

database design & development for e-commerce

CASE STUDY



February 2, 2025

nAME: dHAIRYA tHANKI

EMAIL ID: thankidhairya@outlook.com

COMPANY NAME: Sneakerhead

A logo with a yellow and blue background

AI-generated content may be incorrect.

Table of Contents

[Sneaker Head: Revolutionizing Sustainable Footwear 3](#_Toc189510524)

[Introduction 3](#_Toc189510525)

[Background and Context 3](#_Toc189510526)

[Mission 3](#_Toc189510527)

[Objectives 3](#_Toc189510528)

[Challenges and Solutions 4](#_Toc189510529)

[Tables and Fields 4](#_Toc189510530)

[Users Table 4](#_Toc189510531)

[Orders Table 5](#_Toc189510532)

[Product Categories Table 5](#_Toc189510533)

[Suppliers Table 5](#_Toc189510534)

[Payments Table 6](#_Toc189510535)

[Shipping Information Table 6](#_Toc189510536)

[Feedback Table 7](#_Toc189510537)

[Relationships 8](#_Toc189510538)

[USERS & ORDERS 8](#_Toc189510539)

[ORDERS & PAYMENTS 8](#_Toc189510540)

[ORDERS & SHIPPING INFORMATION 8](#_Toc189510541)

[ORDERS & FEEDBACK 8](#_Toc189510542)

[ORDERS & PRODUCTS 8](#_Toc189510543)

[PRODUCTS & SUPPLIERS 8](#_Toc189510544)

[PRODUCTS & PRODUCT CATEGORIES 8](#_Toc189510545)

[Entity Relation Diagram 9](#_Toc189510546)

[9](#_Toc189510547)

[Join Types in Database 10](#_Toc189510548)

[INNER JOIN: 10](#_Toc189510549)

[Query: 10](#_Toc189510550)

[Output: 10](#_Toc189510551)

[Purpose: 10](#_Toc189510552)

[Use Cases 11](#_Toc189510553)

[LEFT JOIN (or LEFT OUTER JOIN) 11](#_Toc189510554)

[Query: 11](#_Toc189510555)

[Output: 11](#_Toc189510556)

[Purpose: 11](#_Toc189510557)

[Use Cases: 12](#_Toc189510558)

[RIGHT JOIN (or RIGHT OUTER JOIN) 12](#_Toc189510559)

[Query: 12](#_Toc189510560)

[Output: 12](#_Toc189510561)

[Purpose: 13](#_Toc189510562)

[Use Cases: 13](#_Toc189510563)

[CROSS JOIN 14](#_Toc189510564)

[Query: 14](#_Toc189510565)

[Output: 14](#_Toc189510566)

[Purpose: 14](#_Toc189510567)

[Use Case: 14](#_Toc189510568)

[Impact and Results 15](#_Toc189510569)

[Future Plan 15](#_Toc189510570)

[Visuals 16](#_Toc189510571)

[Entity-Relationship Diagram (ERD) 16](#_Toc189510572)

[Entities and their Relationships: 16](#_Toc189510573)

[Example ERD Layout: 16](#_Toc189510574)

[Tables and Fields Diagram 16](#_Toc189510575)

[Data Flow Diagram (DFD) 17](#_Toc189510576)

[Conclusion 18](#_Toc189510577)

# Sneaker Head: Revolutionizing Sustainable Footwear

# Introduction

Sneaker Head is an innovative e-commerce platform specializing in personalized, sustainable sneakers. We are dedicated to offering a curated collection of high-quality footwear that enhances customer experiences and ensures seamless online shopping. Our focus on sustainability and personalization sets us apart, allowing us to meet the unique needs and preferences of our customers.

# Background and Context

The inspiration behind Sneaker Head stems from the growing demand for sustainable fashion and the increasing consumer desire for personalized products. In recent years, the footwear industry has faced criticism for its environmental impact, with many consumers seeking eco-friendly alternatives. Sneaker Head aims to address this need by offering customizable, sustainable sneakers that do not compromise on style or quality. By leveraging data-driven insights, Sneaker Head creates footwear that aligns with customer preferences while minimizing its environmental footprint.

# Mission

Our mission is to use data-driven insights to create sustainable, high-quality footwear, streamline operations, and deliver outstanding customer experiences. We are committed to reducing our environmental footprint while providing top-notch products and services.

