Hammad Mehmood | 241795

DBMS Semester project | E-commerce System

Database Schema Analysis: Normalization & Design Decisions

Executive Summary

The provided e-commerce database schema demonstrates a well-structured relational design that effectively balances normalization principles with practical performance considerations. The schema achieves primarily 3NF (Third Normal Form) to optimize query performance for common e-commerce operations.

Normalization Analysis

First Normal Form (1NF) Compliance:

- Fully Compliant
- Evidence:
  - All tables have atomic values (no repeating groups or arrays)
  - Each column contains indivisible data elements
  - Primary keys uniquely identify each row
  - No multi-valued attributes present
- Examples:
  - User names are split into *first_name* and *last_name* rather than a single *full_name* field
  - Addresses are broken down into atomic components (*address_line_1*, *address_line_2*, *city, state*, etc.)
  - Phone numbers stored as single values without internal structure parsing

Second Normal Form (2NF) Compliance:

- Fully Compliant
- Evidence:
    - All non-key attributes are fully functionally dependent on the entire primary key.
    - Composite primary keys (where present) don't have partial dependencies.
    - No partial dependencies exist in any table.
  - Examples:
    - In *order_item* table, both *quantity* and *unit_price* depend on the complete combination of *order_id* and *product_id .*
    - In *cart_item*, the quantity depends on the specific cart-product combination.
    - Junction tables like *product_supplier* properly eliminate many-to-many relationships.

Third Normal Form (3NF) Compliance:

- Fully Compliant with Strategic Exceptions.
- Evidence:
    - Most transitive dependencies have been eliminated.
    - Related data is properly separated into distinct tables.
    - Lookup tables are used for categorical data.
- Examples:
    - Address Information in *Orders*: *shipping_address* and *billing_address* are stored as *VARCHAR* fields in the *Orders* table rather than just foreign keys to *Address* table
        - Justification: Preserves historical accuracy of addresses at time of order

- Alternative: Could reference Address table, but addresses might change over time

- Product Pricing: Both *base_price* in *Product* and *unit_price* in *OrderItem* exist.
  - Justification: Maintains price history and handles dynamic pricing

Normalized Schema:

- **User & Role Management:**
  - Tables: *user, customer, admin*
  - Justification:
    - Separation of user roles (customer/admin) improves clarity and security.
    - *user_type* in user ensures generalization; specialization achieved via *customer* and *admin*.
    - Sensitive information like password and email kept atomic and unique (normalized 1NF, 2NF).

- **Address Management:**
  - Table: *address*
  - Justification:
    - One-to-many relationship from user to address (1 user can have many addresses).
    - Address split into components (line 1, line 2, city, etc.), atomicity respected.
    - address_type (home/office) and *is_default* increase functionality without violating 3NF.

- **Product Catalogue:**
  - Tables: *product*, *category*, *product_variant*, *product_supplier, supplier*
  - Justification:
    - *product_variant* separates different SKUs for one product (e.g., sizes, colors).
    - Avoids repeating product details for each variation — normalizes 2NF/3NF.
    - category linked via FK for classification — prevents data duplication.
    - *product_supplier* handles M:N relationship between products and suppliers — normalized with junction table.

- **Cart & Cart Items:**
  - Tables: *cart, cart_item*
  - Justification:
    - A customer may have only one active cart → FK to *customer_id.*
    - *cart_item* handles multiple products/variants per cart — avoids repeating customer/cart data per item.
    - Supports cart tracking via created_at, updated_at.

- **Orders & Order Items:**
  - Tables: *orders, order_item*
  - Justification:
    - Clear separation between the order (transaction-level) and items (product-level).
    - Allows proper invoice/receipt generation and auditing.
    - All amounts (subtotal, tax, total) directly tied to that order; no redundancy in *order_item*.

- **Payments:**
  - Tables: *payment, payment_method*
  - Justification:
    - Abstracts payment details into separate table.
    - *payment_method* normalized to avoid repeating payment type strings (e.g., "JazzCash", "COD").
    - Each payment links to one order, preserving 2NF.

