Detailed Explanation of the Normalized Database Structure

Overview

The system represents an inventory and sales management application with interconnected components for tracking products, customers, sales, purchases, and deliveries. The normalized database structure ensures data integrity, reduces redundancy, and establishes clear relationships between different entities.

Core User Management

User and Profile (One-to-One)

- **Relationship**: Each Django User has exactly one corresponding Profile (1:1)
- **Implementation**: The Profile table contains a `user_id` foreign key referencing the Django User table
- **Purpose**: Separates authentication (User) from extended user information (Profile)
- **Data Flow**: User account → Profile details

The Profile model extends the built-in Django User model with additional fields such as profile pictures, contact information, status, and role designations (Operative, Executive, Admin). This separation follows the best practice of keeping authentication separate from application-specific user data.

Inventory Classification

Category and Item (One-to-Many)

- **Relationship**: Each Category can have multiple Items, but each Item belongs to exactly one Category (1
)
- **Implementation**: The Item table contains a `category_id` foreign key referencing the Category table
- **Purpose**: Provides hierarchical classification of inventory
- **Data Flow**: Category definition → Item categorization

Categories allow for logical grouping of inventory items (e.g., Electronics, Furniture, Clothing), making search, filtering, and reporting more efficient. The AutoSlugField generates URL-friendly identifiers for both categories and items.

Supply Chain Management

Vendor Relationships

1. **Vendor to Item (One-to-Many)**
    - **Relationship**: Each Vendor can supply multiple Items, but each Item has one designated Vendor (1
)
    - **Implementation**: The Item table contains a `vendor_id` foreign key referencing the Vendor table
    - **Purpose**: Tracks the source of inventory items
    - **Data Flow**: Vendor registration → Item sourcing information
2. **Vendor to Purchase (One-to-Many)**
    - **Relationship**: Each Vendor can fulfill multiple Purchases, but each Purchase is from one Vendor (1
)
    - **Implementation**: The Purchase table contains a `vendor_id` foreign key referencing the Vendor table
    - **Purpose**: Tracks which supplier is fulfilling an order
    - **Data Flow**: Vendor selection → Purchase order creation

The Vendor model stores essential supplier information (name, contact details, address) that's referenced throughout the system for purchasing and inventory management.

Sales Process

Customer to Sale (One-to-Many)

- **Relationship**: Each Customer can make multiple Sales, but each Sale belongs to one Customer (1
)
- **Implementation**: The Sale table contains a `customer_id` foreign key referencing the Customer table
- **Purpose**: Associates transactions with specific customers
- **Data Flow**: Customer record → Sale transaction

The Customer model tracks not only basic contact information but also loyalty points, enabling a rewards system for repeat customers.

Sale to SaleDetail (One-to-Many)

- **Relationship**: Each Sale contains multiple SaleDetails (line items), but each SaleDetail belongs to exactly one Sale (1
)
- **Implementation**: The SaleDetail table contains a `sale_id` foreign key referencing the Sale table
- **Purpose**: Implements the header-detail pattern for itemized sales
- **Data Flow**: Sale creation → Multiple line items (SaleDetails)

The Sale model handles transaction-level information (totals, taxes, payment) while SaleDetail handles item-specific information (product, quantity, price).

Item to SaleDetail (One-to-Many)

- **Relationship**: Each Item can appear in multiple SaleDetails, but each SaleDetail references one specific Item (1
)
- **Implementation**: The SaleDetail table contains an `item_id` foreign key referencing the Item table
- **Purpose**: Links sale line items to the inventory
- **Data Flow**: Item selection → Addition to sale as line item

This relationship allows the system to track which products are selling and update inventory accordingly.

Inventory Management

Item to Purchase (One-to-Many)

- **Relationship**: Each Item can be ordered in multiple Purchases, but each Purchase line is for one Item (1
)
- **Implementation**: The Purchase table contains an `item_id` foreign key referencing the Item table
- **Purpose**: Tracks inventory acquisition
- **Data Flow**: Item selection → Purchase order creation

The Purchase model includes a custom `save()` method that automatically updates the item's quantity when a purchase is recorded, ensuring inventory levels remain accurate.

Item to Delivery (One-to-Many)

- **Relationship**: Each Item can be associated with multiple Deliveries, but each Delivery is for one Item (1
)
- **Implementation**: The Delivery table contains an `item_id` foreign key referencing the Item table
- **Purpose**: Tracks outgoing deliveries of products
- **Data Flow**: Item selection → Delivery scheduling

The Delivery model tracks customer delivery information including location, date, and delivery status, allowing the business to monitor outbound logistics.

Key Database Design Features

1. **Slugs for URL-Friendly Identifiers**
   - Many tables use AutoSlugField to generate URL-friendly identifiers from name or date fields
   - Improves SEO and user experience with readable URLs
2. **Calculated Fields**
   - Several models include fields that are calculated automatically:
     - SaleDetail's `total_detail` (price × quantity)
     - Sale's `sub_total`, `grand_total`, and tax calculations
     - Purchase's `total_value` (price × quantity)
3. **Status Tracking**
   - Multiple entities include status fields:
     - Profile has status (Active, Inactive, On Leave)
     - Delivery has `is_delivered` boolean
     - Purchase has delivery status (Pending, Successful)
4. **Transaction History**
   - The system records timestamps for activities:
     - Sale's `date_added`
     - Purchase's `order_date` and `delivery_date`
     - Delivery's `date`

Data Flow Examples

1. **Complete Sales Cycle**:
   - Customer is registered in the system
   - Items are selected from inventory and added to a Sale
   - SaleDetails are created for each item
   - Payment is processed and recorded in the Sale
   - Delivery is scheduled for the purchased items
2. **Inventory Replenishment**:

    - Low stock is identified for an Item
    - Purchase is created for that Item from a specific Vendor
    - When the Purchase is saved, the Item's quantity is automatically increased
3. **User Management**:
    - Admin creates a User account
    - Profile is created with role, status, and contact information
    - Profile status can be updated as needed (e.g., marking staff "On Leave")