# Objectives

* **Analyze Customer Data:** We leverage advanced analytics to understand our customers' preferences, enabling us to develop products that align with their needs and expectations.
* **Optimize Inventory Management and Supply Chain:** By utilizing data insights, we ensure efficient inventory management and a responsive supply chain, minimizing waste and enhancing operational efficiency.
* **Enhance Customer Satisfaction:**  We gather and utilize customer feedback to continuously improve our products and services, ensuring a superior customer experience.

# Challenges and Solutions

During the development and implementation of Sneaker Head's database design, several challenges were encountered:

1. **Data Integration:** Integrating data from various sources such as customer interactions, sales, and inventory was complex. To address this, we implemented a robust data management system that consolidates and processes data efficiently.
2. **Scalability:** As our customer base grew, the database needed to handle increased loads. We adopted scalable database technologies that can accommodate growth without compromising performance.
3. **Customer Insights:** Extracting actionable insights from vast amounts of data required advanced analytics. We employed machine learning algorithms to analyze customer behavior and preferences, enabling us to tailor our products and services effectively.

# 

# Tables and Fields

## Users Table

* **USER\_ID (INTEGER):** Unique identifier for each user. This is crucial for maintaining distinct records and ensuring data integrity across the database.
* **FIRSTNAME (VARCHAR (30)):** First name of the user. This field allows for personalization and better customer interaction.
* **Name (VARCHAR (100)):** Full name of the user. Storing the full name helps in formal communications and records.
* **Email (VARCHAR (100)):** Email address of the user. Essential for user account verification, communication, and marketing purposes.
* **Phone (VARCHAR (15)):** Phone number of the user. Useful for direct communication, order confirmations, and support.
* **Address (VARCHAR (250)):** Residential or shipping address of the user. Needed for delivery purposes and maintaining accurate records for shipping.

## Orders Table

* **Order\_ID (INTEGER):** Unique identifier for each order. Key to tracking individual orders and their statuses.
* **USER\_ID (INTEGER):** Foreign key referencing USERS, linking orders to users. Facilitates the connection between users and their respective orders.
* **Order Date (DATE):** Date when the order was placed. Important for tracking order timelines and processing durations.
* **Total Amount (DECIMAL):** Total amount of the order. Necessary for financial records and transaction verification.
* **Payment Status (VARCHAR (50)):** Status of the payment (e.g., Paid, Pending, Failed). Crucial for financial auditing and customer service.

## 

## Product Categories Table

* **Category\_ID (INTEGER):** Unique identifier for each product category. Helps in organizing products into distinct categories for better inventory management.
* **Name (VARCHAR (100)):** Name of the category (e.g., Sneakers, Running Shoes). Facilitates easy identification and classification of products.
* **Description (VARCHAR (250)):** Description of the category. Provides context and information about each category, aiding in customer decision-making.

## 

## Suppliers Table

* **Supplier\_ID (INTEGER):** Unique identifier for each supplier. Important for managing supplier relationships and inventory sourcing.
* **Name (VARCHAR (100)):** Name of the supplier. Essential for identifying and communicating with suppliers.
* **Contact Person (VARCHAR (100)):** Contact person for the supplier. Facilitates direct communication and coordination.
* **Phone (VARCHAR (15)):** Phone number of the supplier. Useful for immediate contact and support.
* **Email (VARCHAR (100)):** Email address of the supplier. Necessary for formal communication and record-keeping.
* **Address (VARCHAR (250)):** Address of the supplier. Important for logistical purposes and verifying supplier locations.

## Payments Table

* **Payment\_ID (INTEGER):** Unique identifier for each payment. Key to tracking individual payments and their details.
* **OrderID (INTEGER):** Foreign key referencing Orders, linking payments to orders. Ensures accurate association between payments and their corresponding orders.
* **Payment Date (DATE):** Date when the payment was made. Important for financial records and verifying transaction dates.
* **Payment Method (VARCHAR (50)):** Method of payment (e.g., Credit Card, PayPal). Helps in understanding customer payment preferences and trends.
* **Payment Status (VARCHAR (50)):** Status of the payment (e.g., Completed, Failed). Crucial for financial auditing and resolving payment issues.

