



**BERLIN SCHOOL OF  
BUSINESS & INNOVATION**

**Essay / Assignment Title: Online Marketplace Data Warehouse  
Management & Business Analytics**

**Programme title: Enterprise Data Warehouses and Database  
Management Systems**

**Name: Anusha Kelagina Peredelu**

**Year:2025**

# CONTENTS

CONTENTS.....	2
LIST OF FIGURES .....	3
ABSTRACT.....	6
CHAPTER ONE: INTRODUCTION .....	7
CHAPTER TWO: ENTITY RELATIONSHIP DIAGRAM.....	9
CHAPTER THREE: DATABASE SCHEMA .....	13
CHAPTER FOUR: SQL QUERY GENERATION.....	23
CHAPTER FIVE: MODEL EVALUATION.....	33
BIBLIOGRAPHY .....	40

## LIST OF FIGURES

Figure 1:Entity Relationship Diagram .....	11
Figure 2:Schema creation .....	15
Figure 3:Table creation .....	15
Figure 4:Table “cost” .....	16
Figure 5:Foreign key in cost table .....	16
Figure 6:Table “coupon” .....	17
Figure 7:Table “customer” .....	17
Figure 8:Table “offers” .....	17
Figure 9:Table “orders” .....	18
Figure 10:Table “orderdetails” .....	18
Figure 11:Table “payments” .....	19
Figure 12:Table “products” .....	19
Figure 13:Table “revenue” .....	19
Figure 14:Table “review” .....	20
Figure 15:Table “vendor” .....	20
Figure 16:Code snippet for data generation .....	21
Figure 17:Table “Orders” with the data .....	21
Figure 18:Regenerated ER Diagram .....	22
Figure 19:Stored Procedure for calculating monthly report .....	23
Figure 20:Output of stored procedure .....	24
Figure 21:Trigger applied on “orders” table before insert statement .....	25
Figure 22:Trigger Output .....	26
Figure 23:Query-1 .....	26

Figure 24:Query-1 Output.....	27
Figure 25:Query-2.....	27
Figure 26:Query-2 Output.....	28
Figure 27:Query-3.....	28
Figure 28:Query-3 Output.....	29
Figure 29:Query-4.....	29
Figure 30:Query-4 Output.....	30
Figure 31:Query-5.....	30
Figure 32:Query-5 Output.....	31
Figure 33:Query-6.....	31
Figure 34-Query-6 Output .....	32
Figure 35:Analytical SQL query .....	33
Figure 36:Stored Procedure with ACID properties .....	35

## **Statement of compliance with academic ethics and the avoidance of plagiarism**

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the postgraduate program).



Name and Surname (Capital letters):

ANUSHA KELAGINA PEREDELU

Date: 2025/02/03

## ABSTRACT

In this assignment, I will be creating an analytical data warehouse for clothing industry using MySQL. The Online Clothing Marketplace is an e-commerce platform that helps in connecting customers, vendors, and operational services. In the present growing society, digitization has a great demand. One such requirement is digital retail solutions. Considering this requirement, the marketplace provides a very beneficiary platform providing online transactions in the clothing industry. Key features such as secure payment gateways, offering coupons, and customer review systems distinguish this platform as a number one choice for customers of this generation. This report explores the objectives, features, and operational flow of the Online Clothing Marketplace, with its great approach to simplifying and enhancing the online shopping experience for all stakeholders.

**Keywords:** Clothing marketplace, Payment gateways, Coupons, Review systems, MySQL.

## CHAPTER ONE: INTRODUCTION

The Online Clothing Marketplace is the platform for finding easy way and having variety of options in online shopping for the consumers. As customers search for different options while comparing price, businesses should adopt unique approaches to stand out in a competitive digital market. This clothing marketplace helps to build a platform which provides customer, a satisfaction of online shopping and vendor to grow eventually. This even addresses key points such as limited options, unverified product quality, and inefficient delivery processes. It provides a platform that not only meets but exceeds customer expectations by offering one shop for all clothing needs.

This database has tables such as products, vendor, revenue, customer, review, orders, order details, payments, cost and coupon. Product, vendor, revenue includes the product details, vendor information and revenue details respectfully. Then followed by customer which has the personal details of the customer, followed by orders and order details of the products ordered by customer, review includes the reviews provided by the customer. This platform also includes a table for cost detail which showcases the cost that has been added to each order, payment details for a secure payment and coupon facility for customers to get offers on each order.

Here is a small story which tells us the importance of this online clothing marketplace.

**Customer:** Anaya, is a working professional who is out looking for a dress for an event. She does not have time to go to any of the local shops. Therefore, she chooses to sign into the Online Clothing platform where she witnesses multiple ranges of dresses that are sorted by size, type, fabric and even cost. She uses the platform's filters to narrow down her options and chooses a dress with excellent customer reviews. After applying a discount coupon, she makes a secure payment and tracks her order until it is delivered on time. Delighted with the quality, Anaya leaves a five-star review, helping other customers make informed decisions.

**Vendor:** Rahul, a small-scale clothing manufacturer, joins the Online Clothing Marketplace to build his own business. Using the vendor tool available in the platform, he will list his products with detailed descriptions and competitive pricing. Within few weeks, his products will be sold out at a fast pace due to positive customer reviews and high ratings. He then utilizes the

platform's revenue analytics to identify best-selling items and adjust his stock accordingly. This platform also ensures timely shipments, increasing Rahul's sales. Over time, he grows his business.

The unique features in our Database and effective tracking for the payments and shipments makes every transaction logged, enables in gaining valuable insights which enhances customer experience. The Product review by Customers increases its authenticity. This platform minimizes delays and maximizes customer satisfaction.

This approach displays the platform's potential to transform the online shopping experience. Whether you are a customer looking for a convenient shopping, a vendor struggling for growth, or an operator ensuring smooth workflows, the Online Clothing Marketplace is the platform that delivers it all.



## CHAPTER TWO: ENTITY RELATIONSHIP DIAGRAM

An **entity** refers to concept or object that can be identified and represented within a database. An **attribute** is a characteristic of an entity that describes or identifies it. It stores the details related to an entity. In my database, that is online clothing marketplace, there are 10 entities namely Customer, Order, Order details, Review, Product, Cost, Payments, Coupon, Revenue, Vendor. Below are the details on the available entities and their corresponding attributes.

**Product:** It gives us the available product in the marketplace. The entities include primary key that is the **ProductID**, which is the unique attribute. **Name** is the title of the product and **Description** gives the details of that particular product in short. **Category** is the type of product for example men/women/kids. **Price** is the selling price, while the **Material** describes the composition of the product, such as cotton or silk. **Brand**, which tells us the manufacturer, **Size** and **Color** are the size and color in which the products are available, **StockQuantity**, which is the quantity available in stock, and **Average rating**, the rating of the product given by other customers.

**Customer:** This entity is the person who orders from the marketplace. The primary attribute, **CustomerID**, uniquely identity for each customer. Other attributes are **Name**, stores the name of the customer. The contact information is provided through **Email** and **PhoneNumber**, **MembershipStatus** describes whether the customer is a registered member or a guest user. Personal details about the customer include **DateOfBirth** and **Gender**. While the location of the customer is given in **Address**. **JoinDate** describes when the customer registered on the site.

**Orders:** This entity represents the information required to place the customer's orders. It contains a primary key, which is **OrderID**; **OrderDate** refers to the date on which this specific order was made. **Totalamount**, the total amount of the order placed. Delivery information attributes include **ShippingAddress**, showing the location where product will be delivered; **DeliveryDate** provides the exact date when product will be delivered. **TrackingNumber** which gives out a shipment-tracking code that is unique. **DeliveryStatus**, information about delivery if the delivery is complete/incomplete.

**OrderDetails:** This entity gives us the details of each particular order. This is uniquely identified by the **OrderDetailsID** primary key. The **OrderQuantity** attribute is the number of products ordered. **Discount** attribute tells us if there is any discount applied on the item, while **TaxAmount** is the tax calculated on the product. The **Subtotal** attribute sums up the total amount which includes the tax and discount. It also includes the **ShippingCost** attribute, which is the delivery cost of that particular product.

**Review:** Represents the customers opinion about the product they purchased. Each review has a unique attribute called **ReviewID**. **Rating** is for rating the customers feedback on a scale from 1 to 5. The **ReviewComments** attribute is used to store the text of the customers review, and **TimeSpent** is to denote the duration in minutes spent by the customer. The date on which the feedback was given has been recorded in **ReviewDate**. The attributes **DeviceUsed**, which says whether the review was submitted via Mobile or Desktop, and **Likes** represents the number of likes given by the customers to the product.

**Cost:** The **Cost** entity which specifies each cost incurred with the product. Specifically, those related to logistics and fraud. It has a unique attribute **Cost ID** and includes attributes such as **Packaging Cost** and **Advertising Cost**, which are the relevant operational costs. **Delivery Failure Cost** and **Reattempt Cost** track costs are the cost of unsuccessful deliveries, while **Fraud Cost** is the expense arising from fraudulent product returns.

**Payments:** The entity Payments describes the transactions made by customers in purchasing a product. **PaymentID** is the primary key. **Payment method** specifies the mode of payment (E.g., credit card, PayPal). **Payment Date** is the date on which the transaction was made. **Payment Status** indicates whether the payment was success or a failure.

**Coupon:** The entity coupon is the promotional codes offered to the customers. It is identified with a unique key that is **CouponID**. The attribute **Code** is the unique coupon string. **Discount Rate** specifies the percentage of the reduction amount offered in the coupon. **Expiry Date** is the date until when the coupon is valid.

