Data Modeling for Amazon Neptune

**SPL-TF-200-DBANDM-1 - Version 1.0.3**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab Overview**

A business has defined a set of rules that describes what products to recommend when a customer is browsing through the web portal. Detailed requirements can be broken down into the following categories:

For a given Customer ID X:

* Find the list of recommended products with a 200ms SLA.
* Find products from the customer’s order history and recommend products from the product packages table.
* DO NOT offer the product if that product has already been purchased by someone in the same household.
* Only offer products with rating 3.5 and above.

In this lab you learn to model your source data from relational databases and document stores to Amazon Neptune in order to meet this business need. You will use a sample customer order database to build a product recommendation system which is used to send real-time product recommendations as customers are browsing the products portal.

With graph databases, it is a best practice to start working backwards from business goals or access patterns before you start with data modeling. This lab is broken down into 5 tasks that walk through the full scenario.

OBJECTIVES

By the end of this lab you should be able to do the following:

* Model source data from relational databases into vertices and edges in Amazon Neptune database.
* Extract embedded entities from source data.
* Use additional datasets to support new use cases with more relevant results.

TECHNICAL KNOWLEDGE PREREQUISITES

* Experience working with relational databases, database structures and data query language.
* Familiarity with foundational concepts of graph databases.
* Familiarity with Amazon Neptune basics.
* A basic competency with the AWS Management Console.

DURATION

This lab requires approximately *60* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Task complete:** A conclusion or summary point in the lab.
* **Note:** A hint, tip, or important guidance.
* **Learn more:** Where to find more information.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**Warning:** Do not change the **Region** unless instructed.

COMMON SIGN-IN ERRORS

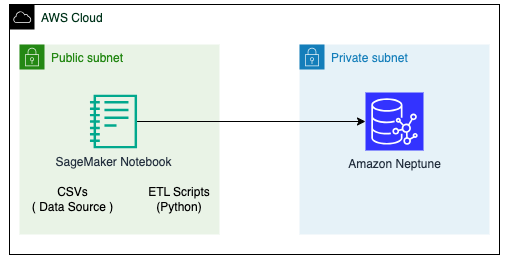
**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

LAB ENVIRONMENT

When you start the lab, the environment contains the resources shown in the following diagram:

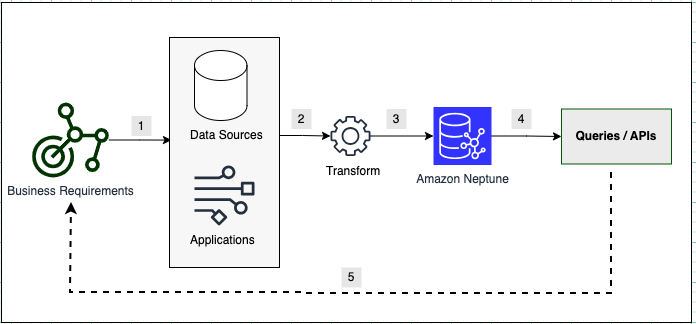


*Image description: The architecture contains Amazon SageMaker notebooks installed in the same VPC as Amazon Neptune. SageMaker notebooks explore source data, transforms data into a graph data model, and then loads that to Amazon Neptune.*

AWS SERVICES NOT USED IN THIS LAB

AWS service capabilities used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

GRAPH DATA MODELING OVERVIEW



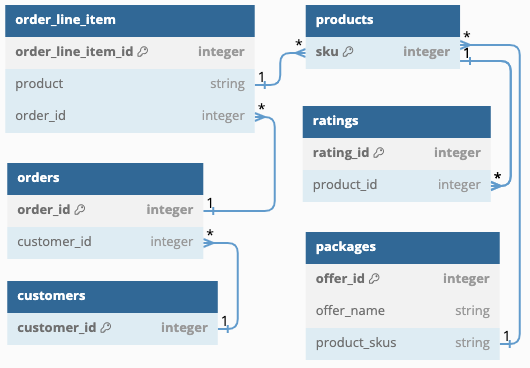
*Image description: In the preceding diagram, graph data modeling goes through steps 1 to 5 starting from the business requirements. The steps are as described below.*

