**Product Catalogue Search**

* **Technologies Used**

1. Spring Boot Web Framework
2. Spring JPA Repository
3. In memory H2 Database
4. Postman Client

* **Database Design Considerations**

1. In order to have maximum flexibility in database table structure for the Product Catalogue, I have followed EAV Design Pattern for database Design.

**EAV, stands for Entity-Attribute-Value** design pattern.

2. This application is having the following database schema,

* + **Entities <Attributes>**
  + **Product <ID, NAME, DESCRIPTION, SKU, QUANTITY, LIST<CATEGORY>, LIST< ATTRIBUTE\_VALUE>>**
  + **Attribute <ID, NAME, IS\_ACTIVE, LIST< ATTRIBUTE\_VALUE>>**
  + **AttributeValue <ID, DATA, IS\_ACTIVE, *ATTRIBUTE\_ID*, *PRODUCT\_ID*>**
  + **Category <ID, NAME, DESCRIPTION, IS\_ACTIVE, *PRODUCT\_ID* >**

**Note:** Each Entity/Table has Column **ID** as Primary Key

Above underlined columns are Foreign Keys to the Parent Table Primary Key(ID)

* + **Entities <Relationships>**
  + **Product 🡪 One-To-Many 🡪 Category**
  + **Product 🡪 One-To-Many 🡪 AttributeValue**
  + **Attribute 🡪 One-To-Many 🡪 AttributeValue**

3. We can add any number of attributes for the product dynamically, without disturbing any of the existing table structures.

4. Similarly, any number of categories for the product can also be added.

**For Instance,** currently the data.sql file present in /src/main/resources folder package inside the project preloads the in-memory database with a couple records for all the tables. Example,

**Product Attributes are – Color, Size, Brand, Price, Seller** all added dynamically to the table Attributes, and the product can have any new product added on the fly without changing the table schema.

5. Referential Integrity constraints are being taken care of by using the ORM Frameworks Annotations like - CascadeType

* **Assumptions**

1. Database will be prepopulated using the data.sql script by the Framework. As it was not mentioned in the Case Study Description, the API does not support any kind of POST requests handling as of now.
2. Only the queries mentioned in the Case study are implemented along with getAll queries for all Entities. Those are implemented to test the created schema.
3. It is assumed that the Group By operations used to fetch COUNT aggregation for the Products. As no Aggregator function was specified in points 1,2,3,4 mentioned in the case study requirement document.
4. It is assumed that the Response Format is basic minimal JSON. As no formal specifications were given for the Response Body.
5. It is assumed that a Product can have multiple categories like – Pant/Trousers etc at a time.
6. It is assumed that all the Products are in active state. This attribute of the product can be manipulated to support any specific filter criteria.
7. All the service classes have been given concrete implementations without specifying any generic interfaces, due to time constraints.

* **Unit Test Cases**
* Test the happy path scenario after inserting data to all four tables.
* Test Product and Category tables for the foreign key constraints.
* Test Product and AttributeValue tables for the foreign key constraints.
* Test Attribute and AttributeValue tables for the foreign key constraints.
* Test the SKU Attribute for Unique constraint.
* Test query for a Product having multiple categories.
* Test query for a Product with Quantity 0
* Test query for a Product for a missing attribute, it returns an empty list.
* Test for nullable attributes.
* **Sample URLs to run the Application**

<http://localhost:8080/products/group/Brand>

<http://localhost:8080/products/group/Size>

<http://localhost:8080/products/group/Color>

<http://localhost:8080/product/900471>

<http://localhost:8080/products/available/Seller/OmniRetail>

Returns Product count for those Products which are available meaning having quantity greater than 0, given the Seller name.

* **Future Scope**

1. This Application has the design that can support extension to fully grown Ecommerce portal for a variety of Products.
2. Implementation of multiple types of Product Search options with/without filters can be easily incorporated.
3. Update and Delete features for the Products and Attributes could also be incorporated.