**Vendor:** The entity gives the details about sellers on this marketplace. This has a unique attribute that is **VendorID**. Attributes like **Name**, **Phone Number**, and **Email** offer basic contact information for the vendor. **Location** gives the vendor's business address. **Vendor Rating** shows

the customer feedback on the vendor's service. The **Date of Registration** is the vendor's onboarding date, and **Vendor Status** is their current operational status (e.g., active or inactive). **Total Sales** tracks the revenue generated by the vendor's products on the platform.

**Revenue:** The entity calculates the earnings generated by the platform from vendor sales. It is identified by **Revenue ID** and includes attributes like **Commission Rate**, which defines the percentage charged to vendors for each transaction. **Discount Provided** records any price reductions granted, while **Tax Deducted** captures tax obligations. **Net Revenue** calculates the earnings retained after all deductions.

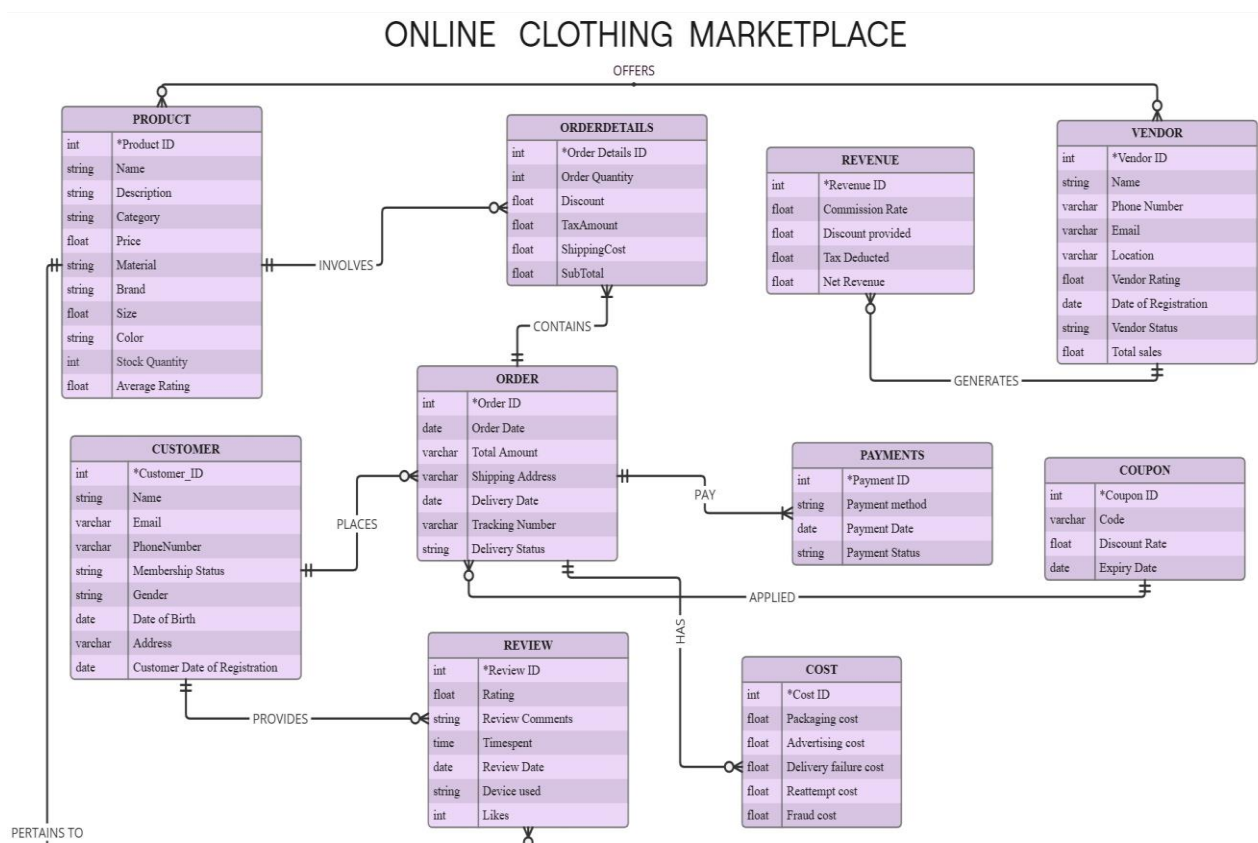


Figure 1:Entity Relationship Diagram

Figure 1, illustrates the relationships, participation, and cardinalities between entities in the online clothing marketplace. The **Customer** entity is related to the **Orders** which has a one-to-many relationship named “PLACES”, here a customer can place multiple orders (1: N cardinality), but each order belongs to only one customer. The **Orders** entity is connected to the **Payments** entity with one-to-many relationship “PAY”, where each order must have one

associated payment or more which includes success or failure (1:N cardinality), and each payment is of only one order.

The **Coupon** entity is connected to the **Orders** entity through the "APPLIED" relationship(many-to-one), where each order can have only one coupon (N:1 cardinality), and a coupon can be applied to multiple orders. The **Vendor** entity is connected to the **Revenue** entity through the "GENERATES" relationship(one-to-many). Each vendor can contribute to multiple revenue entries (1:N cardinality), and each revenue entry is linked to one vendor.

The **Product** entity is linked to the **OrderDetails** entity via "INVOLVES" relationship(one-to-many), A product can be included in many order details (1:N cardinality), but each order detail belongs to exactly one product. The **Orders** entity is also connected to the entity **OrderDetails** through the "CONTAINS" relationship(one-to-many). An order must have one or multiple order details (1:N cardinality), while each order detail is of only one order.

The **Customer** is connected to **Review** via "PROVIDES" relationship(one-to-many), where each customer can leave multiple reviews (1:N cardinality), and each review is associated with one customer. Lastly, the **Orders** entity is linked to the **Cost** entity through the "PERTAINS TO" relationship (One-to-many). Each order can have multiple costs (1:N cardinality), while each cost entry is associated with a single order. **Product** is linked to **Vendor** with a many-to-many relationship "OFFERS" where each product can be associated with many vendors as well as vendors may offer many products(M:N)

## CHAPTER THREE: DATABASE SCHEMA

A schema is the structural representation of the entities and their relationships with one another. Below are the schema details for online clothing marketplace.

### 3.1 SCHEMA DETAILS

**customer (\*customer\_id, name, email, phone\_number, membership\_status, gender, dob, address, join\_date)**

The Customer schema has details about users with attributes such as customer\_id (primary key), name, email, phone\_number, membership\_status, gender, dob (date of birth), address, and join\_date (customer's registration date).

**orders (\*order\_id, order\_date, total\_amount, shipping\_address, delivery\_date, tracking\_number, delivery\_status, FK (customer\_id), FK (coupon\_id))**

The Orders schema contains information, including order\_id (primary key), order\_date, total\_amount, shipping\_address, delivery\_date, tracking\_number, and delivery\_status. It includes foreign keys (customer\_id and coupon\_id) to associate orders with customers and applied coupons.

**orderdetails (\*orderdetails\_id, order\_quantity, discount, tax\_amount, shipping\_cost, sub\_total, FK (order\_id), FK (product\_id))**

The OrderDetails schema has details about order, with attributes such as orderdetails\_id (primary key), order\_quantity, discount, tax\_amount, shipping\_cost, and sub\_total. It also has foreign keys (order\_id and product\_id) to link order details to specific orders and products.

**review (\*review\_id, rating, review\_comments, timespent, review\_date, device\_used, likes, FK (customer\_id), FK (product\_id))**

The Review schema has customer feedback and has attributes like review\_id (primary key), rating, review\_comments, timespent, review\_date, device\_used, and likes. It also has foreign keys (customer\_id and product\_id) to link reviews to specific customers and products.

**coupon (\*coupon\_id, code, discount\_rate, expiry\_date)**

The Coupon schema has promotional details, with attributes such as coupon\_id (primary key), code, discount\_rate, and expiry\_date.

**payments (\*payment\_id, payment\_method, payment\_date, payment\_status, FK(order\_id))**

The Payments schema has payment information, including payment\_id (primary key), payment\_method, payment\_date, and payment\_status. It uses the foreign key order\_id to link payments to specific orders.

**cost (\*cost\_id, packaging\_cost, advertising\_cost, delivery\_failure\_cost, reattempt\_cost, fraud\_cost, FK (order\_id))**

The Cost schema captures expenses related to orders, with attributes such as cost\_id (primary key), packaging\_cost, advertising\_cost, delivery\_failure\_cost, reattempt\_cost, and fraud\_cost. It also includes the foreign key order\_id to associate costs with specific orders.

**products (\*product\_id, name, description, category, price, material, brand, size, color, stock\_quantity, average\_rating)**

The Products schema has product information, with attributes such as product\_id (primary key), name, description, category, price, material, brand, size, color, stock\_quantity, and average\_rating.

**vendor (\*vendor\_id, name, phone\_number, email, location, vendor\_rating, vendor\_join\_date, vendor\_status, total\_sales)**

The Vendor schema provides details about vendors, including vendor\_id (primary key), name, phone\_number, email, location, vendor\_rating, vendor\_join\_date, vendor\_status, and total\_sales.

**revenue (\*revenue\_id, commission\_rate, discount\_provided, tax\_deducted, net\_revenue, FK (vendor\_id))**

The Revenue schema tracks earnings, with attributes such as revenue\_id (primary key), commission\_rate, discount\_provided, tax\_deducted, and net\_revenue. It includes the foreign key vendor\_id to link revenue to specific vendors.

**offers (FK (product\_id), FK (vendor\_id))**

Offers schema establishes a many-to-many relationship between products and vendors through the foreign key's product\_id and vendor\_id, enabling the marketplace to track which vendors offer specific products.