* Step 1: Identify Data Sources – Identify data sources, columns, attributes or relationships in source data that is required to meet your business requirements. Each business use case may require an evaluation of what columns are needed to build the required data model.
* Step 2: Extract, Transform, Load (ETL) Scripts – Write ETL scripts either to convert source data in bulk or convert streaming data into a graph data model which includes converting rows to vertices and edges.
* Step 3: Ingestion – Both bulk ETL and streaming conversion will need to insert data into the graph database using API endpoints made available through Amazon Neptune.
* Step 4: Build Queries and APIs – Once the data is loaded, developers can start writing queries and build APIs to meet business requirements. Developers have a choice of building data in either Resource Description Framework (RDF) or Property Graph formats.
* Step 5: Evaluate Results – Evaluating your data model against business goals and SLAs is a crucial part of graph data modeling and development. If business goals are met then you can stop and deliver APIs. If not, then you have to go back to identify if you need more data sources, columns, or to model your data differently. It is a best practice to break down complex business requirements to microservices and APIs and then go through these steps for each API.

SOURCE DATA MODEL

The source data for this lab has been generated for a fictitious scenario of AnyCompany’s retail portal, which is using 6 tables in a relational database. Tables are exported to CSV and made available in the Jupyter Notebooks environment within the AWS Labs account. Here is the list of tables:

| **Sno** | **Value** | **Description** |
| --- | --- | --- |
| 1 | customers | customer attributes like address, ids, credit card info |
| 2 | orders | customers can place 1-to-many orders |
| 3 | order\_line\_item | orders contain line item for each product ordered |
| 4 | products | system of records for products table and its metadata |
| 5 | packages | A package is a combination of 2 or more related product offered to customers for product recommendation |
| 6 | ratings | Line item for each rating given by customers to product. |



*Image description: The preceding diagram shows an entity relationship for 6 source sales tables that are used to build a graph data model.*

In graph databases, it is customary to work backwards from the requirements to design the data model. Requirements rely on order history, customer household, packages and product ratings whose source data is present in a format that can be directly consumed or transformed into a format that can be ingested into the graph database. Neptune supports two graph model types: Property Graph and RDF. For this lab you use property graph and the gremlin query language to access the data model.

**Task 1: Setup Neptune utilities**

In this task, you log into the labs account and setup a graph notebook which is utilized to build the graph data model from source csv files.

**Learn more:** This lab uses a Neptune graph data model for managing the data. Refer to *Neptune Graph Data Model* in the **Additional resources** section for more information.

1. At the top of the AWS Management Console, in the search bar, search for and choose **Neptune**.
2. In the navigation pane at the left of the page, choose **Notebooks**.
3. Select the radio button for the notebook listed on the screen.
4. Choose **Actions** and select **Open Jupyter** which opens SageMaker notebook in a new tab.
5. In the SageMaker notebook select the **Neptune-Data-Modeling** folder link.
6. Choose the **Start-Labs.ipynb** link to launch the Jupyter Notebooks. This will open a graph-notebook in a new tab.

**Note:** To complete each task and sub-task, you run the corresponding code block in this notebook by selecting it, then choosing the **Run** button at the top of the screen. Review each code block as you run it.

1. Run the task 1.1 code block, which initializes the Neptune client to be used for the remaining tasks in the lab.

**Note:** You can skip task 1.2 if this is the first time you are running the lab. If you wish to restart the lab, come back and run the task 1.2 code block to delete any existing vertices and edges.

**Task complete:** You have successfully accessed graph-notebooks and are ready to model data for Amazon Neptune.

**Task 2: Store Rows as vertices ( Customers, Orders, Products, Product Packages )**

In this task, you convert each row in some relational database tables to a vertex in the graph database. Here are some of the design principles used in this task:

* Convert the column names to property keys.
* Use the table name as the label for each vertex.
* Use only columns that are required to meet the objectives of the data model (i.e building a product recommendation for a customer).
* Concatenate primary key values to generate each vertex ID.

TASK 2.1: STORE ROWS AND VERTICES AND COLUMNS AS PROPERTIES

1. Run the task 2.1.1 code block in the notebook.

**Expected output:**

The output returns a few lines from the customers.csv file. Out of the 20 columns showing, only 5 (*customer\_id, first\_name, last\_name, username,* and *email*) are needed for the lab objectives.

