TRENDIFY

E-Commerce Database Design

BY: APPLE GRACE MASMELA

TRENDIFY

Introduction

With a background in Information Technology focusing on multimedia design, my exposure to **database management** was initially limited. While I was familiar with some technical terms, the rapid evolution of technology has introduced **new trends and advancements** in data management. Having worked in project management for several years, I gained extensive experience in handling and overseeing projects. However, as I transitioned into the field of data science, my curiosity about **database design, data analytics, and data engineering grew.**

E-commerce platforms require a well-structured and efficient database system to handle the vast amount of data generated daily. I chose **Trendify,** a fictional e-commerce project, as the subject of this case study to understand the intricacies of database design in an online shopping environment. Trendify's mission to provide a seamless shopping experience with efficient order management, inventory tracking, and logistics made it a compelling choice for this project. By developing this relational database, I aimed to bridge the gap between **theoretical database concepts and real-world applications.**

In this article, I will walk you through the process of creating my first relational database and the methodology I used to convert raw e-commerce processes into a streamlined and effective system for managing operations efficiently. This case study provides insights into database structuring, optimization, and the key considerations involved in building a **scalable and functional database.**

Identifying the Requirements to Create the Database

To ensure Trendify's database met business and operational needs, I identified key requirements:

- **Order and Customer Management:** Store customer information, track purchases, and manage payments.
- **Inventory and Supplier Tracking:** Maintain stock levels and link products to respective suppliers.
- **Logistics and Shipment Management:** Track order status, shipment details, and delivery times.
- **Promotional Offers and Discounts:** Store discount offers and apply them to eligible orders.

Walkthrough on the Thinking Process and Database Creation

The design process began by defining the core functionalities that the database needed to support. These included:

1. Identifying the critical entities in the system.
2. Defining the attributes associated with each entity.
3. Establishing relationships between entities to maintain data integrity and avoid redundancy.
4. Normalizing the database to enhance efficiency and scalability.

TRENDIFY

Identifying Entities and Attributes

Entities represent real-world objects in an e-commerce system. The key entities identified were:

- **Customers:** Includes attributes such as customer ID, name, email, phone number, and address.
- **Products:** Contains product ID, name, price, description, stock quantity, and supplier reference.
- **Orders:** Tracks order ID, customer ID, product ID, order date, quantity, and payment method.
- **Employees:** Includes employee ID, name, role, and contact details.
- **Suppliers:** Stores supplier ID, name, and contact information.
- **Payment Mode**: Identifies different payment methods available.
- **Shipment:** Tracks shipping details, including shipment ID, status, and date.
- **Offer:** Stores promotional discounts and their validity periods.

Preliminary Entities:

- Stores
- Products
- Customers
- Employees
- Orders
- Categories
- Payment Type
- Suppliers
- Shipment
- Offers
- Categories

Predefined Tables:

- Stores
- Products
- Product
- Category
- Customers
- Employees
- Orders
- Payment Mode
- Suppliers
- Shipment

Defining Relationship

One-to-Many Relationships:

