A Case Study on E-commerce Database Design

A logo with a yellow and blue background

AI-generated content may be incorrect.

Database Design for Sneakerhead

-**Presented by:** [Sharankumar Kodimyala]

**Index**

1. **Introduction**
2. **Mission**
3. **Objectives**
4. **List of Tables**
   * **User & Order Management Tables**
   * **Product & Supplier Management Tables**
   * **Transaction & Shipping Tables**
5. **Tables Description**
   * Users Table
   * Orders Table
   * Feedback Table
   * Products Table
   * Product Categories Table
   * Suppliers Table
   * Payments Table
   * Shipping Information Table
6. **ER Diagram**
7. **Relationships**
8. **Types of Joins in Database**
   * Inner Join
   * Left Join
   * Cross Join
   * Right Join
9. **Conclusion**

**Introduction:**

Sneakerhead is a forward-thinking e-commerce platform specializing in the design and sale of personalized, sustainable sneakers. Focused on providing footwear that combines both style and performance, the platform meets the growing demand for high-quality, customizable sneakers that align with modern values. By offering advanced customization options and using eco-friendly materials, Sneakerhead aims to transform the sneaker industry, creating a seamless shopping experience for every customer.

The platform’s core mission is to offer more than just footwear—it delivers a personalized shopping journey, allowing customers to create unique sneakers that reflect their individual tastes. Whether it's custom designs, colors, or sizes, Sneakerhead ensures each pair is tailored to meet customer preferences. Moreover, the brand is committed to environmental responsibility by offering products made from sustainable materials that minimize waste and reduce the carbon footprint.

In an era where conscious consumerism is on the rise, Sneakerhead not only meets the need for stylish footwear but also encourages a more sustainable approach to fashion.

**Mission Statement:**

Use data-driven insights to create sustainable, high-quality footwear, streamline operations, and deliver outstanding customer experiences.

**Objectives**

1. Analyze Customer Data to Develop Products That Align with Their Needs

2. Optimize Inventory Management and Supply Chain Using Data Insights

3. Gather and Utilize Customer Feedback to Enhance Satisfaction

**List of Tables**

**1. User & Order Management Tables**

* **Users Table** – Stores user details, including personal information, contact details, and account-related data.
* **Orders Table** – Tracks customer purchases, including order status, payment details, and shipping information.
* **Feedback Table** – Captures customer reviews, ratings, and comments on purchased products.

**2. Product & Supplier Management Tables**

* **Products Table** – Stores product details, such as name, description, price, available sizes, and stock levels.
* **Product Categories Table** – Organizes products into distinct categories for easier navigation and filtering.
* **Suppliers Table** – Contains details about suppliers, including contact information and products they provide.

**3. Transaction & Shipping Tables**

* **Payments Table** – Tracks payment transactions, including payment methods, amounts, and statuses.
* **Shipping Information Table** – Stores shipment details, including carrier, tracking numbers, and delivery status.

Tables:

**1.Users Table**

A screenshot of a user account

AI-generated content may be incorrect.

 **USER\_ID:** Primary key.

 **FIRSTNAME:** User's first name.

 **Name:** Full name (consider renaming).

 **Email:** Unique user email.

 **Phone:** Contact number.

 **Address:** Full user address.

**2.Orders Table**

A screenshot of a computer

AI-generated content may be incorrect.

 **Order\_ID:** Primary key, uniquely identifies each order.

 **User\_ID:** Foreign key linking the order to a specific user.

 **OrderDate:** Records the date when the order was placed.

 **TotalAmount:** Stores the total cost of the order, including all items.

 **PaymentStatus:** Indicates whether the payment is pending, completed, or failed.

**3.Feedback Table**

A screenshot of a computer

AI-generated content may be incorrect.

 **Feedback\_ID:** Primary key, uniquely identifies each feedback entry.

 **User\_ID:** Foreign key linking feedback to a specific user.

 **Order\_ID:** Associates feedback with a particular order.

 **FeedbackText:** Stores user comments or reviews about the order.

 **Rating:** Numeric value representing customer satisfaction.

**4.Products Table**

A screenshot of a phone

AI-generated content may be incorrect.