The code blocks labeled as tasks 2.1.2 -> 2.2.5 load customer, products, packages and orders into Neptune. These code blocks utilize the utils library initialized as part of Task 1.

1. Run the task 2.1.2 code block in the notebook.

**Expected output:**

'100 Customer:vertices inserted'

This shows that 100 *Customer* entities were inserted into the graph as vertices.

1. Run the task 2.1.3 code block in the notebook.

**Expected output:**

'20 Product:vertices inserted'

This shows that 20 *Product* entities were inserted into the graph as vertices.

1. Run the task 2.1.4 code block in the notebook.

**Expected output:**

'10 Package:vertices inserted'

This shows that 10 *Package* entities were inserted into the graph as vertices.

1. Run the task 2.1.5 code block in the notebook.

**Expected output:**

'112 Order:vertices inserted'

This shows that 112 *Order* entities were inserted into the graph as vertices.

**Note:** You have now finished loading all source CSV rows as vertices into the graph database.

TASK 2.2 : LIST VERTICES LOADED INTO NEPTUNE

**Learn more:** You can use gremlin queries to explore data loaded into Neptune. Refer to *Accessing a Neptune graph with Gremlin* in the **Additional resources** section for more information.

1. Run the task 2.2.1 code block in the notebook.

**Expected output:**

1 {'Order': 112}

2 {'Customer': 100}

3 {'Product': 20}

4 {'Package': 10}

After running, the code block returns the count of Order, Customer, Product and Package vertices from the source csv files.

1. Run the task 2.2.2 code block in the notebook.

**Expected output:**

1 {<T.id: 1>: '10047'}

2 {<T.label: 4>: 'Customer'}

3 {'first\_name': 'Nathan'}

4 {'last\_name': 'Bryan'}

5 {'username': 'jasminebrown@weaver.biz'}

6 {'email': 'jasminebrown@weaver.biz'}

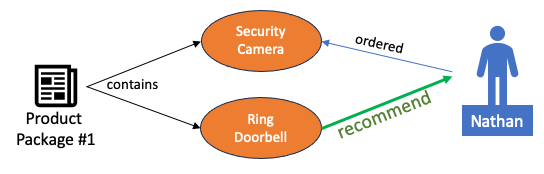
This shows a customer with a customer id of *10047* named *Nathan*. Further lab steps use this user.

**Task complete:** You have successfully modeled and loaded vertices to the graph database from each row in some CSV files.

**Task 3: Store relationships as edges**

In this task, you use existing keys in CSV data sets to build edges between the vertices.

**Note:** In graph databases, edges are the representation of relationships among vertices. Keys (primary keys or foreign keys) from relational databases can be used to build edges in graph databases. This task uses existing keys in CSV data sets to build edges between the vertices.



*Image description: The preceding diagram shows a logical data model for a customer named Nathan who ordered an item called Security Camera and should have an item called Ring Doorbell recommended to him as they are part of the same product package.*

Here are some of the design principles you will use in this task:

* Choose an edge direction and label that best express the domain semantics of the relationship.
* Concatenate primary and foreign key values to generate the edge ID.
* Choose a label for edges that make sense to business requirements (i.e customer -> placed --> orders where **placed** is the edge label. )

TASK 3.1 USE KEYS ( PRIMARY + FOREIGN ) AS DETERMINISTIC EDGE IDS

Foreign keys are used to create edge ids that are linking **customers** and **orders** entities.

1. Run the task 3.1 code block in the notebook.

**Expected output:**

'112 placed:edges inserted'

This shows that 112 edges of type *placed* have been inserted into the graph database. Edges link customers to orders vertices.

TASK 3.2: STORE ORDER TO PRODUCT EDGES

In the source tables, note that the *Orders* and *Order Item* tables have a one -> many relationship. Instead of storing each line-item, use *order\_line\_items.csv* to extract the relationship between orders and products.