## 

## Shipping Information Table

* **Shipping\_ID (INTEGER):** Unique identifier for each shipping transaction. Key to tracking shipping details and ensuring accurate delivery records.
* **Order\_ID (INTEGER):** Foreign key referencing Orders. Facilitates the connection between orders and their shipping details.
* **Shipping Address (VARCHAR (250)):** Address where the order is being shipped. Necessary for delivery purposes and maintaining accurate shipping records.
* **Shipping Method (VARCHAR (100)):** Shipping method (e.g., Standard, Express). Provides options for customers and helps in managing delivery expectations.
* **Shipping Date (DATE):** Date when the order was shipped. Important for tracking shipping timelines and ensuring timely deliveries.
* **Delivery Date (DATE):** Estimated or actual delivery date. Helps in managing customer expectations and tracking delivery performance.

## 

## Feedback Table

* **Feedback\_ID (INTEGER):** Unique identifier for each feedback entry. Key to tracking individual feedback and ensuring accurate records.
* **User\_ID (INTEGER):** Foreign key referencing USERS. Links feedback to specific users for personalized responses.
* **Order\_ID (INTEGER):** Foreign key referencing ORDERS. Connects feedback to specific orders for context.
* **Feedback Text (TEXT):** Customer feedback or review. Provides valuable insights into customer experiences and areas for improvement.
* **Rating (INTEGER):** Rating given by the customer (e.g., 1-5 stars). Quantitative measure of customer satisfaction.
* **Feedback Date (DATE):** Date when the feedback was submitted. Important for tracking feedback timelines and trends.

# Relationships

## USERS & ORDERS

One-to-Many relationship: A User (USER\_ID) can place multiple Orders (ORDER\_ID), but each order is placed by only one user. This relationship is essential for tracking user activity and order history.

## ORDERS & PAYMENTS

One-to-One relationship: Each Order (Order\_ID) has a corresponding Payment (PAYMENT\_ID) with details like payment method and status. This ensures accurate financial records and payment tracking.

## ORDERS & SHIPPING INFORMATION

One-to-One Relationship: Each Order (Order\_ID) has a single Shipping Information record (SHIPPING\_ID), which tracks details like shipping method and delivery date. This is crucial for ensuring accurate and timely deliveries.

## ORDERS & FEEDBACK

One-to-Many Relationship: A single Order (Order\_ID) can have multiple Feedback entries (FEEDBACK\_ID), but each feedback entry is linked to one specific order and user. This relationship helps in understanding customer satisfaction and areas for improvement.

## ORDERS & PRODUCTS

One-to-Many Relationship: An Order (Order\_ID) can contain multiple Products (PRODUCT\_ID), but each product is assigned to a specific order. This is important for managing inventory and tracking product sales.

## PRODUCTS & SUPPLIERS

Many-to-One Relationship: Multiple Products (PRODUCT\_ID) can be supplied by a single Supplier (SUPPLIER\_ID), but each product is associated with only one supplier. This relationship helps in managing supplier relationships and sourcing inventory.

## PRODUCTS & PRODUCT CATEGORIES

Many-to-One Relationship: A Product (PRODUCT\_ID) belongs to a single Product Category (CATEGORY\_ID), but each category can have multiple products. This is crucial for organizing inventory and improving product discoverability.

# Entity Relation Diagram

# A screenshot of a computer

# Join Types in Database

## INNER JOIN:

This join returns only the records that have matching values in both tables. For example, if we join the Users and Orders tables using an inner join, we will get a list of users who have placed at least one order. Users who have never placed an order will not be included 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;

# A screenshot of a computerOutput:

### Purpose:

Retrieves only matching records from both tables. If no match is found, the row is excluded.

# Use Cases

• Getting customers who have placed orders.

• Finding employees who belong to a department.

