

Product Attributes

Category\_Id

Name (VARCHAR)

Description (Text)

Categories

Attribute\_Id

Product\_id (FK)

Attribute\_name (VARCHAR)

Attribute \_value (Text)

Products

Product\_Id

Product\_id (FK)

Name (VARCHAR)

Price (Decimal)

Image.url (VARCHAR)

Description (Text)

**Database Design**

**Database Design Explanation:**

Entities and Attributes Breakdown:

* Categories Table:
  + category\_id: Primary Key, VARCHAR(36) (for UUID) or INT (auto-increment). Uniquely identifies each product category.
  + name: VARCHAR(255). The name of the category (e.g., "Smartphones", "Dresses"). Must be unique.
  + description: TEXT. An optional, longer description for the category.
* Products Table:
  + product\_id: Primary Key, VARCHAR(36) (for UUID) or INT (auto-increment). Uniquely identifies each product.
  + category\_id: Foreign Key referencing Categories(category\_id). Links a product to its specific category.
  + name: VARCHAR(255). The name of the product (e.g., "iPhone 15", "Leather Loafers").
  + price: DECIMAL(10,2). The price of the product, allowing for 8 digits before and 2 digits after the decimal point.
  + image\_url: VARCHAR(2048). A URL pointing to the product's image. (2048 characters to accommodate longer URLs).
* Product\_Attributes Table:
  + attribute\_id: Primary Key, VARCHAR(36) (for UUID) or INT (auto-increment). Uniquely identifies each specific attribute value entry.
  + product\_id: Foreign Key referencing Products(product\_id). Links this attribute value to a specific product.
  + attribute\_name: VARCHAR(255). The name of the custom attribute (e.g., "OS", "RAM", "Length").
  + attribute\_value: TEXT. The value for that custom attribute for the specific product (e.g., "Android 15", "12GB", "Knee-length").

Relationships:

* One-to-Many (Categories to Products):
  + A single Category can be associated with multiple Products.
  + Each Product belongs to exactly one Category.
  + This is enforced by the category\_id in the Products table acting as a Foreign Key to the Categories table's category\_id.
* One-to-Many (Products to Product\_Attributes):
  + A single Product can have multiple custom Product\_Attributes (each row represents one custom attribute-value pair for that product).
  + Each Product\_Attribute entry is specific to one Product.
  + This is enforced by the product\_id in the Product\_Attributes table acting as a Foreign Key to the Products table's product\_id.

Justification of Design Decisions

This database schema leverages normalization to ensure data integrity, flexibility, and scalability.

1. Dynamic Product Categories & Custom Attributes (Flexibility & Scalability):
   * The Product\_Attributes table is the key to supporting dynamic categories and custom attributes. Instead of adding specific columns like os\_type, ram\_size, dress\_length directly to the Products table, which would require schema changes for every new attribute or category type, we use an Entity-Attribute-Value (EAV) like model in Product\_Attributes.
   * This means if you introduce a "Laptops" category that needs "Processor Type" and "Storage Capacity" attributes, you don't modify the Products table. You simply add rows to Product\_Attributes for each laptop, specifying attribute\_name and attribute\_value. This makes the system extremely flexible and scalable for future expansion without database downtime or complex migrations.
2. Normalization (Third Normal Form - 3NF) (Data Integrity & Efficiency):
   * By separating Categories, Products, and Product\_Attributes into distinct tables, we eliminate data redundancy. For example, a category's name and its common attributes are defined once in the Categories table, not repeated for every product in that category. This saves storage space and reduces the risk of inconsistencies.
   * Changes to category details or attribute names only need to be made in one place, preventing update anomalies (where updating one piece of data requires updating many records) and deletion anomalies (where deleting a record inadvertently deletes related, necessary data).
3. Future-Proofing (Adaptability):
   * This design is highly adaptable to future requirements. You can easily add new categories, new types of products, or new custom attributes without altering the core table structures. This agility is crucial for an evolving e-commerce platform.
   * Using VARCHAR(36) for IDs is a common practice for UUIDs (Universally Unique Identifiers), which ensures global uniqueness even if data is merged from different systems or generated concurrently, avoiding ID conflicts.