## 3.2 MySQL IMPLEMENTATION

### Schema and table creation

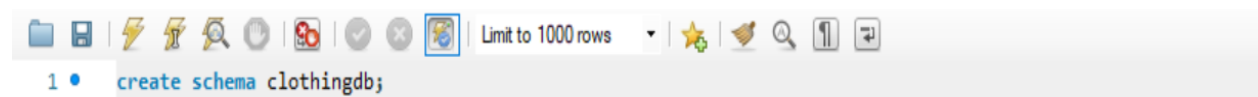


Figure 2:Schema creation

Figure 2, gives us the SQL command to create a schema, where schema name is clothingdb.

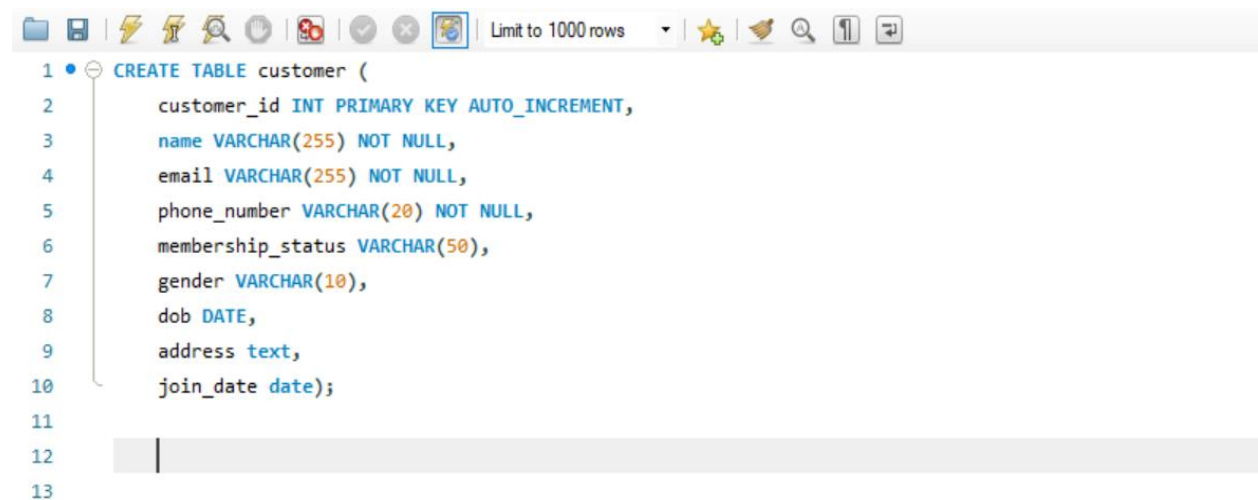


Figure 3:Table creation

Figure 3, gives the the SQL command to create a table. In the figure I am creating a table named customer with the attributes customer\_id which is a primary key and auto increment is also set, along with other attributes such as name, email, phone\_number, membership\_status, gender, dob, address and join\_date.

## Tables and Columns

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
cost_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
packaging_cost	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
advertising_cost	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
delivery_failure_cost	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
reattempt_cost	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
fraud_cost	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
order_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 4:Table “cost”

Figure 4, shows us the table “cost” with columns having cost\_id as a unique key and auto increment feature which also has a foreign key order\_id.

Foreign Key Name	Referenced Table
cost_ibfk_1	`clothingdb`.`orders`

Column	Referenced Column
<input type="checkbox"/> cost_id	
<input type="checkbox"/> packaging_cost	
<input type="checkbox"/> advertising_cost	
<input type="checkbox"/> delivery_failure...	
<input type="checkbox"/> reattempt_cost	
<input type="checkbox"/> fraud_cost	
<input checked="" type="checkbox"/> order_id	order_id

Foreign Key Options

On Update: RESTRICT

On Delete: RESTRICT

☐ Skip in SQL generation

Foreign Key Comment

Figure 5:Foreign key in cost table

Figure 5, shows the foreign key named cost\_ibfk\_1 is referencing to the column order\_id of table orders in schema clothingdb.



Table Name:  Schema: **clothingdb**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
coupon_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
code	VARCHAR(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
discount_rate	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
expiry_date	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 6:Table “coupon”

Figure 6, shows us the columns available in table “coupon” where coupon\_id is the primary key and also set to autoincrement, columns code,discount\_rate and expiry\_date are set to not null as these are the mandatory fields in the table coupon.

Table Name:  Schema: **clothingdb**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
customer_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
email	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
phone_number	VARCHAR(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
membership_status	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
gender	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
dob	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
address	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
join_date	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 7:Table “customer”

Figure 7, gives us the details on table “customer” ,it has a primary key customer\_id,where name,email and phonenumber are the mandatory fields .Email should be unique.

Table Name:  Schema: **clothingdb**

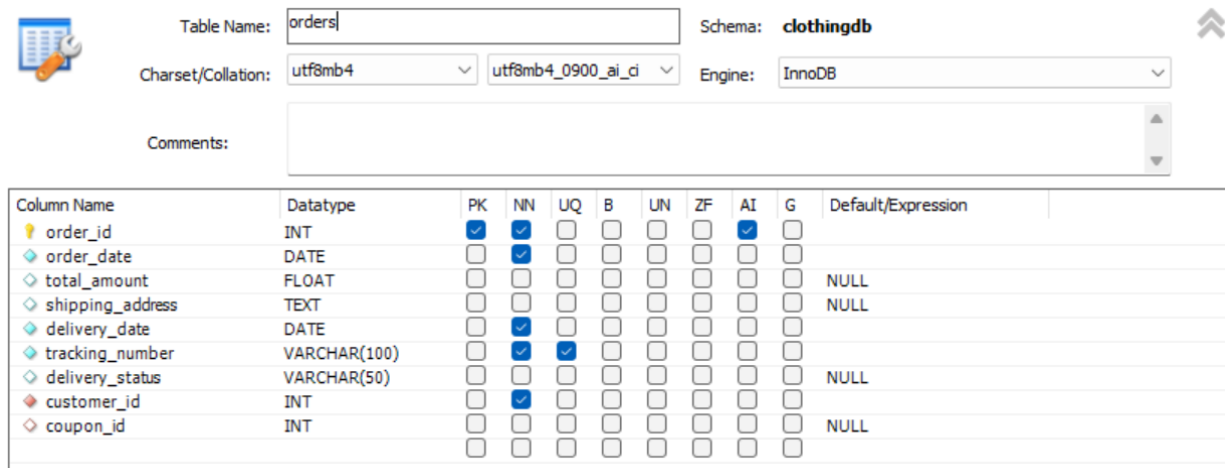
Charset/Collation:   Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
product_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
vendor_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 8:Table “offers”

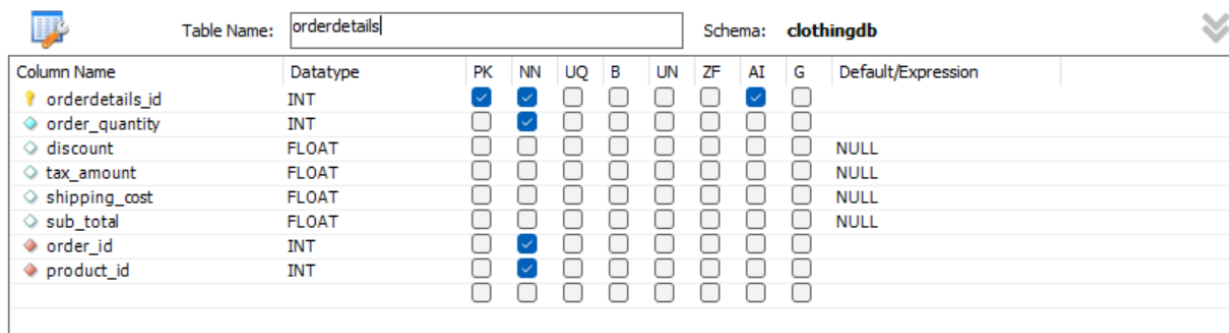
Figure 8, is the table “offers” which has the foreign keys product\_id and vendor\_id which is referenced with the product\_id of product table and vendor\_id of vendor table.



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
order_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
order_date	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
total_amount	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
shipping_address	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
delivery_date	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
tracking_number	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
delivery_status	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
customer_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
coupon_id	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 9:Table “orders”

Figure 9, gives us the details of table “orders” where order\_id is the primary key and set to auto increment, where delivery\_date and tracking\_number is made mandatory and tracking\_number as unique.



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
orderdetails_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
order_quantity	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
discount	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
tax_amount	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
shipping_cost	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
sub_total	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
order_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
product_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 10:Table “orderdetails”

Figure 10,gives us the details of table “orderdetails” where orderdetails\_id is the primary key with auto increment set to yes and order\_quantity is set to not null and having the foreign keys order\_id and product\_id.

Table Name: 

payments

Schema: **clothingdb**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
<div></div> payment_id	INT	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	
<div></div> payment_method	VARCHAR(50)	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	
<div></div> payment_date	DATE	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	
<div></div> payment_status	VARCHAR(50)	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	
<div></div> order_id	INT	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	

Figure 11:Table “payments”

Figure 11, gives us the details on table “payments” where payment\_id is the primary key with autoincrement and all other attributes are made mandatory with order\_id as the foreign key.




Table Name:

products

Schema:

clothingdb












Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 product_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
 name	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 description	TEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 category	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 material	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 brand	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 size	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 color	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 stock_quantity	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 average_rating	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 12:Table “products”

Figure 12, gives us the details on table “products” where product\_id is the primary key and name,description,category,price and stock\_quantity are made as not null.