1. Run the task 3.2 code block in the notebook.

**Expected output:**

'237 contains:edges inserted'

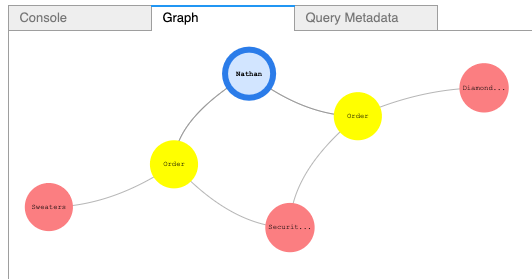
This shows 237 edges of type *contains* were inserted into the graph database. Edges link orders and product vertices.

TASK 3.3: VISUALIZE ORDERS AND PRODUCTS PURCHASED BY NATHAN

This visualizes the relationship between product and orders for customer *Nathan* that was created by previous tasks.

1. Run the task 3.3.1 code block in the notebook, then choose the **Graph** tab from the results.

**Expected output:**



*Image description: In the preceding image, a user Nathan is shown with 2 orders where he ordered 3 products ( Diamond Ring, Sweaters and Security Camera). Security Camera was ordered twice in 2 orders.*

**Note:** Visual orientation of vertices and edges might be different than what is shown in the picture.

You can confirm that there are two *Security Camera* purchases by listing just products purchased by *Nathan* using the next query.

1. Run the task 3.3.2 code block in the notebook.

**Expected output:**

1 {<T.id: 1>: '7000010', <T.label: 4>: 'Product', 'product\_name': 'Diamond Ring'}

2 {<T.id: 1>: '7000015', <T.label: 4>: 'Product', 'product\_name': 'Security Camera'}

3 {<T.id: 1>: '7000015', <T.label: 4>: 'Product', 'product\_name': 'Security Camera'}

4 {<T.id: 1>: '7000017', <T.label: 4>: 'Product', 'product\_name': 'Sweaters'}

This lists 4 products in orders for the customer Nathan. *Security Camera* was purchased twice, in 2 separate orders.

TASK 3.4: LINKING PRODUCTS TO PACKAGES

Each package in *packages.csv* contains multiple products that can be sold together or can be used to build product recommendations for customers. Explore *packages.csv* and run the code to create edges between products and packages. This code uses an edge label **contains** to represent package contents.

1. Run the task 3.4 code block in the notebook.

**Expected output:**

'20 contains:edges inserted'

This shows that 20 *contains* edges were inserted into the graph database. Edges link packages to product vertices.

TASK 3.5: QUERY TO GET PRODUCT RECOMMENDATION FOR NATHAN

Once links are created, you can walk through the graph to get a list of recommended products from Nathan’s product purchase history. The task 3.5 code block is doing the following:

* Start with Vertex id ‘10047’ which is the customer Nathan.
* Aggregate list of products purchased into custom aggregate *purchased\_products*.
* Find packages from purchased products.
* Find similar products from packages.
* Remove if they are already purchased.
* List products recommended and remove duplicates, if any.

1. Run the task 3.5 code block in the notebook.

**Expected output:**

1 {<T.id: 1>: '7000011', <T.label: 4>: 'Product', 'product\_name': 'Necklace'}

2 {<T.id: 1>: '7000016', <T.label: 4>: 'Product', 'product\_name': 'Winter Hat'}

3 {<T.id: 1>: '7000014', <T.label: 4>: 'Product', 'product\_name': 'Ring DoorBell'}

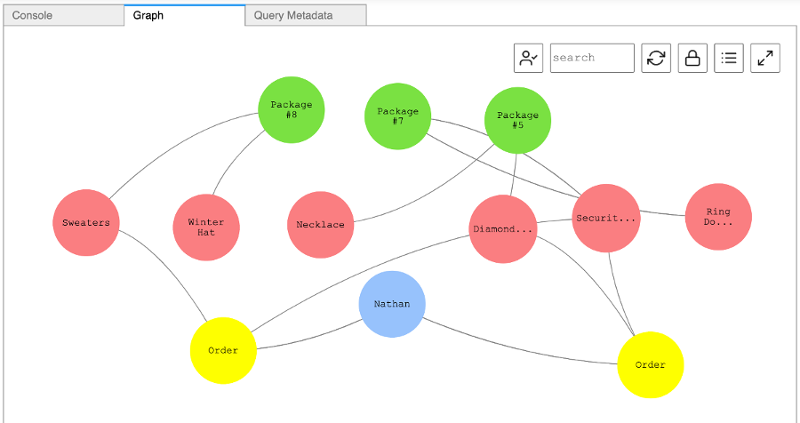
Notice that the query recommended that Nathan purchase a *Necklace*, *Winter Hat* and *Ring Doorbell* as they are part of the same product package (i.e Diamond Ring and Necklace are part of the same package #5).