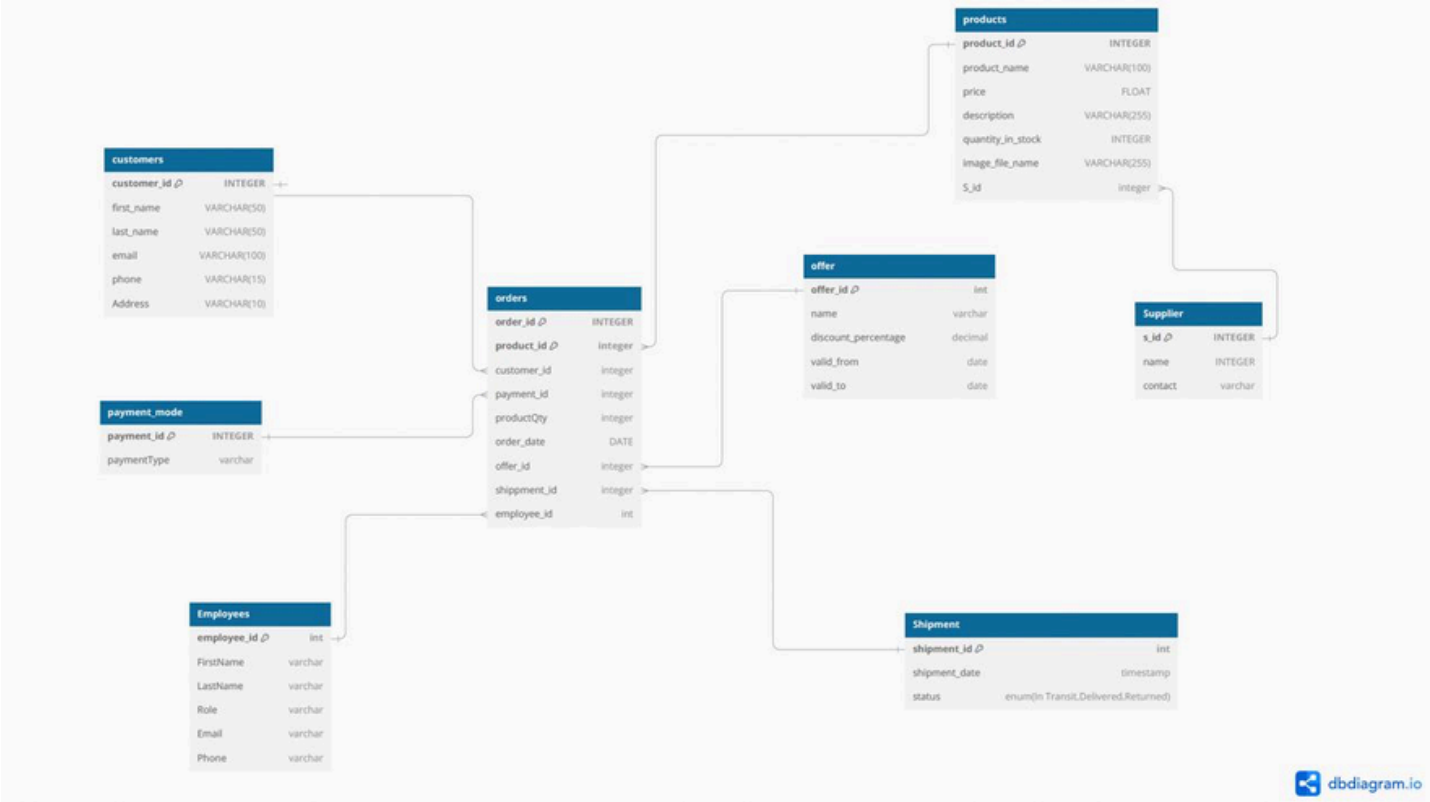
1. **Customers → Orders** (One Customer can place many Orders, but an Order belongs to only one Customer.)
2. **Products → Orders** (One Product can be part of many Orders, but each Order contains specific Products.)
3. **Payment Mode → Orders** (One Payment Mode can be used in multiple Orders, but each Order has only one Payment Mode.)
4. **Employees → Orders** (One Employee can process multiple Orders, but each Order is handled by one Employee.)
5. **Suppliers → Products** (One Supplier can provide multiple Products, but each Product has only one Supplier.)
6. **Orders → Shipment** (One Order is associated with one Shipment, but a Shipment can handle multiple Orders.)
7. **Offer → Orders** (One Offer can apply to multiple Orders, but an Order can have only one Offer.)

Many-to-One Relationships:

1. Each Order is associated with only one Customer, Payment Mode, Shipment, Employee, and Offer, but each of these entities can be linked to multiple Orders.
2. Each Product is associated with one Supplier, but a Supplier can provide multiple Products.

