chọn **Knowledge Bases** ở thanh điều hướng bên trái
   * Chọn Knowledge Base bạn đã cấu hình
   * Mở tab **Test** để tương tác trực tiếp với dữ liệu
2. **AWS SDK hoặc API**
   * Sử dụng SDK của AWS (ví dụ: [**Python’s Boto3**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/bedrock-agent-runtime/client/retrieve_and_generate.html))
   * Hoặc gọi trực tiếp [**RetrieveAndGenerate API**](https://docs.aws.amazon.com/bedrock/latest/APIReference/API_agent-runtime_RetrieveAndGenerate.html)để gửi truy vấn một cách lập trình (programmatically)

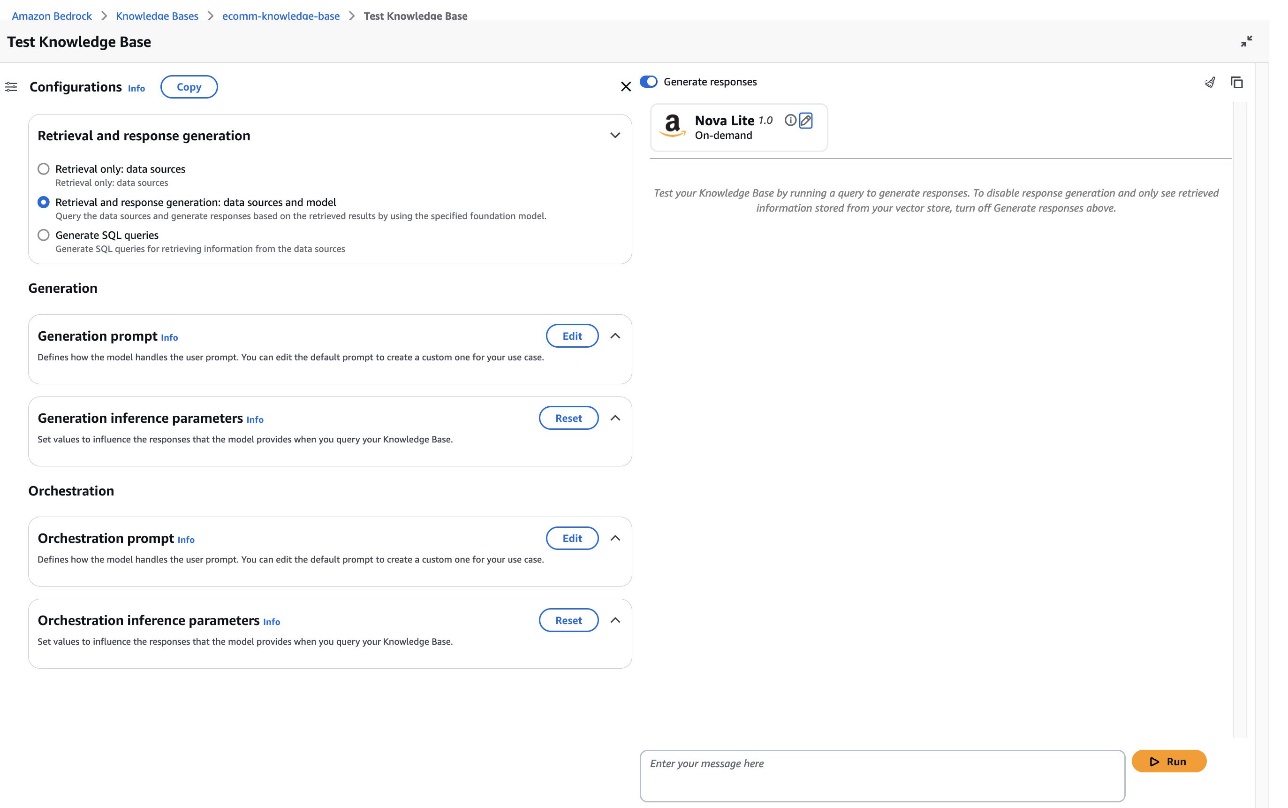
**Trong phần này, chúng tôi minh họa trải nghiệm trên bảng điều khiển (console)**

Trên bảng điều khiển Amazon Bedrock, bạn có thể tương tác với Knowledge Base của mình ở hai chế độ:

**Tạo truy vấn SQL (Generate SQL queries)** – Tùy chọn này sẽ hiển thị **câu lệnh SQL thô** được tạo ra từ **câu hỏi ngôn ngữ tự nhiên** mà bạn nhập vào.