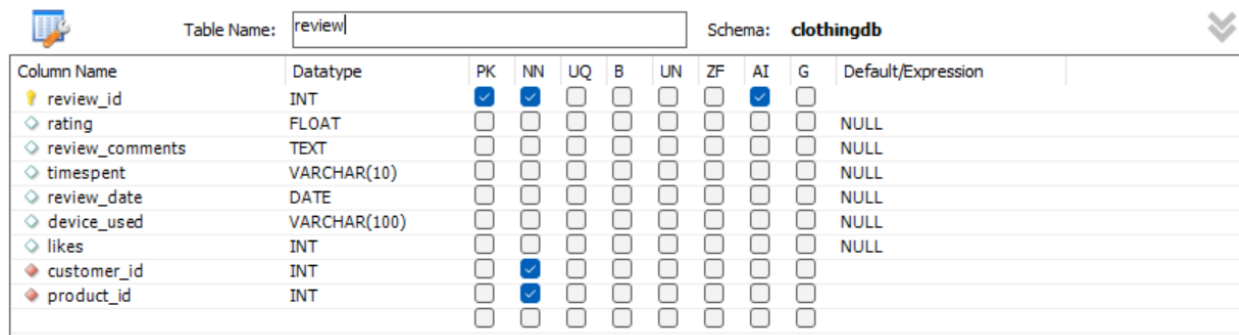
Table Name: <div>revenue</div>		Schema: <b>clothingdb</b>								
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 revenue_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
 commission_rate	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 discount_provided	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 tax_deducted	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 net_revenue	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 vendor_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 13:Table “revenue”

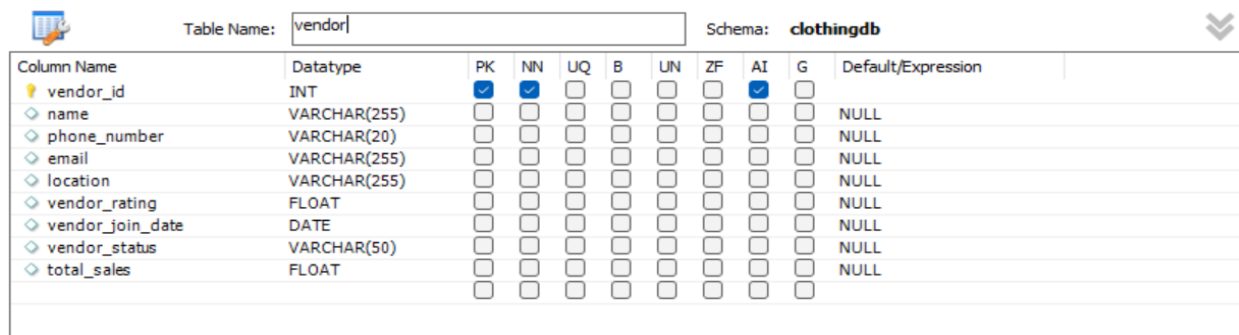
Figure 13, gives us the details on table “revenue” which has the revenue\_id as the primary key and which is set to auto increment. It also has the vendor\_id as the foreign key.



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
review_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
rating	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
review_comments	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
timespent	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
review_date	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
device_used	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
likes	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
customer_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
product_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 14:Table “review”

Figure 14, gives us the details on table “review” where review\_id is the primary key and auto increment is set to yes. It also has customer\_id and product\_id as foreign keys.



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
vendor_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
phone_number	VARCHAR(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
email	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
location	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
vendor_rating	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
vendor_join_date	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
vendor_status	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
total_sales	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Figure 15:Table “vendor”

Figure 15, gives us the details on the table “vendor” where vendor\_id is the primary key and set to auto increment.

### 3.3 DATA GENERATION

For data generation, I have used **faker()** library to generate the data for the online clothing marketplace. While generating data, I have generated around 1000 customers, while there are 1300 products, 1100 vendors, 200 revenue details, 500 orders, 1000 orderdetails, 500 reviews by the customer, 600 payments, around 100 cost details for the orders, and 100 different coupons.

```

for _ in range(NUM_ORDERS):
    order = {
        "OrderDate": fake.date_this_year().strftime("%Y-%m-%d"),
        "TotalAmount": round(random.uniform(10.0, 25000.0), 2),
        "ShippingAddress": fake.address(),
        "DeliveryDate": fake.date_this_year().strftime("%Y-%m-%d"),
        "TrackingNumber": fake.uuid4(),
        "DeliveryStatus": random.choice(["Shipped", "Pending", "Delivered"]),
        "CustomerId": random.randint(1, 1000),
        "CouponId": random.randint(1, 50),
    }
    table_statements["Orders"].append(
        f"INSERT INTO orders (order_date, total_amount, shipping_address, delivery_date, tracking_number, delivery_status, customer_id, coupon_id)"
        f"VALUES ( '{order['OrderDate']}', {order['TotalAmount']}, '{order['ShippingAddress']}', "
        f"'{order['DeliveryDate']}', '{order['TrackingNumber']}', '{order['DeliveryStatus']}', {order['CustomerId']}, {order['CouponId']}');"
    )

```

Figure 16:Code snippet for data generation

Figure 16, gives the code snippet used for generating fake data to “orders” table using python.

order_id	order_date	total_amount	shipping_address	delivery_date	tracking_number	delivery_status	customer_id	coupon_id
1	2025-01-11	21457.3	716 Walker Mea...	2025-01-02	IRSC6QIVO7	Shipped	818	37
2	2025-01-23	3486.79	512 Stevens Exp...	2025-01-20	EDS3QKKI74	Delivered	940	16
3	2025-01-19	14748	USCGC Jones FP...	2025-01-07	Y8B62C58C7	Delivered	545	13
4	2025-01-15	21126.4	5078 John Trail ...	2025-01-21	95JEVFT8I3	Delivered	822	72
5	2025-01-17	10420.9	46583 Kara Terr...	2025-01-13	O97EUOCPTO	Pending	819	88
6	2025-01-06	796.91	236 Nelson Inlet ...	2025-01-03	ZUSC3DL6KE	Delivered	7	13
7	2025-01-03	7221.58	Unit 0030 Box 91...	2025-01-23	8W30FL2FBJ	Pending	485	17
8	2025-01-18	3693.91	4817 Dorothy Es...	2025-01-18	GMG6M8PPM1	Delivered	570	21
9	2025-01-15	15995.6	348 Dakota Mou...	2025-01-13	1KOS6V67FS	Pending	408	6
10	2025-01-11	11840	724 Miller Spring...	2025-01-01	LPNUTBJXPE	Delivered	154	25
11	2025-01-10	7541.35	4847 Justin Point...	2025-01-06	I8WN5ANLQR	Shipped	674	39
12	2025-01-16	882.27	142 Samuel Curv...	2025-01-03	UG6P429MAM	Delivered	153	92
13	2025-01-07	2856.97	792 Lori Causew...	2025-01-04	5BH86TSIZO	Delivered	126	13
14	2025-01-19	15979.9	08406 Melissa R...	2025-01-10	LHY91GRQZ3	Pending	412	22
15	2025-01-07	12524.9	67238 Valentine ...	2025-01-14	4EBPFJ2CT7	Shipped	725	28
16	2025-01-01	11659.8	17908 Flores Ke...	2025-01-09	PL5HTIP2I6	Shipped	228	46
17	2025-01-03	1636.81	8704 Burgess Pin...	2025-01-06	LYB3CKEZ4S	Pending	930	95
18	2025-01-23	19160.7	2951 Bright Cam...	2025-01-20	87KNMWXXHI	Pending	445	59
19	2025-01-15	11837.2	Unit 3235 Box 03...	2025-01-16	MBVYJEPBNU	Pending	956	57
20	2025-01-14	21524.9	090 Troy Village ...	2025-01-21	YAAVWRA82A	Pending	748	10

Figure 17:Table “Orders” with the data

Figure 17, shows the data that has been generated for “orders” table ,which has the columns such as order\_id, order\_date, total\_amount, shipping\_address, delivery\_date, tracking\_number, deliver\_status, and foreign keys customer\_id and coupon\_id.