TRENDIFY

ER Diagram



To design the **ER diagram** for Trendify, I started by identifying the key components needed for an e-commerce platform, such as **customers, orders, products, suppliers, payments, shipments, employees, and offers**. I defined the necessary attributes for each entity and mapped out one-to-many relationships, like Customers placing multiple Orders and Suppliers providing multiple Products. To ensure data integrity, I used **primary and foreign keys,** and I structured the tables to minimize redundancy through normalization. This process helped create a **well-organized database** that supports **efficient order management and transaction processing.**

TRENDIFY

Creating Database

```
-- Step 1: Create Database
Create DATABASE trendify;
use trendify;

-- Step 2: Create Tables
CREATE TABLE customers (
 customer_id INT AUTO_INCREMENT PRIMARY KEY,
 first_name VARCHAR(50),
 last_name VARCHAR(50),
 email VARCHAR(100) UNIQUE
);

CREATE TABLE products (
 product_id INT AUTO_INCREMENT PRIMARY KEY,
 product_name VARCHAR(100),
 price DECIMAL(10,2)
);

CREATE TABLE orders (
 order_id INT AUTO_INCREMENT PRIMARY KEY,
 customer_id INT,
 product_id INT,
 quantity INT,
 order_date DATE,
 FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
 FOREIGN KEY (product_id) REFERENCES products(product_id)
);

CREATE TABLE payment_mode (
 payment_id INT AUTO_INCREMENT PRIMARY KEY,
 order_id INT,
 payment_date DATE,
 total_amount DECIMAL(10,2),
 payment_method VARCHAR(50),
 FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

TRENDIFY

The Role of Views in the Database

- Views are essential for simplifying complex queries, improving data organization, and enhancing readabilitywithout altering the underlying tables.
- They help streamline business operations by providing quick access to frequently used information.

Why These Views Were Created

- The views were designed to meet key business needs like order tracking, inventory management, and financial transparency.
- Instead of running multiple queries across different tables, views provide a centralized way to access structured data.

Why These Views Were Created

- **Invoice View:** Helps generate billing details by consolidating customer information, purchased products, payment mode, and order totals.
- **Inventory View:** Gives a clear snapshot of stock levels, tracking product quantities and suppliers to ensure smooth inventory management.
- **Order Details View:** Combines customer orders, product details, quantities, and shipping status in one place, making order tracking more efficient.

Business Impact and Benefits

- **Efficiency**: Reduces the need for complex joins and repetitive queries.
- **Data Consistency:** Ensures that all relevant information, like orders, invoices, and inventory, is always up to date.
- **Scalability:** Makes it easy to expand the database by adding new data points without disrupting the current structure.

TRENDIFY

Invoice View

Purpose: To gather information for tracking invoices per customer by including customer details, order items, and payment transactions.
Scenario: To know the total amount invoiced per customer, including payment details, unit price, and total price calculation.
Tables Used: Orders (orders) , Customers (customers), Products (products), Payment Mode (payment_mode)
View Name: INVOICE SUMMARY
Fields Used: order_id. customer_name, email, order_date, product_name, quantity, payment_method, payment_date, total_amount

```sql
CREATE VIEW InvoiceView AS
SELECT
    o.order_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    c.email,
    o.order_date,
    p.product_name,
    o.quantity,
    pm.payment_method,
    pm.payment_date,
    pm.total_amount
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN products p ON o.product_id = p.product_id
JOIN payment_mode pm ON o.order_id = pm.order_id;

-- Step 5: Verify Data
SELECT * FROM customers;
SELECT * FROM products;
SELECT * FROM orders;
SELECT * FROM payment_mode;

-- Step 6: Query the Invoice View
SELECT * FROM InvoiceView;
```

	order_id	customer_name	email	order_date	product_name	quantity	payment_method	payment_date	total_amount
1	1	John Doe	john@example.com	2024-01-29	Red Sneakers	2	Credit Card	2024-01-29	99.98
2	2	Jane Smith	jane@example.com	2024-01-30	White T-Shirt	1	PayPal	2024-01-30	19.99

TRENDIFY

Order Details View

Purpose: To gather detailed order information, providing insights into customer purchases, pricing, and shipment tracking.
Scenario: To know the detailed order information, including customer name, product details, unit price, total price, and order date.
Tables Used: Orders (orders) , Customers (customers), Products (products),
View Name: ORDER DETAILS VIEW
Fields Used: order_id. customer_name, order_date, product_name, quantity, unit_price, total_price. Tracking_id, Shipment_status