**Truy xuất và tạo phản hồi (Retrieval and response generation)** – Tùy chọn này sẽ cung cấp **câu trả lời bằng ngôn ngữ tự nhiên** nhờ vào **mô hình ngôn ngữ lớn (LLM)**.



Bảng dưới đây chứa một số ví dụ về truy vấn ngôn ngữ tự nhiên, câu lệnh SQL tương ứng được tạo ra, và phản hồi do mô hình tạo ra:

| **Truy vấn ngôn ngữ tự nhiên** | **Câu lệnh SQL được tạo ra** | **Phản hồi từ mô hình** | **Model Used for Response Generation** |
| --- | --- | --- | --- |
| How many customers do we have? | SELECT COUNT(DISTINCT customer\_id) AS num\_customers  FROM public.customer; | We currently have 11 unique customers. | Amazon Nova Lite |
| Which all customers have purchased the most products? | SELECT c. "customer\_name", COUNT(o."product\_id")  AS "total\_products\_purchased"  FROM public.customer c  INNER JOIN public.orders o  ON c. "customer\_id" = o. "customer\_id"  GROUP BY c. "customer\_name"  ORDER BY "total\_products\_purchased" DESC; | Based on the data, the customers who have purchased the most products are Charlie Davis, Alice Brown, and John Doe, each having purchased 14 products. Following closely are Jane Smith, Grace Lee, and Bob Johnson, who have each purchased 13 products. Henry Taylor, Frank Miller, and Eva Wilson have each purchased 12 products, while Ivy Anderson has purchased 11 products. | Amazon Nova Lite |
| Who all have purchased more than one number of the most expensive product? | SELECT c.customer\_name  FROM public.customer c  INNER JOIN public.orders o  ON c.customer\_id = o.customer\_id  INNER JOIN public.product p  ON o.product\_id = p.product\_id  WHERE p.price = (SELECT MAX(price)  FROM public.product)  GROUP BY c.customer\_name  HAVING COUNT(DISTINCT o.order\_id); | The customers who have purchased more than one number of the most expensive product are Grace Lee, Jane Smith, Alice Brown, and Eva Wilson. | Amazon Nova Micro |

**Dọn dẹp tài nguyên (Clean up)**

Khi bạn đã hoàn tất việc sử dụng giải pháp này, hãy **xóa các tài nguyên đã tạo** để **tránh phát sinh chi phí liên tục**.

**Kết luận**

Trong bài viết này, chúng tôi đã trình bày cách **kích hoạt truy vấn ngôn ngữ tự nhiên trên dữ liệu Aurora PostgreSQL** bằng cách sử dụng **Amazon Bedrock Knowledge Bases** thông qua **tích hợp zero-ETL với Amazon Redshift**.

Chúng tôi đã hướng dẫn:

* Cách **thiết lập cơ sở dữ liệu** Aurora PostgreSQL
* Cách **cấu hình tích hợp zero-ETL** giữa Aurora và Redshift
* Cách **thiết lập kết nối knowledge base** để truy cập dữ liệu một cách liền mạch

Mặc dù đây là một giải pháp hiệu quả để tương tác với dữ liệu bằng ngôn ngữ tự nhiên, bạn nên **xem xét chi phí lưu trữ bổ sung phát sinh trong Amazon Redshift** khi áp dụng kiến trúc này cho các trường hợp sử dụng thực tế.

**Thông Tin Về Tác Giả**

A person with a mustache and glasses

AI-generated content may be incorrect.

**Girish B** là Kiến trúc sư Giải pháp Cấp cao (Senior Solutions Architect) tại AWS India Pvt Ltd, có trụ sở tại Bengaluru. Anh làm việc với nhiều khách hàng là nhà cung cấp phần mềm độc lập (ISV), hỗ trợ họ thiết kế và xây dựng các giải pháp sáng tạo trên nền tảng AWS.

A person in a blue shirt

AI-generated content may be incorrect.

**Dani Mitchell** là Kiến trúc sư Giải pháp Chuyên về Trí tuệ Nhân tạo Sinh ngữ (Generative AI Specialist Solutions Architect) tại AWS. Anh tập trung vào việc giúp các doanh nghiệp trên toàn thế giới tăng tốc hành trình ứng dụng AI sinh ngữ của họ với Amazon Bedrock.