### 3.4 RE-GENERATED ER DIAGRAM

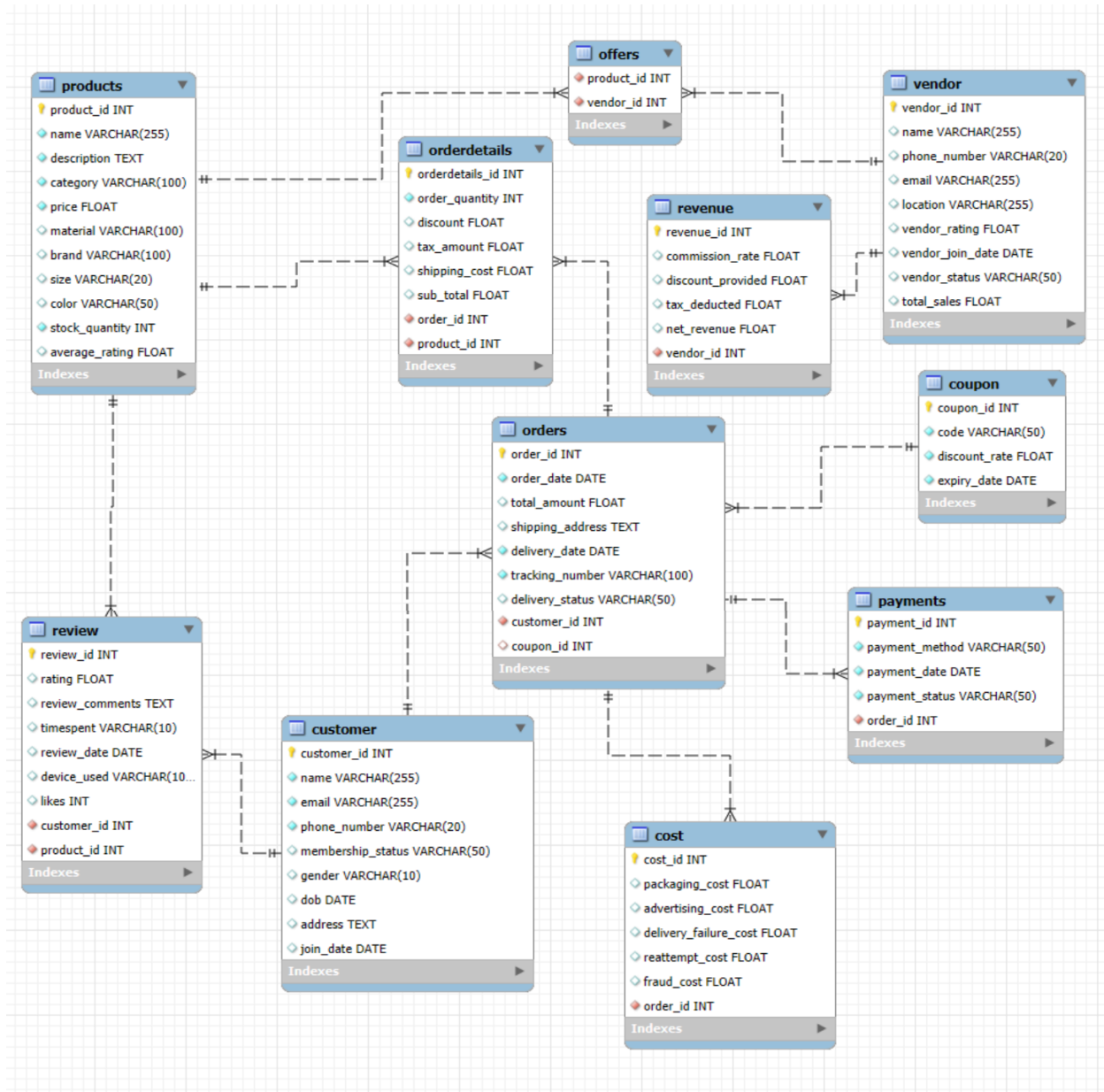


Figure 18:Regenerated ER Diagram

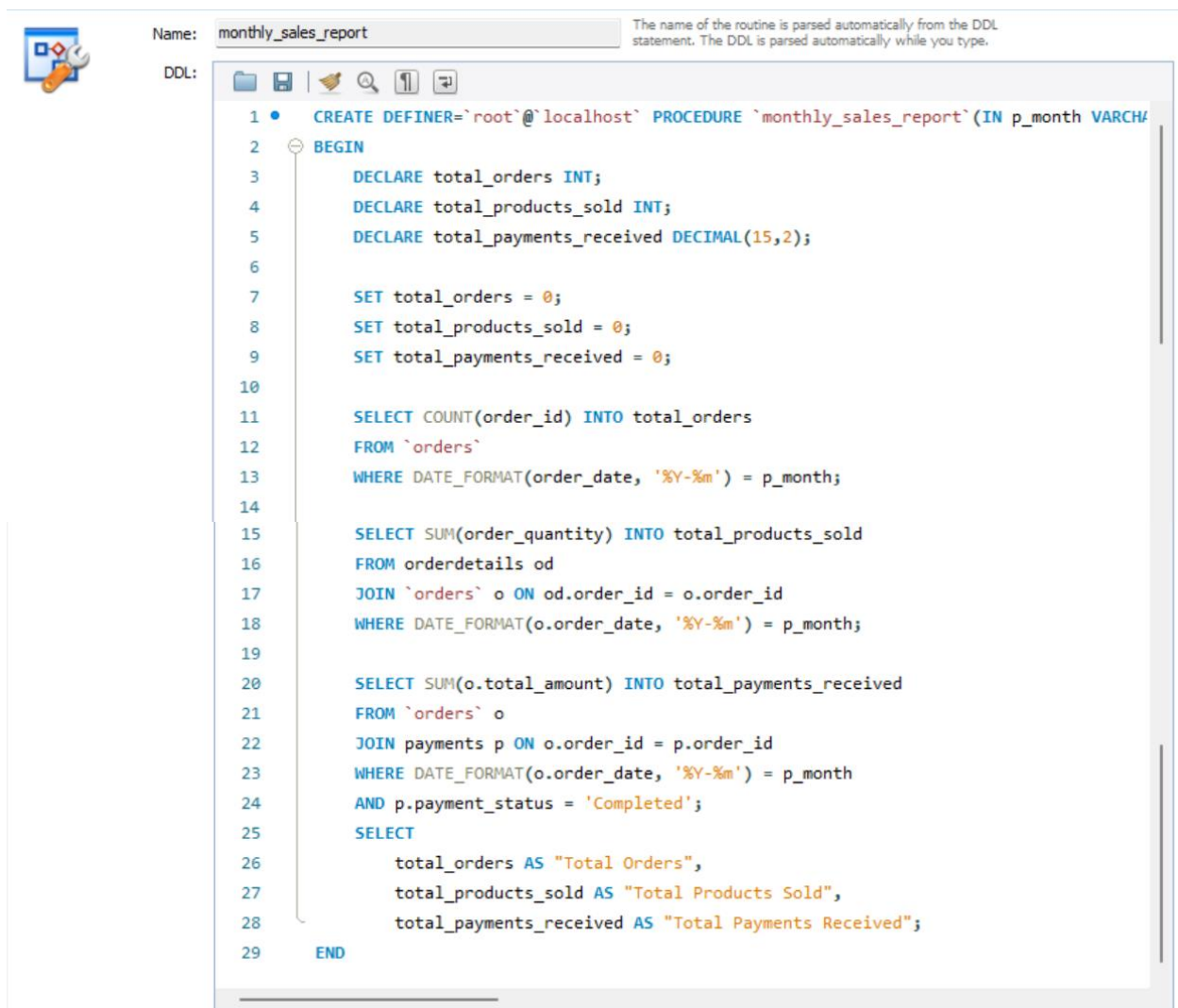
In figure 18, we can see the regenerated ER diagram. Tables like orders, orderdetails, review, payments, cost, revenue have their corresponding foreign keys as shown in the figure. Entity “offer” is the new entity which has been generated due to the many-to-many relationship between the entity “product” and the entity “vendor”. This entity has the attributes which are the “product\_id” and “vendor\_id” which are the primary keys of the corresponding entities.

## CHAPTER FOUR: SQL QUERY GENERATION

In this chapter, I will be discussing on having a trigger and a stored procedure in my database along with few queries which uses few SQL operations.

### 4.1 STORED PROCEDURE

A stored procedure is a set of SQL queries that are pre-written and stored in a database. It is executed by calling its name, it helps in repetitive operations, thus improving the performance database operations.



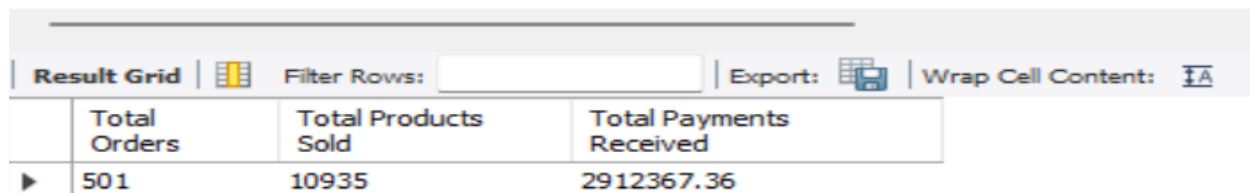
The screenshot shows a database IDE interface. At the top, there's a 'Name:' field containing 'monthly\_sales\_report'. Below it, the 'DDL:' section contains the SQL code for creating the stored procedure. The code is as follows:

```
1 • CREATE DEFINER=`root`@`localhost` PROCEDURE `monthly_sales_report`(IN p_month VARCHAR(10))
2 BEGIN
3     DECLARE total_orders INT;
4     DECLARE total_products_sold INT;
5     DECLARE total_payments_received DECIMAL(15,2);
6
7     SET total_orders = 0;
8     SET total_products_sold = 0;
9     SET total_payments_received = 0;
10
11     SELECT COUNT(order_id) INTO total_orders
12     FROM `orders`
13     WHERE DATE_FORMAT(order_date, '%Y-%m') = p_month;
14
15     SELECT SUM(order_quantity) INTO total_products_sold
16     FROM orderdetails od
17     JOIN `orders` o ON od.order_id = o.order_id
18     WHERE DATE_FORMAT(o.order_date, '%Y-%m') = p_month;
19
20     SELECT SUM(o.total_amount) INTO total_payments_received
21     FROM `orders` o
22     JOIN payments p ON o.order_id = p.order_id
23     WHERE DATE_FORMAT(o.order_date, '%Y-%m') = p_month
24     AND p.payment_status = 'Completed';
25     SELECT
26         total_orders AS "Total Orders",
27         total_products_sold AS "Total Products Sold",
28         total_payments_received AS "Total Payments Received";
29 END
```