* **Product\_ID:** Primary key, uniquely identifies each product.
* **Name:** Stores the product's name.
* **Category\_ID:** Foreign key linking the product to a category.
* **Order\_ID:** Associates the product with an order.
* **Supplier\_ID:** Links the product to a supplier.
* **Price:** Stores the product’s price.
* **Material:** Specifies the material used in the product.
* **Size:** Indicates the product size.
* **Color:** Defines the product color.
* **StockQuantity:** Represents the available stock of the product.
* **LaunchDate:** Stores the date the product was launched.

**5.Product Categories Table**

A screenshot of a computer

AI-generated content may be incorrect.

 **Category\_ID:** Primary key, uniquely identifies each product category.

 **Name:** Stores the name of the product category.

 **Description:** Provides a brief explanation of the category.

**6.Suppliers Table**

A screenshot of a computer

AI-generated content may be incorrect.

* **Supplier\_ID:** Primary key, uniquely identifies each supplier.
* **Name:** Stores the name of the supplier company.
* **ContactPerson:** Stores the name of the main contact person at the supplier company.
* **Phone:** Stores the supplier's phone number, limited to 15 characters.
* **Email:** Stores the email address of the supplier.
* **Address:** Stores the physical address of the supplier, allowing up to 250 characters.

**7.Payments Table**

A screenshot of a computer

AI-generated content may be incorrect.

 **Payment\_ID:** Primary key, uniquely identifies each payment.

 **OrderID:** Foreign key, links the payment to a specific order.

 **PaymentDate:** Stores the date when the payment was made.

 **PaymentMethod:** Stores the type of payment (e.g., Credit Card, PayPal, Bank Transfer).

 **PaymentStatus:** Stores the status of the payment (e.g., Pending, Completed, Failed).

**8.Shipping Information Table**

A screenshot of a box

AI-generated content may be incorrect.

 **Shipping\_ID:** Primary key, uniquely identifies each shipping record.

 **Order\_ID:** Foreign key, links the shipping record to a specific order.

 **ShippingAddress:** Stores the address where the order is to be delivered.

 **ShippingMethod:** Stores the type of shipping (e.g., Standard, Express, Same-day).

 **ShippingDate:** Stores the date when the order was shipped.

 **DeliveryDate:** Stores the estimated or actual delivery date.

**ER DIAGRAM**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**RELATIONSHIPS:**

**Users & Orders (One-to-Many)**

* **Users Table:** USER\_ID (Primary Key)
* **Orders Table:** ORDER\_ID (Primary Key), USER\_ID (Foreign Key → Users)

**Orders & Payments (One-to-One)**

* **Orders Table:** ORDER\_ID (Primary Key)
* **Payments Table:** PAYMENT\_ID (Primary Key), ORDER\_ID (Foreign Key → Orders)

**Orders & Shipping Information (One-to-One)**

* **Orders Table:** ORDER\_ID (Primary Key)
* **ShippingInformation Table:** SHIPPING\_ID (Primary Key), ORDER\_ID (Foreign Key → Orders)

**Orders & Feedback (One-to-Many)**

* **Orders Table:** ORDER\_ID (Primary Key)
* **Feedback Table:** FEEDBACK\_ID (Primary Key), ORDER\_ID (Foreign Key → Orders), USER\_ID (Foreign Key → Users)

**Orders & Products (One-to-Many)**

* **Orders Table:** ORDER\_ID (Primary Key)
* **Products Table:** PRODUCT\_ID (Primary Key), ORDER\_ID (Foreign Key → Orders)

**Products & Suppliers (Many-to-One)**

* **Suppliers Table:** SUPPLIER\_ID (Primary Key)
* **Products Table:** PRODUCT\_ID (Primary Key), SUPPLIER\_ID (Foreign Key → Suppliers)

**Products & Product Categories (Many-to-One)**

* **ProductCategories Table:** CATEGORY\_ID (Primary Key)
* **Products Table:** PRODUCT\_ID (Primary Key), CATEGORY\_ID (Foreign Key → ProductCategories)`

**TYPES OF JOINS IN DATABASE:**

**Inner join:**

A screenshot of a computer

AI-generated content may be incorrect.

An **INNER JOIN** in SQL is used to retrieve records that have matching values in both tables. It only returns rows where there is a match between the columns specified in the JOIN condition.