TASK 3.6 VERIFY RESULTS FOR NATHAN THROUGH VISUALIZATION

You can also verify the results through visualization by building the path between Nathan and recommended products.

1. Run the task 3.6 code block in the notebook, then choose the **Graph** tab from the results.

**Expected output:**



*Image description: The preceding image shows the order history of 2 orders and 6 products. Those 6 products purchased by Nathan are part of 3 different packages.*

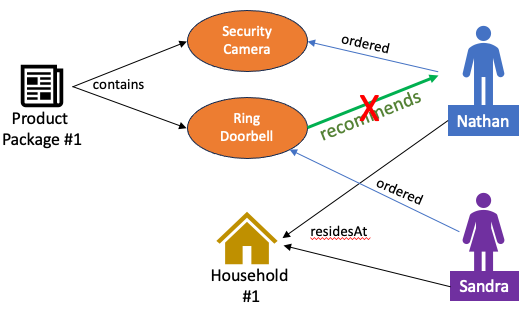
**Note:** Visual orientation of vertices and edges might be different than what is shown in the picture.

**Task complete:** You have converted foreign keys to relationships and built your first product recommendation query.

**Task 4: Extract embedded entities and identify households**

In this task, you denormalize addresses into separate vertices and link them to customer vertices. You then make the recommendation smarter by excluding products that have already been purchased by another household member.

At times, graph data modeling requires extracting vertices and edges that are available as columns in source data. In the case of *customers.csv*, address information is embedded within the customer entity and broken down into multiple columns.



*Image description: The preceding image shows a customer named Sandra who resides at the same address as Nathan. Therefore, the recommendation query for Nathan excludes any products that have already been ordered by Sandra. As a result of this, the Ring Doorbell is not recommended to Nathan*

TASK 4.1: EXTRACT HOUSEHOLDS AND STORES AS ENTITIES

The first code block extracts **address\_line**, **city**, **state** and **zipcode** and inserts these as unique households. As the code is inserting the household vertices it also creating the edge between *customer* and *household*.

1. Run the task 4.1 code block in the notebook.

**Expected output:**

100 Household:vertices inserted

100 residesAt:edges inserted

This shows that the code block identified and added 100 *Household* vertices across the customer database and added 100 *residesAt* edges from customers to those household vertices.

TASK 4.2 FIND PRODUCT RECOMMENDATION FOR NATHAN

Now you can use the modified product recommendation query that uses the new **residesAt** edge to exclude products that were already purchased by a member of this household.

1. Run the task 4.2 code block in the notebook.

**Expected output:**

1 {<T.id: 1>: '7000008', <T.label: 4>: 'Product', 'product\_name': 'Christmas Decoration'}

2 {<T.id: 1>: '7000016', <T.label: 4>: 'Product', 'product\_name': 'Winter Hat'}

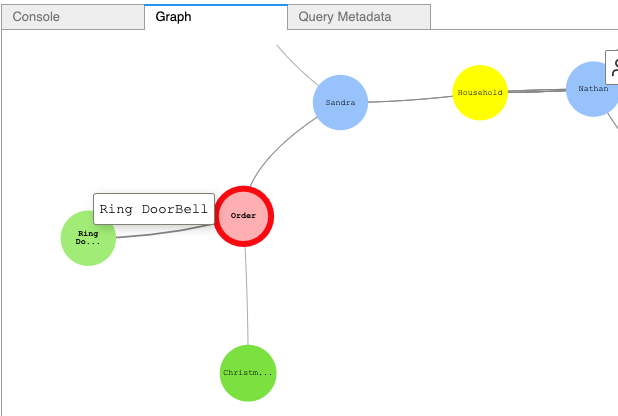
Notice in the output above that *Ring Doorbell* is no longer suggested, as it has already been purchased by Sandra who lives in the same household.