• Fetching products that have categories assigned

# LEFT JOIN (or LEFT OUTER JOIN)

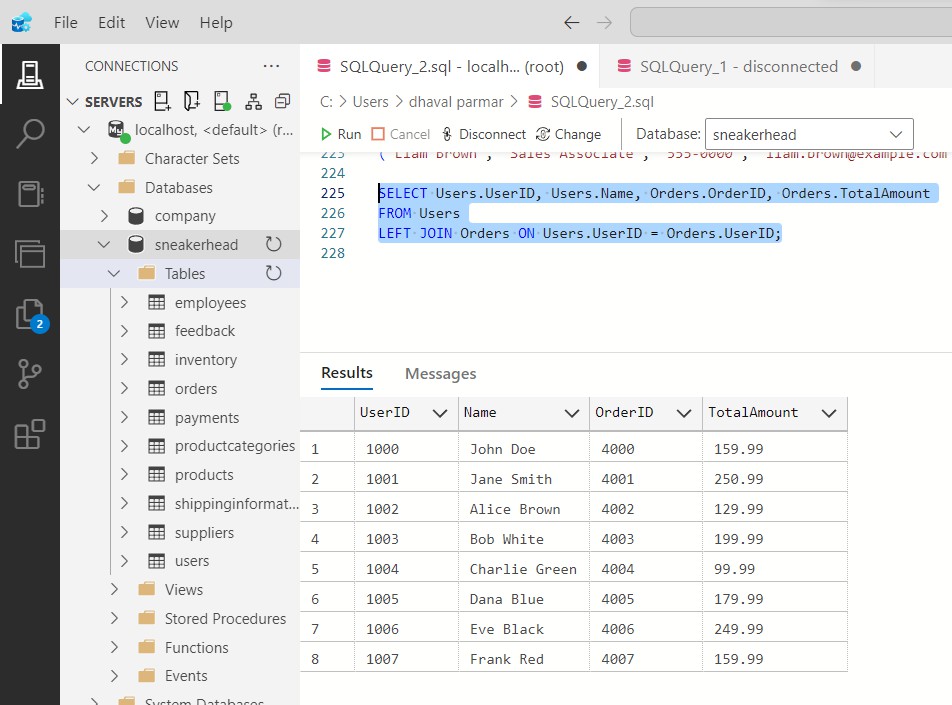
This join returns all records from the left table and only the matching records from the right table. If there are no matches, NULL values are displayed for columns from the right table. For example, if we join Users with Orders using a left outer join, we will get all users, including those who haven’t placed an order. Users without orders will have NULL values in the order-related columns.

## Query:

SELECT Users.UserID, Users.Name, Orders.OrderID, Orders.TotalAmount FROM Users

LEFT JOIN Orders ON Users.UserID = Orders.UserID.

## Output:

****

### Purpose:

Return all records from the left table and matching records from the right table. If no match is found, NULL is returned.

## Use Cases:

* Finding users who haven’t placed orders.
* Listing all products, even if they have no supplier.
* Getting employees, even if they are not assigned to a department.

# RIGHT JOIN (or RIGHT OUTER JOIN)

This join returns all records from the right table and only the matching records from the left table. If there are no matches, NULL values are displayed for columns from the left table. For example, if we join Orders with Users using the right outer joint, we will get all orders, including those that do not have associated users (e.g., orders placed by deleted accounts). Orders without users will have NULL values in the user-related columns.

## Query:

SELECT Suppliers.SupplierID, Suppliers.Name, Products.Name AS ProductName FROM Suppliers

RIGHT JOIN Products ON Suppliers.SupplierID = Products.SupplierID.

## Output:

A screenshot of a computer

AI-generated content may be incorrect.

### Purpose:

Returns all records from the right table and matching records from the left table. If no match is found, NULL is returned.

### Use Cases:

* Listing all suppliers, even those who haven’t supplied any products.
* Finding departments that don’t have any employees assigned.
* Getting all orders, even if they don’t have a registered user.