Figure 19:Stored Procedure for calculating monthly report

Figure 19, gives us stored procedure named “**monthly\_sales\_report**” to generate a monthly report for the online clothing marketplace, including: Total number of **orders** for the month, total number of **products** sold in the month and total **payments** received for completed orders in the given month. The procedure accepts a **month parameter** (YYYY-MM format) and produces a report that contains the output columns such as **Total Orders**, **Total Products Sold** and **Total Payments Received**.

```
1 • use clothingdb;  
2 • CALL monthly_sales_report('2025-01');
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid has three columns: 'Total Orders', 'Total Products Sold', and 'Total Payments Received'. The first row of data shows 501 Total Orders, 10935 Total Products Sold, and 2912367.36 Total Payments Received. The interface also includes a 'Filter Rows' field, an 'Export' button, and a 'Wrap Cell Content' checkbox.

	Total Orders	Total Products Sold	Total Payments Received
▶	501	10935	2912367.36

Figure 20:Output of stored procedure

In Figure 20, I am using the database clothingdb and calling the procedure monthly\_sales\_report with the parameter being month 01 of the year 2025 in the format (2025-01).This gives me the report for this particular month as shown in the figure , where we have 501 Total Orders,10935 Total Products Sold and Total Payments Received is 2912367.36

This stored procedure will help in calculating the performance of this marketplace during any given month, that can be used for several purposes:

**Performance Monitoring:** Total orders, total products sold, and amount of money received depict the performance of the clothing marketplace overtime.

**Forecasting Revenue:** The company can make better revenue forecasts regarding the future, by identifying the trends in clothing industry.

**Customer Insights:** It also provides insights into the buying behavior of the customer, which helps in making necessary changes in the marketing strategy.



## 4.2 TRIGGER

A trigger in SQL is a special kind of stored procedure that automatically executes or "fires" when certain events occur on a particular table or view in a database.

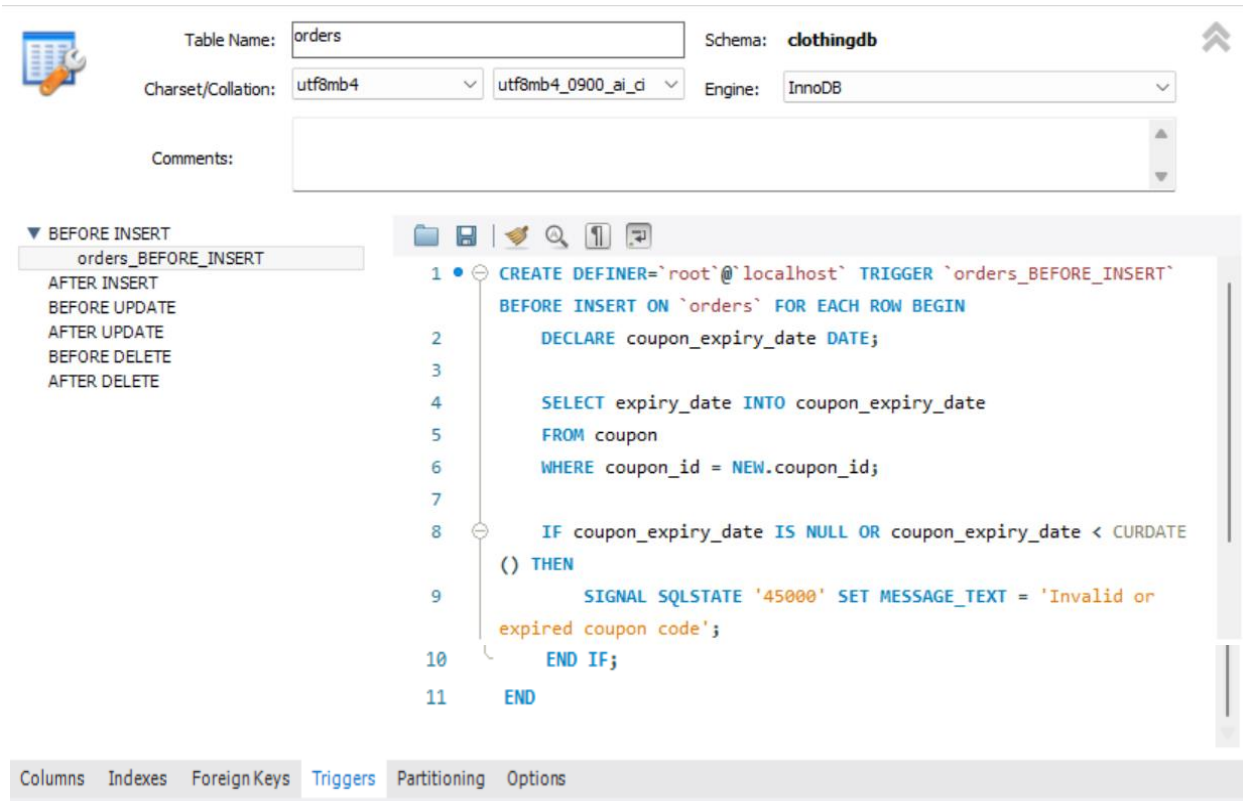


Figure 21: Trigger applied on “orders” table before insert statement

### Trigger: Validate Coupon Code Before Order Insertion

Figure 21, **BEFORE INSERT** trigger, ensures that when a new order is placed, the coupon code provided by the customer is valid. It checks if the coupon exists in the coupon table, and whether the coupon has expired or is still active. If the coupon is invalid or expired, the trigger will raise an error and prevent the order from being inserted.

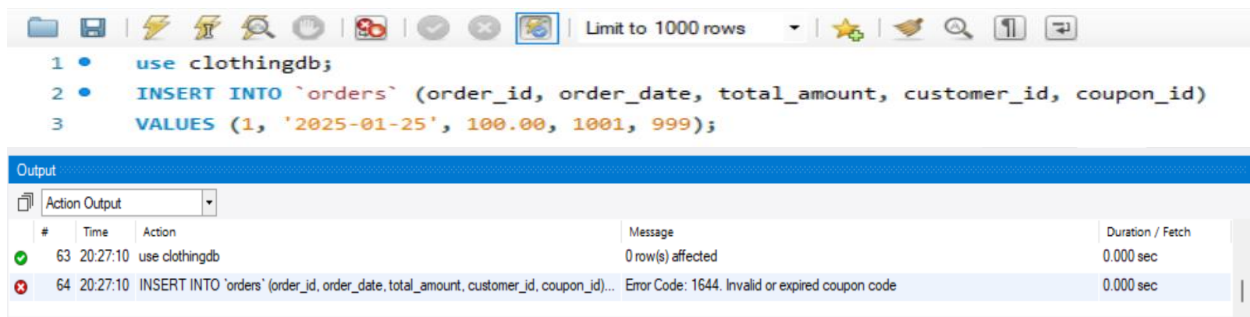


Figure 22:Trigger Output

In Figure 22, I am trying to insert an order with coupon\_id 999, In this case, the order insertion will fail with the error message: "Invalid or expired coupon code" as there is no coupon with id 999.

This trigger helps in many ways as it ensures only valid coupons are applied. Prevents expired or non-existent coupons from being used. Enhances the customer experience by providing immediate feedback if the coupon is invalid or expired.

## 4.3 QUERIES

### Query 1: Identify customers who have placed orders using a coupon



Figure 23:Query-1

Result Grid			
Filter Rows: <input type="text"/>			
Export: <input type="button" value="Export"/>			
Wrap Cell Content: <input type="button" value="Wrap"/>			
Fetch rows: <input type="button" value="Fetch"/>			
customer_id	customer_name	total_orders_with_coupon	
502	John Bennett	4	
92	Sean Lang	3	
112	Jacob West	3	
748	Jennifer Gonzalez	3	
386	Howard McDonald	3	
266	Jose Macdonald	3	
432	Kelly Chang	3	
154	Abigail Jennings	3	
396	Christina Robertson	3	
228	David Perez	2	

Figure 24:Query-1 Output

Figure 23, The query which identifies the **customers** who frequently use **coupons** while ordering the items from this marketplace. This query helps in analysing the customer behaviour regarding coupon usage. **JOIN**: Combines customer and orders to associate customer details with their orders. **WHERE**: Filters for orders where a coupon was applied (o.coupon\_id IS NOT NULL). **GROUP BY**: Groups data by customer to count the total orders with coupons. **ORDER BY**: Sorts the results in descending order of coupon usage.

Figure 24, gives the output of the query which gives the customer\_id, customer\_name and total\_orders\_with\_coupon. This query helps to identify the customers who are more responsive towards discount and we can plan the campaign accordingly.

## Query 2: Lists top-rated products based on customer reviews

```

1 • use clothingdb;
2 • SELECT
3     p.product_id,
4     p.name AS product_name,
5     AVG(r.rating) AS average_rating
6 FROM
7     products p
8 INNER JOIN review r ON p.product_id = r.product_id
9 GROUP BY
10    p.product_id, p.name
11 HAVING
12    AVG(r.rating) > 4.5
13 ORDER BY
14    average_rating DESC limit 10;

```

Figure 25:Query-2

Result Grid			
Filter Rows:			
Export: Wrap Cell Content: Fetch rows:			
product_id	product_name	average_rating	
203	listen	5	
238	certain	5	
73	her	5	
293	by	5	
259	lot	5	
93	matter	5	
49	however	5	
182	story	5	
284	throw	5	
62	have	5	

Figure 26:Query-2 Output

Figure 25, This query identifies **products** with high **average ratings** (greater than 4.5), helping customers with written proofs by other customers. **JOIN**: Links products and review to analyze product feedback. **GROUP BY**: Groups reviews by product to calculate average ratings. **HAVING**: Filters products with an average rating above 4.5. **ORDER BY**: Sorts products by their average rating in descending order.

Figure 26, gives the output of the query with the product\_id, Product\_name and average rating of each product which is greater than 4.5. This helps to identify products which has higher rating while helping vendors focus on maintaining or improving the quality of these items.

### Query 3: Calculate Total Revenue Earned by Each Vendor