TASK 4.3 VALIDATE PRODUCT RECOMMENDATION FOR NATHAN USING THIS VISUALIZATION

You can validate the previous results through visualization that shows Sandra has already purchased a *Ring Doorbell*.

1. Run the task 4.3 code block in the notebook, then choose the **Graph** tab from the results.

**Expected output:**



*Image description: The preceding image shows that Sandra, who has already purchased a Ring Doorbell, is linked to Nathan as a household.*

**Note:** Visual orientation of vertices and edges might be different than what is shown in the picture.

**Task complete:** You have extracted embedded entities and stored them to the graph. Your recommendation query has also improved as it uses a customer’s household purchases as input.

**Task 5: Use additional data sources to improve recommendations**

In this task, you integrate an additional dataset to incorporate new use cases. Many times additional datasets are added in the later stages of project development, which may lead to a change in the requirements. In this case, a *Product Ratings* table is used to exclude products that may be rated too low to recommend.

TASK 5.1 EXPLORE RATINGS DATA

The ratings table stores individual rows of customer ratings for a product.

1. Run the task 5.1 code block in the notebook.

**Expected output:**

| **rating\_id** | **product\_id** | **date** | **value** | **customer\_id** |
| --- | --- | --- | --- | --- |
| 700011 | 7000010 | 2023-09-27 23:31:30.661879 | 3 | 10062 |
| 700011 | 7000010 | 2023-09-27 23:31:30.661879 | 4 | 10062 |
| 700011 | 7000010 | 2023-09-27 23:31:30.661879 | 1 | 10076 |
| 700011 | 7000010 | 2023-09-27 23:31:30.661879 | 1 | 10076 |
| 700011 | 7000010 | 2023-09-27 23:31:30.661879 | 1 | 10018 |

This table shows a line item for each time a product was rated.

TASK 5.2 CREATE AVERAGE/MEAN OF RATINGS AND STORE WITH PRODUCT

The business requirement is to use the average ratings of products. You can calculate product average using the *Ratings* table and store that as a property with product vertices.

1. Run the task 5.2 code block in the notebook.

**Note:** This code block does not return any output. It is adding the *average\_rating* of the product as a property.

TASK 5.3 CHECK AVERAGE RATINGS OF PRODUCTS RECOMMENDED TO NATHAN

You can use the same product recommendation query as used before but this time you can also list the *average\_rating* property of the products.

1. Run the task 5.3 code block in the notebook.

**Expected output:**

1 {<T.id: 1>: '7000008', <T.label: 4>: 'Product', 'product\_name': 'Christmas Decoration', 'average\_rating': 2.6666666666666665}

2 {<T.id: 1>: '7000016', <T.label: 4>: 'Product', 'product\_name': 'Winter Hat', 'average\_rating': 3.5}

Note that of the two recommended products, the *Christmas Decoration* has an average of *2.6*, which is below the business requirements threshold.

TASK 5.4 REMOVE PRODUCTS WITH AN AVERAGE RATING LESS THAN 3.5 OUT OF 5

The final code block completes the business requirements. It queries for products to recommend for a given user, based on these criteria that fit the requirements:

* Offer products related to the user’s previous purchases, by offering products in the same product package as the user’s previous purchases.
* Only offer products that have not already been purchased by someone in the same household.
* Only offer products that are rated 3.5+ and above.

1. Run the task 5.4 code block in the notebook.

**Expected output:**

{<T.id: 1>: '7000016', <T.label: 4>: 'Product', 'product\_name': 'Winter Hat', 'average\_rating': 3.5}

Note that the *Christmas Decoration* is no longer recommended, as it fell below the 3.5 rating threshold.

**Task complete:** You used additional datasets to improve the results of the existing recommendation query. Ratings data was added as an attribute *average\_rating* in the product vertices.

**Conclusion**

 Congratulations! You now have successfully:

* Modeled source data from relational databases into vertices and edges in Amazon Neptune database.
* Extracted embedded entities from source data.
* Used additional datasets to support new use cases with more relevant results.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional Resources**

* [Neptune Graph Data Model](https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html).
* [Accessing a Neptune graph with Gremlin](https://docs.aws.amazon.com/neptune/latest/userguide/access-graph-gremlin.html).

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).