# CROSS JOIN

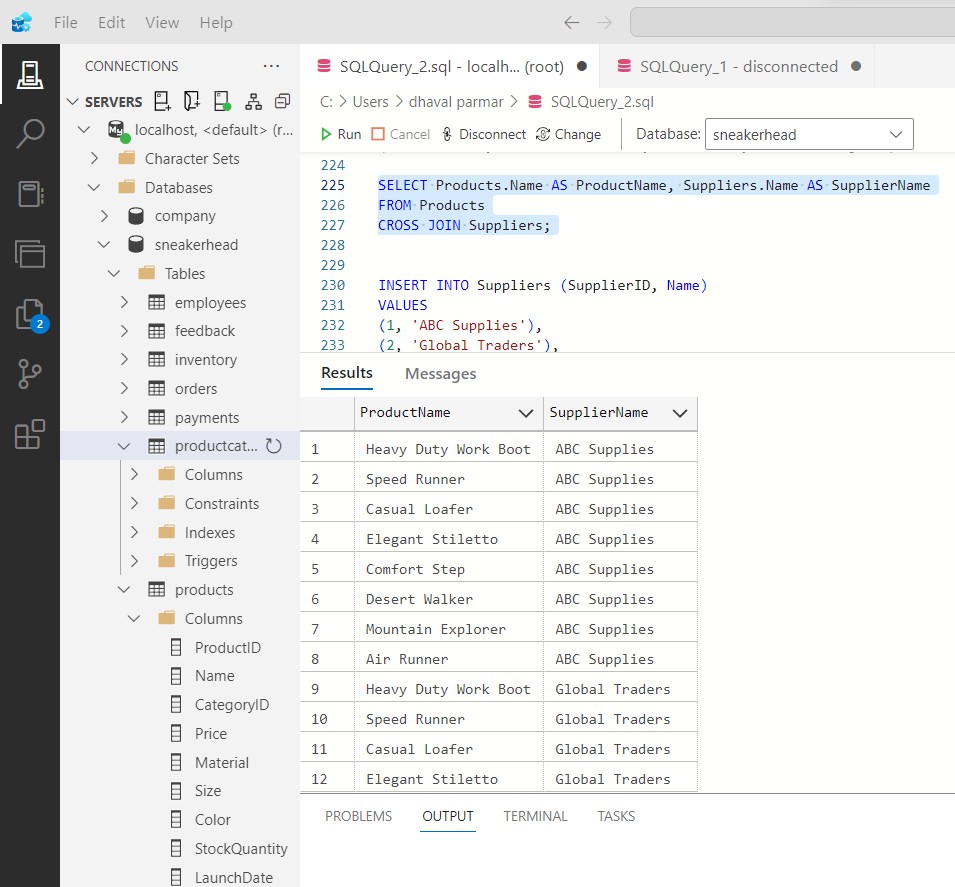
This join returns the Cartesian product of both tables, meaning every row from the first table is paired with every row from the second table. For example, if we perform a cross join between Users and Products, the result will show every user associated with every product, even if they have never purchased it. This can be useful for generating all possible combinations for recommendations or promotions.

## Query:

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

CROSS JOIN Suppliers.

## Output:



Purpose:Creates a Cartesian product (all possible combinations of rows).

### Use Case:

* Listing all possible combinations of products and suppliers.
* Generating all possible matchups between players in a tournament.
* Creating pricing models for all combinations of services and discounts.

# Impact and Results

Since its launch, Sneaker Head has received positive feedback from customers who appreciate the personalized and sustainable approach to footwear. Early metrics indicate a significant reduction in returns and increased customer satisfaction, thanks to our data-driven insights and tailored product offerings. Our efficient inventory management has minimized waste and optimized our supply chain, further enhancing our sustainability efforts.

# Future Plan

Looking ahead, Sneaker Head plans to expand its product range to include more eco-friendly materials and innovative designs. We aim to integrate advanced AI-driven personalization features that provide even more customized experiences for our customers. Additionally, we are exploring partnerships with sustainable suppliers and enhancing our data analytics capabilities to continue improving our operations and customer satisfaction.

Visuals

To create the visuals, you can use the following descriptions:

# Entity-Relationship Diagram (ERD)

## Entities and their Relationships:

* Users: Connects to Orders (One-to-Many)
* Orders: Connects to Payments (One-to-One), Shipping Information (One-to-One), Feedback (One-to-Many), and Products (One-to-Many)
* Products: Connects to Suppliers (Many-to-One) and Product Categories (Many-to-One)

### Example ERD Layout:

Users --------< Orders --------< Payments

| | |--------< Shipping Information

| |

| | |--------< Feedback

| |

+----------< Products ---------< Suppliers

Product Categories

## Tables and Fields Diagram

Create a visual representation of each table with its fields, showing their data types and primary keys.

Example Table Layout for "Users":

+------------------+

| Users |

+------------------+

| USER\_ID (PK) |

| FIRSTNAME |

| Name |

| Email |

| Phone |

| Address |

+------------------+

## Data Flow Diagram (DFD)

Illustrates how data flows between different parts of your system.

Example DFD Layout:

User -> Orders -> Payments

|-> Shipping Information

|-> Feedback

User -> Products -> Suppliers

|-> Product Categories

# Conclusion

This database design serves as the backbone for Sneaker Head’s operations, ensuring seamless management of production, inventory, orders, and customer interactions. With clear objectives and a robust foundation, Sneaker Head is well-equipped to meet its mission and thrive in the competitive footwear industry. By continuously leveraging data-driven insights and maintaining a strong focus on sustainability and customer satisfaction, Sneaker Head is poised to make a lasting impact in the world of e-commerce and sustainable fashion.