```
108  CREATE VIEW OrderDetailsView AS
109  SELECT
110      o.order_id,
111      CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
112      p.product_name,
113      o.quantity,
114      p.price AS unit_price,
115      (o.quantity * p.price) AS total_price,
116      o.order_date,
117      s.tracking_id,
118      s.status AS shipment_status
119  FROM orders o
120  JOIN customers c ON o.customer_id = c.customer_id
121  JOIN products p ON o.product_id = p.product_id
122  LEFT JOIN shipments s ON o.order_id = s.order_id;
123
124  SELECT * FROM OrderDetailsView;
125
```

	order_id	customer_name	product_name	quantity	unit_price	total_price	order_date	tracking_id	shipment_status
1	1	John Doe	Red Sneakers	2	49.99	99.98	2024-01-29	TRK123456	Shipped
2	2	Jane Smith	White T-Shirt	1	19.99	19.99	2024-01-30	TRK789012	Delivered

TRENDIFY

Order Details View

Purpose: To gather detailed order information, providing insights into customer purchases, pricing, and shipment tracking.
Scenario: To know the detailed order information, including customer name, product details, unit price, total price, and order date.
Tables Used: Orders (orders) , Customers (customers), Products (products),
View Name: ORDER DETAILS VIEW
Fields Used: order_id. customer_name, order_date, product_name, quantity, unit_price, total_price. Tracking_id, Shipment_status

```
108  CREATE VIEW OrderDetailsView AS
109  SELECT
110      o.order_id,
111      CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
112      p.product_name,
113      o.quantity,
114      p.price AS unit_price,
115      (o.quantity * p.price) AS total_price,
116      o.order_date,
117      s.tracking_id,
118      s.status AS shipment_status
119  FROM orders o
120  JOIN customers c ON o.customer_id = c.customer_id
121  JOIN products p ON o.product_id = p.product_id
122  LEFT JOIN shipments s ON o.order_id = s.order_id;
123
124  SELECT * FROM OrderDetailsView;
125
```

	order_id	customer_name	product_name	quantity	unit_price	total_price	order_date	tracking_id	shipment_status
1	1	John Doe	Red Sneakers	2	49.99	99.98	2024-01-29	TRK123456	Shipped
2	2	Jane Smith	White T-Shirt	1	19.99	19.99	2024-01-30	TRK789012	Delivered

TRENDIFY

Inventory View

Purpose: To gather real-time inventory data, for accurate tracking of stock levels, supplier sources, and pricing.
Scenario: We want to track product inventory, including available stock, supplier details, and pricing information.
Tables Used: Supplier (supplier), Products (products)
View Name: INVENTORY VIEW
Fields Used: product_id. product_name, quantity_in_stock, supplier_name, price

```sql
--Create Inventory View
CREATE VIEW InventoryView AS
SELECT
    p.product_id,
    p.product_name,
    COALESCE(p.quantity_in_stock, 0) AS quantity_in_stock, -- Prevent NULL values
    COALESCE(s.name, 'Unknown Supplier') AS supplier_name, -- Default supplier name if missing
    p.price
FROM products p
LEFT JOIN supplier s ON p.s_id = s.s_id;

--
SHOW FULL TABLES IN shein WHERE TABLE_TYPE LIKE 'VIEW';
DESC products;

--Show Inventory View
SELECT * FROM InventoryView;
```

	product_id	product_name	quantity_in_stock	supplier_name	price
1	1	Red Sneakers	100	Nike	49.99
2	2	White T-Shirt	50	Adidas	19.99

Conclusion

The **Trendify database** is a crucial step in organizing and improving e-commerce operations. It provides a structured way to manage **orders, inventory, payments, and customer information,** ensuring that data is easily accessible and transactions are tracked efficiently. More than just storing information, this database helps make better business decisions by identifying **sales trends, customer preferences, and overall performance**. Having clear and well-organized data makes day-to-day operations smoother and more reliable.

This initial version focuses on **order processing, inventory tracking, and customer management,** but it is built to grow with the business. As Trendify expands, new features like **real-time analytics, personalized promotions, and better supplier management** can be added without needing a complete system overhaul. Future updates will focus on making processes even more **automated and efficient**, helping the business stay competitive.

The relational database structure ensures that the system remains **flexible and scalable,** allowing easy integration of new features as business needs evolve. This design not only saves time and effort but also provides a **solid foundation for future growth,** ensuring that Trendify continues to run smoothly and efficiently as it expands.