```

1 • use clothingdb;
2 • SELECT
3     v.vendor_id,
4     v.name AS vendor_name,
5     ROUND(SUM(r.net_revenue), 2) AS total_revenue
6 FROM
7     vendor v
8 INNER JOIN revenue r ON v.vendor_id = r.vendor_id
9 GROUP BY
10    v.vendor_id, v.name
11 ORDER BY
12    total_revenue DESC limit 10;

```

Figure 27:Query-3

Result Grid			
Filter Rows:		Export:	Wrap Cell Content: Fetch rows:
vendor_id	vendor_name	total_revenue	
99	Haynes-Davis	327401.43	
46	Schwartz, Beck and Mills	305995.16	
11	Smith-Wood	289539.69	
98	Brooks, McBride and Russell	275891.92	
42	Miller and Sons	267957.62	
23	Henderson-Maddox	267361.04	
92	Johnson, Singh and Phillips	258108.3	
30	Wu, Taylor and Dominguez	240749.59	
41	Gonzalez, Bailey and Figueroa	237538.82	
14	Davis, Randolph and Shaffer	229981.47	

Figure 28:Query-3 Output

Figure 27, This query calculates the total revenue which is earned by each vendor. **INNER JOIN**: Links vendor and revenue to calculate revenue per each vendor. **GROUP BY**: Groups data by vendor to calculate the net revenue. **ORDER BY**: Sorts vendors by total revenue in descending order.

Figure 28, gives the output of the query which has the vendor\_id, vendor\_name and total\_revenue. This query can help in identifying the Vendors with high revenues as top performers, while the underperforming vendors can be supported with more manufacturing strategies.

#### Query 4: Find Vendors with Products in Multiple Categories

```

1 • use clothingdb;
2 • SELECT
3     v.vendor_id,
4     v.name AS vendor_name,
5     COUNT(DISTINCT p.category) AS distinct_categories
6 FROM
7     vendor v
8 INNER JOIN offers o ON v.vendor_id = o.vendor_id
9 INNER JOIN products p ON o.product_id = p.product_id
10 GROUP BY
11     v.vendor_id, v.name
12 HAVING
13     COUNT(DISTINCT p.category) > 1 limit 10;

```

Figure 29:Query-4

Result Grid			
Filter Rows:			
Export:			
Wrap Cell Content:			
Fetch rows:			
vendor_id	vendor_name	distinct_categories	
6	Ward PLC	2	
10	Hernandez Ltd	2	
18	Woods, Anderson and Rodgers	3	
19	Smith, Hebert and Nguyen	2	
20	Harris LLC	2	
23	Henderson-Maddox	3	
24	Chase, Smith and Long	2	
51	Horton-Bennett	2	
54	Davis Ltd	2	
55	Huff, Watkins and Mathews	3	

Figure 30:Query-4 Output

Figure 29, This query identifies vendors who sell products across multiple categories. **JOIN:** Combines vendor, offers, and products to analyze product categories owned by each vendor. **GROUP BY:** Groups data by vendor to count unique product categories. **HAVING:** Filters vendors offering products in more than one category.

Figure 30, gives the output which has the details of vendor\_id,vendor\_name and the distinct\_categories which they include.This query identifies vendors with a broader product range which will help in calculating the revenue.

### Query-5 Identify the Top-Selling Product in Each Category

```

1 • use clothingdb;
2 • WITH RankedProducts AS (
3     SELECT
4         p.category,
5         p.name AS product_name,
6         SUM(od.order_quantity) AS total_quantity_sold,
7         RANK() OVER (PARTITION BY p.category ORDER BY SUM(od.order_quantity) DESC) AS category_rank
8     FROM
9         products p
10    INNER JOIN orderdetails od ON p.product_id = od.product_id
11    GROUP BY
12        p.category, p.product_id, p.name)
13 SELECT
14     category,product_name,total_quantity_sold,category_rank FROM RankedProducts WHERE category_rank = 1;

```

Figure 31:Query-5

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
category	product_name	total_quantity_sold	category_rank
Men	defense	142	1
Kids	affect	139	1
Women	rich	96	1

Figure 32:Query-5 Output

Figure 31, This query finds the best-selling product in each category based on sales quantity. **JOIN:** Links products and orderdetails to analyze sales data. **GROUP BY:** Groups data by product and category to calculate sales. **Window Function:** Ranks products within each category by total quantity sold. **HAVING:** Filters for the top product in each category (category\_rank = 1).

Figure 32, Gives the output of the query which has the category of products, product\_name, total\_quantity\_sold and the category\_rank of the products. This query helps the vendors to focus on the most popular products in each category so that they can manufacture such items more to increase their net revenue.

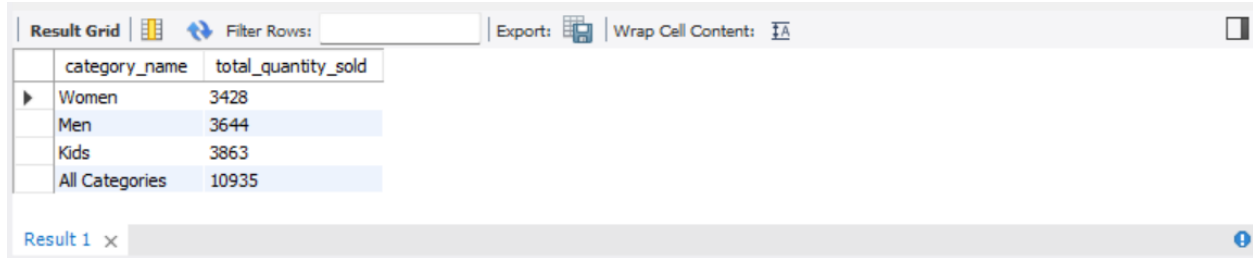
### Query-6 Compare Total Sales Across All Categories

```

1 • use clothingdb;
2 • SELECT
3     p.category AS category_name,
4     SUM(od.order_quantity) AS total_quantity_sold FROM products p
5
6     INNER JOIN orderdetails od ON p.product_id = od.product_id GROUP BY p.category
7 UNION
8 SELECT 'All Categories' AS category_name, SUM(od.order_quantity) AS total_quantity_sold FROM orderdetails od
9     INNER JOIN products p ON od.product_id = p.product_id;
10

```

Figure 33:Query-6



The screenshot shows a database interface with a 'Result Grid' tab. It includes a 'Filter Rows' search bar, 'Export' and 'Wrap Cell Content' buttons, and a table with two columns: 'category\_name' and 'total\_quantity\_sold'. The table contains four rows: 'Women' (3428), 'Men' (3644), 'Kids' (3863), and 'All Categories' (10935). A 'Result 1' tab is visible at the bottom.

category_name	total_quantity_sold
Women	3428
Men	3644
Kids	3863
All Categories	10935

Figure 34-Query-6 Output

Figure 33, This query compares individual category sales with overall platform sales. **JOIN:** Combines products and orderdetails to calculate sales per each category. **GROUP BY:** Groups data by category for the sum of individual category. **Set Operation (UNION):** Combines the results of category totals and overall totals into a single output.

Figure 34, Gives the output which has the category\_name and total\_quantity\_sold of each category and also the sum of all the categories. This query helps in providing a complete view of sales distribution, helping the vendors in knowing the high performing category to manufacture more such categories.



## CHAPTER FIVE: MODEL EVALUATION

In this chapter, I will be discussing on the performance of the database based on larger size of data. ACID properties of the database along with the CAP theorem and discuss on a case study of an analytical data warehouse.

```
1 • use clothingdb;
2 • WITH RankedProducts AS (
3     SELECT
4         p.category,
5         p.name AS product_name,
6         SUM(od.order_quantity) AS total_quantity_sold,
7         RANK() OVER (PARTITION BY p.category ORDER BY SUM(od.order_quantity) DESC) AS category_rank
8     FROM
9         products p
10    INNER JOIN orderdetails od ON p.product_id = od.product_id
11    GROUP BY
12        p.category, p.product_id, p.name
13    SELECT
14        category, product_name, total_quantity_sold, category_rank FROM RankedProducts WHERE category_rank = 1;
```

Figure 35: Analytical SQL query

Figure 35, is the analytical SQL query which gives the best-selling product in each category based on sales quantity. An INNER JOIN is performed between products and orderdetails to link products with their sales data. SUM(od.order\_quantity) calculates the total quantity sold for each product. Window Function – RANK(): assigns a rank to each product within its category based on the descending order of total sales. Grouping: The GROUP BY clause ensures the aggregation is calculated at the product level.

### 5.1 PERFORMANCE ANALYSIS

This query takes around 0.015 seconds to complete and does a great job for a dataset of 1300 products, 500 orders, and 1000 order details. Since we have used INNER join, which provides us with the products and related order details, aggregation giving the sum of amounts, and window function ranking, it will show that the query is optimized. The GROUP BY clause aggregates product-level data efficiently, and the end result set is kept compact and to the point by filtering by (category\_rank = 1).

## 5.2 QUERY OPTIMIZATION

For instance, if the dataset grows ten times bigger, that would mean 13,000 products, 5,000 orders, and 10,000 order details, which may hike up the execution time of this query to about 0.15 seconds. In this respect, indexing for the `product_id`, `category`, or other such frequently used columns of the Product table, along with `quantity_sold` of the order details table for rapid search of details, will be in place. Proper **indexing** avoids degradation of performance while scaling up the database by keeping the joins, grouping, and partitioning efficient.

## 5.3 ACID PROPERTIES

The acronym for Atomicity, Consistency, Isolation, and Durability is ACID. It is a collection of characteristics that ensures database transactions are processed reliably. These characteristics guarantee that Database transactions are handled with dependability even when there are mistakes, unanticipated events, or system malfunctions (Mapanga & Kadebu, 2013).

In RDBMS like MySQL, ACID properties ensure that database transactions are processed reliably and the integrity is maintained even if system fails. Since I have used MySQL database we will breakdown on how ACID properties works in this clothing marketplace.

Figure 36, gives us the example of a guaranteed ACID property. Here, `redeem_coupon_and_place_order` is a stored procedure which redeems a coupon for a customer, applies its discount to an order, and records the transaction.

**Atomicity:** This guarantees that a transaction is handled as a single, unbreakable work unit. If any part of the transaction fails then the entire transaction is rolled back. (Mapanga & Kadebu, 2013). In the `redeem_coupon_and_place_order` stored procedure (Figure 36), when placing an order, the will first validate the coupon, gives the product price, calculates the total amount, and inserts data into both the orders and orderdetails tables. If any of these steps fails (for example, if the coupon is invalid or the product ID is incorrect), the entire transaction is rolled back using the ROLLBACK statement. This makes the system's state remain the same even if an error occurs.