**Purpose**

The purpose of an INNER JOIN is to combine related data from two tables based on a common column. It ensures that only records that have matching values in both tables appear in the result set.

### **Use Case**

A typical use case of an INNER JOIN is in an **e-commerce system**, where we need to retrieve a list of all **orders along with user details**. This ensures that only users who have placed at least one order appear in the result.

### **Query**

SELECT Orders.OrderID, Users.Name, Users.Email, Orders.OrderDate, Orders.TotalAmount

FROM Orders

INNER JOIN Users ON Orders.UserID = Users.UserID;

**LEFT JOIN:**

**A screenshot of a computer

AI-generated content may be incorrect.**

A **LEFT JOIN** in SQL is used to retrieve all records from the **left table** and the matching records from the **right table**. If there is no match, the result will still include all records from the left table, with NULL values for columns from the right table.

### **Purpose:**

The purpose of a LEFT JOIN is to ensure that all data from the left table is preserved, even if no corresponding data exists in the right table. It is commonly used when working with **optional relationships**, where some records may not have related data.

### **Use Case:**

A typical use case of a LEFT JOIN is in an **e-commerce system**, where we need to retrieve a list of all users along with their orders. Users who have not placed any orders will still appear in the result set, with NULL values for order-related columns.

### **Query:**

SELECT

Users.UserID, Users.Name,

Users.Email,

Orders.OrderID,

Orders.OrderDate,

Orders.TotalAmount

FROM Users

RIGHT JOIN Orders ON Users.UserID = Orders.UserID;

**CROSS JOIN:**

**A screenshot of a computer

AI-generated content may be incorrect.**

A **CROSS JOIN** in SQL produces the Cartesian product of two tables. It returns all possible combinations of rows from both tables, meaning that each row from the first table is combined with every row from the second table.

### **Purpose**

The purpose of a CROSS JOIN is to generate all possible pairings between two datasets. It is often used when analyzing **combinations** of data, such as pairing all **products with all suppliers**, or creating a **schedule** of all employees against all work shifts.

### **Use Case**

A common use case for a **CROSS JOIN** is in an **e-commerce system**, where we want to **list all possible product and supplier combinations**, regardless of whether a supplier actually provides a specific product. This can help in **potential business analysis** or identifying **new supplier opportunities**.

**Query**

SELECT Products. Name AS ProductName, Suppliers.Name AS SupplierName

FROM Products

CROSS JOIN Suppliers;

**RIGHT JOIN:**

**A screenshot of a computer

AI-generated content may be incorrect.**

A **LEFT JOIN** in SQL is used to retrieve all records from the **left table** and the matching records from the **right table**. If there is **no match**, the result will still include all records from the left table, with **NULL values** for columns from the right table.

### **Purpose**

The purpose of a **LEFT JOIN** is to **preserve all data** from the left table, even if there is **no corresponding** data in the right table. It is commonly used in cases where **not all entities have related data**.

### **Use Case**

A typical use case of a **LEFT JOIN** is in an **e-commerce system**, where we need to **retrieve a list of all users along with their orders**. Users who have **not placed any orders** will still appear in the result set, with **NULL values** for order-related columns.

**Query**

SELECT

Users.UserID, Users.Name, Users.Email, Orders.OrderID, Orders.OrderDate, Orders.TotalAmount

FROM Users

LEFT JOIN Orders ON Users.UserID = Orders.UserID

UNION

SELECT

Users.UserID, Users.Name, Users.Email, Orders.OrderID, Orders.OrderDate, Orders.TotalAmount

FROM Users RIGHT JOIN Orders ON Users.UserID = Orders.UserID;Cons

**Conclusion:**

A well-structured database is essential for efficient data management, ensuring seamless relationships between entities like users, orders, products, and suppliers through primary and foreign keys. SQL joins, including **Inner Join, Left Join, Right Join, and Cross Join**, enable efficient data retrieval by linking related tables. Proper table relationships enhance data integrity, minimize redundancy, and improve query performance, making databases scalable and reliable. Understanding these concepts is crucial for data analysts, developers, and administrators to optimize data handling in real-world applications, ensuring consistency, security, and efficiency in business operations.