- **Shipping:**
  - Tables: *shipment, shipping_carrier*
  - Justification:
    - One shipment per order — normalized.
    - *shipping_carrier* avoids repeating company names and tracking URL patterns.
    - Delivery status and dates support tracking and logistics.

- **Returns:**
  - Table: *returns*
  - Justification:
    - Decouples return logic from orders and payments.
    - Enables refund management, return statuses, and customer support workflows without redundancy.

- **Wishlist & Review:**
  - Tables: *wishlist, review*
  - Justification:
    - wishlist avoids M:N duplication between products and customers.
    - review includes verified purchase flag and is normalized — reviews are not stored within product.

- **Discount System:**
  - Tables: *discount_type, discount_code, order_discount*
  - Justification:
    - Discount type abstraction (percentage/fixed) avoids repetitive description strings.
    - discount_code avoids coupling discounts directly with orders, improving reusability.
    - order_discount as junction table supports M:N between orders and discounts (e.g., multiple codes per order or vice versa).

Key Normalization Highlights

- Repeated user address info in multiple tables may cause redundancy, so it is linked via address table.
- Discount logic embedded in orders had Rigid design so it is generalized via *discount_code* and *order_discount*
- Supplier info stored in product may lead to update anomalies, so it is handled with *product_supplier* junction table
- Review and wishlist mixed with orders may cause Functional dependency violation, so their concerns separated.
- Redundant shipping info in orders may create redundancy, so it is linked via shipment table.

Relations in the database

1. User

(user_id (PK), first_name, last_name, email, password, phone, created_at, last_login, user_type)

2. customer

(customer_id (PK, FK), loyalty_points)

3. admin

(admin_id (PK, FK), role, permissions)

4. address

address_id (PK), user_id (FK)

address_line_1, address_line_2, city, state, address_type, is_default, created_at)

5. category

(category_id (PK), category_name, description)

6. product

(product_id (PK), product_name, description, base_price, stock_quantity, category_id (FK), created_at, updated_at)

7. product_variant

(variant_id (PK), product_id (FK), variant_name, description, additional_price, stock_quantity, sku, is_active)

8. **supplier**

(supplier_id (PK), company_name, contact_person, phone, email, address_line_1, address_line_2, city, state, postal_code, country)

9. **product_supplier**

(product_id (FK), supplier_id (FK), supply_price, lead_time_days,is_primary_supplier)

10. **cart**

(cart_id (PK), customer_id (FK), created_at, updated_at)

11. **cart_item**

(cart_item_id (PK), cart_id (FK), product_id (FK), variant_id (FK), quantity, added_at)

12. **orders**

(order_id (PK), customer_id (FK), shipping_address_id, billing_address_id, order_date, status, subtotal, tax_amount, shipping_cost, discount_amount, total_amount)

13. **order_item**

(order_item_id (PK),

order_id (FK), product_id (FK), variant_id (FK), quantity, unit_price, line_total)

14. payment_method

(payment_method_id (PK), method_name, description)

15. payment

(payment_id (PK) , order_id (FK), payment_method_id (FK),

payment_date, amount, payment_status, transaction_id, gateway_response)

16. shipping_carrier

(carrier_id (PK), carrier_name, tracking_url_template, is_active)

17. shipment

(shipment_id (PK), order_id (FK), carrier_id (FK), shipment_date, tracking_number,

estimated_delivery_date, actual_delivery_date, delivery_status, shipping_cost)

18. returns

(return_id (PK), order_id (FK),

shipment_id (FK), return_date, reason, return_status, refund_amount)

19. wishlist

(wishlist_id (PK), customer_id (FK), product_id (FK), added_at)

20. review

(review_id (PK), product_id (FK),

customer_id (FK), rating, comment, review_date, is_verified_purchase)

21. discount_type

(discount_type_id (PK), type_name, description)

22. discount_code

(discount_code_id (PK), discount_type_id (FK), code, description, discount_value, minimum_order_amount, valid_from, valid_till, usage_limit, times_used, is_active)

23. order_discount

(order_id (FK), discount_code_id (FK) , discount_applied)

NOTE: The integrity constraints and the data types of each attribute of every table is not described here, rather it is shown in the .sql file (attached).