```

1 • CREATE DEFINER='root'@'localhost' PROCEDURE `redeem_coupon_and_place_order`(
2     IN p_customer_id INT,
3     IN p_coupon_id INT,
4     IN p_product_id INT,
5     IN p_order_quantity INT,
6     OUT p_final_amount DECIMAL(15, 2)
7 )
8 BEGIN
9     DECLARE v_coupon_discount DECIMAL(5, 2);
10    DECLARE v_product_price DECIMAL(10, 2);
11    DECLARE v_subtotal DECIMAL(15, 2);
12    DECLARE v_total_amount DECIMAL(15, 2);
13
14    START TRANSACTION;
15
16    SELECT discount_rate
17    INTO v_coupon_discount
18    FROM coupon
19    WHERE coupon_id = p_coupon_id
20        AND expiry_date >= CURDATE();
21
22    IF v_coupon_discount IS NULL THEN
23        ROLLBACK;
24        SIGNAL SQLSTATE '45000'
25        SET MESSAGE_TEXT = 'Invalid or expired coupon';
26    END IF;
27
28    SELECT price
29    INTO v_product_price
30    FROM products
31    WHERE product_id = p_product_id;
32
33    IF v_product_price IS NULL THEN
34        ROLLBACK;
35        SIGNAL SQLSTATE '45000'
36        SET MESSAGE_TEXT = 'Invalid product ID';
37    END IF;
38
39    SET v_subtotal = v_product_price * p_order_quantity;
40    SET v_total_amount = v_subtotal - (v_subtotal * v_coupon_discount / 100);
41
42    INSERT INTO `orders` (order_date, total_amount, customer_id, coupon_id)
43    VALUES (NOW(), v_total_amount, p_customer_id, p_coupon_id);
44
45    SET @order_id = LAST_INSERT_ID();
46
47    INSERT INTO orderdetails (order_id, product_id, order_quantity, sub_total)
48    VALUES (@order_id, p_product_id, p_order_quantity, v_subtotal);
49
50    COMMIT;
51
52    SET p_final_amount = v_total_amount;
53 END

```

Figure 36: Stored Procedure with ACID properties

**Consistency:** Consistency refers to the ability for all nodes to see the same data at the same time. Prior to and following the transaction, the database must adhere to a set of predetermined integrity requirements (Ajayi Rosiji, 2015). The `redeem_coupon_and_place_order` procedure maintain consistency by confirming that there are valid coupons and product prices. For example, the procedure checks if the coupon is valid by verifying its expiry date against the current date. If the coupon is expired or invalid, the transaction is rolled back and an error message is raised, so that there is no invalid data entered into the database. Similarly, the procedure ensures that the product exists by checking its price in the products table. Thus, consistency gives us the database which is always in a valid state after the transaction.

**Isolation:** Isolation mainly deals with concurrency control and visibility of effects of concurrently executing transactions to each other (Syed Akbar Mehdi, 2017). In the `redeem_coupon_and_place_order` procedure, the use of `START TRANSACTION` and `COMMIT` ensures that the operations on orders and orderdetails are isolated. If another customer tries to redeem the same coupon, The transaction takes place without any delay. If multiple customers are placing orders at the same time, their individual transactions will not affect one another.

**Durability:** Durability guarantees that once a transaction is committed, its effects will persist, even in the case of a system failure (e.g., power loss or hardware failure) (Anju Santosh Yedatkar, 2024). After the `redeem_coupon_and_place_order` procedure successfully inserts data into the orders and orderdetails tables, the transaction is committed using the `COMMIT` statement. This guarantees that the changes made by the transaction will persist in the database. If the system crashes immediately after committing the transaction, the changes will still be available when the system recovers.

## 5.4 CAP THEOREM

Consistency, Availability, and Partition Tolerance are three essential distributed system characteristics that are represented by CAP. Theorem shows that all three of these features cannot be achieved together in a distributed system.

According to CAP theorem, **consistency** refers to the ability for all nodes to see the same data at the same time. **Availability** refers to the guarantee that all reads and writes are always successful and **partition tolerance** refers to the guarantee that the CAP properties of the system are

maintained even in the event of network that prevents some machines or computers to communicate with one another (Anju Santosh Yedatkar, 2024).

The MySQL database in our online clothing marketplace focuses on **Consistency** and **Availability** over **Partition tolerance** to ensure customer satisfaction.

**Consistency** is maintained in this database as customers expect product availability, pricing, and promotions. If the data displayed to customers is inconsistent, such as showing products in stock that are already sold out, then the customers trust in the platform could be damaged. Here, consistency is achieved through MySQL's transactional integrity, which orders and payments are synchronized at all times.

**Availability** ensures that the system responds to customer actions quickly, especially during peak traffic periods. For example, when customers browse products, apply discount coupons, or complete a payment, the system processes these requests quickly to provide a seamless shopping experience. MySQL ensures availability by handling multiple queries concurrently.

**Partition Tolerance** is less critical in my MySQL-based online clothing marketplace because the database operates as a centralized system rather than a distributed one. For example, in this marketplace, when a customer places an order, the system directly interacts with the MySQL database to update the inventory, record the order, and apply the payment. Since this all happens within a single, centralized database, there is no need to account for scenarios where nodes might fail to synchronize due to a partition. Therefore, there is rare need for handling network partition scenarios.

By focusing on **Consistency** and **Availability**, the marketplace ensures reliable inventory updates, smooth transactions, and a highly accessible shopping experience, which are all key to retain customers.

## 5.5 CASE STUDY

A relevant case study highlighting the use of an analytic data warehouse by an e-commerce company is Levi Strauss & Co.'s partnership with Google Cloud. Levi Strauss & Co. (LS&Co.), renowned for its iconic denim jeans, has integrated artificial intelligence (AI) into its supply chain to streamline fabric procurement and enhance operational efficiency.

In 2020, Levi's collaborated with Google Cloud to develop a comprehensive data warehouse solution that integrates data from various sources, including purchases, web browsing, retail partner sales, and its loyalty program. This integration enabled Levi's to analyze customer behavior and market trends more effectively. For instance, by leveraging this data warehouse, Levi's identified a growing preference for baggy jeans across different demographics, leading to targeted marketing campaigns and adjustments in inventory and design strategies. This strategic use of data resulted in a 15% increase in sales of loose-fit jeans in a recent quarter. Levi Strauss & Co. (LS&Co.), renowned for its iconic denim jeans, has integrated artificial intelligence (AI) into its supply chain to streamline fabric procurement and enhance operational efficiency.

### Technology Used in the Data Warehouse

Levi Strauss & Co. uses **Google Cloud's BigQuery** as the foundation of their data warehouse. BigQuery is a fully managed, serverless cloud-based data warehouse solution capable of handling massive amounts of structured and semi-structured data with ease. This technology was chosen because of its scalability, speed in query execution, real-time analytics capabilities, and integration with other tools in the Google Cloud ecosystem, such as AI/ML features and data visualization tools like Looker. Alternatives that could have been considered include **Amazon Redshift**, **Snowflake**, or **Microsoft Azure Synapse Analytics**, but BigQuery's seamless integration and advanced analytics functionalities gave it a competitive edge.

### Why This Technology Was Chosen and what alternatives were considered

BigQuery's ability to process terabytes of data quickly and handle real-time analytics made it an ideal choice for Levi Strauss. The company needed a platform that could integrate multiple data streams, including retail sales, loyalty programs, and website activity, into a single view. Additionally, Google Cloud's advanced tools for artificial intelligence and machine learning

enabled Levi's to uncover patterns in consumer behavior, such as emerging fashion trends. Its cost-effectiveness (pay-per-query pricing model) and serverless architecture also reduced the burden of managing on-premise infrastructure.

### **What is one use case that is particularly important for that company**

One critical use case for Levi's data warehouse is trend analysis and predictive modeling. For example, Levi's used their integrated data warehouse to identify an increase in consumer interest in loose-fitting jeans. The insight helped the company proactively adjust its marketing strategies, optimize inventory for high-demand styles, and cater to specific customer demographics. This resulted in a 15% increase in sales of baggy jeans in a recent quarter, demonstrating the value of real-time analytics and data-driven decision-making.

### **What access management considerations must be taken into account in managing this company's databases**

In managing Levi's data warehouse, it is essential to ensure that access to data is appropriately controlled based on roles and responsibilities. Internal stakeholders such as marketing teams need access to consumer trends, purchase data, and loyalty program insights, while inventory managers require real-time stock level data and forecasts. Meanwhile, executives and financial teams need high-level performance metrics and revenue insights. Access management is enforced using Google Cloud Identity and Access Management (IAM), which enables granular permission controls to ensure only authorized users can access sensitive data. Additionally, privacy compliance (such as GDPR for European customers) must be adhered to, ensuring that personally identifiable information (PII) is encrypted and only accessible when necessary.

## BIBLIOGRAPHY

Mapanga, I. & Kadebu, P., 2013. Database Management Systems: A NoSQL Analysis.

Ajayi Rosiji, 2015. ACID Properties, CAP Theorem & Mobile Databases - Advanced Topics in Database Systems. Colorado Technical University.

Syed Akbar Mehdi, 2017. CAP Theorem and Distributed Database Consistency. India.

Anju Santosh Yedatkar, 2024. Database Management Systems: An Examination using NoSQL. Assistant Professor, Computer Science Department, Sarhad College of Arts, Commerce and Science, Katraj, Pune, India.

AI Expert Network, 2023. How Levi Strauss is using AI to improve business efficiency and decision-making. Available at: <https://aiexpert.network/levi-strauss-ai/>