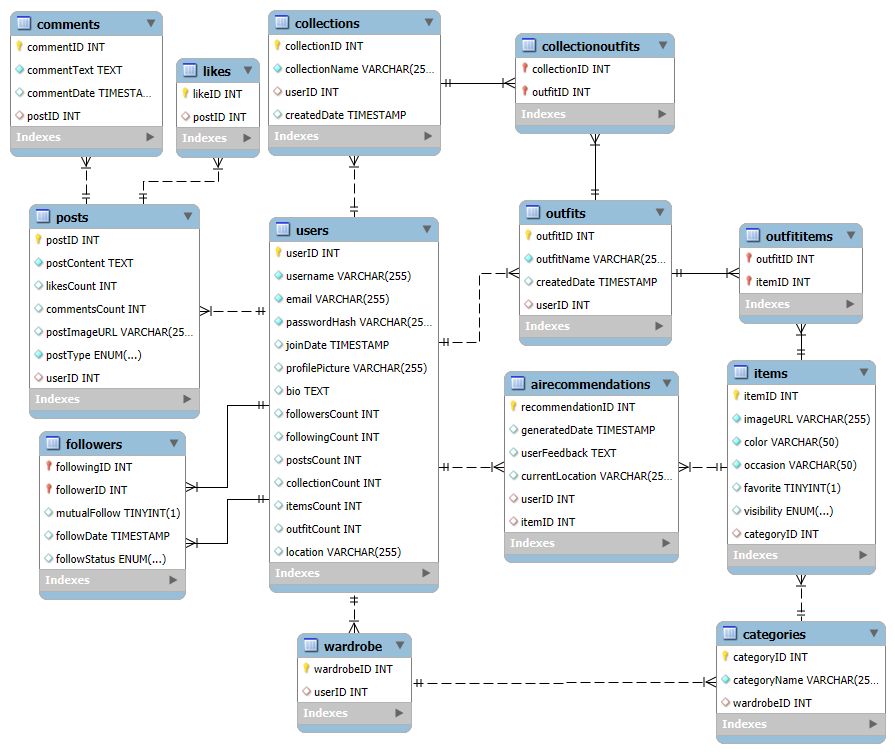
**Fashionaly has 3 databases:**

* User-related features (profile, wardrobe, AI recommendations, social engagement) are in **one NoSQL database**, making queries simpler.
* Transactions (orders, purchases) are in **one SQL database**, ensuring structured consistency.
* Notifications and chats are handled separately for **real-time efficiency** without impacting core functionalities.

**Database 1:** User-related features (profile, wardrobe, AI recommendations, social engagement) are in one NoSQL database, making queries simple****

**1) Users Table**

* **Purpose:** Stores primary user data and activity statistics.
* **How It Works:**
  + Each user has a unique **userID**.
  + **email** is unique for authentication.
  + **passwordHash** ensures secure login.
  + **profilePicture, bio, and location** store additional personal details.
  + **followersCount, followingCount, postsCount, collectionCount, itemsCount, outfitCount** dynamically track user activity.
* **Why It’s Important:**
  + Core table linking to multiple features (social engagement, AI, wardrobe).
  + Securely manages user authentication.

**2) Followers Table (Many-to-Many Relationship)**

* **Purpose:** Manages user-to-user follow relationships.
* **How It Works:**
  + **followingID** refers to the user being followed.
  + **followerID** refers to the user following someone.
  + **mutualFollow** indicates if both users follow each other.
  + **followStatus** manages follow requests (Requested, Accepted, Blocked).
  + If a user deletes their account, all their follow records are automatically removed.
* **Why It’s Important:**
  + Supports the platform’s social network structure.
  + Enables follow requests and content customization.

**3) Wardrobe Table (Each User has One Wardrobe)**

* **Purpose:** Assigns a wardrobe to each user.
* **How It Works:**
  + Each **wardrobeID** belongs to a unique **userID** (1-to-1 relationship).
  + If a user deletes their account, their wardrobe is deleted as well.
* **Why It’s Important:**
  + Stores all clothing items added by the user.
  + Serves as the foundation for AI recommendations.

**4) Categories Table (Previously WardrobeCategories)**

* **Purpose:** Organizes wardrobe items into predefined categories (e.g., casual, formal, activewear).
* **How It Works:**
  + **categoryID** uniquely identifies each category.
  + **categoryName** defines the type (e.g., "Summer", "Winter", "Formal").
  + **wardrobeID** links it to a user’s wardrobe.
* **Why It’s Important:**
  + Helps AI filter clothing types.
  + Allows users to organize their wardrobe efficiently.

**5) Items Table (Previously WardrobeItems)**

* **Purpose:** Stores individual clothing items that belong to a user's wardrobe.
* **How It Works:**
  + **itemID** uniquely identifies each clothing item.
  + **imageURL** stores the link to the clothing image.
  + **color, occasion, favorite, visibility** define item attributes.
  + **categoryID** links each item to a category.
* **Why It’s Important:**
  + Allows AI to generate outfit suggestions.
  + Provides an efficient way for users to manage their wardrobe.

**6) AI Recommendations Table**

* **Purpose:** Stores AI-generated outfit suggestions.
* **How It Works:**
  + **recommendationID** uniquely identifies each recommendation.
  + **generatedDate** records when the AI made the suggestion.
  + **userFeedback** allows users to rate AI-generated outfits.
  + **currentLocation** adjusts suggestions based on weather.
  + **userID** links to the user who received the recommendation.
  + **itemID** links to the recommended clothing items.
* **Why It’s Important:**
  + Stores AI-generated fashion insights for users.
  + Allows users to refine AI recommendations with feedback.

**7) Posts Table (Previously SocialPosts)**

* **Purpose:** Enables users to create and share content (e.g., outfit inspirations, fashion advice).
* **How It Works:**
  + **postID** uniquely identifies each post.
  + **postContent** holds the text description.
  + **likesCount, commentsCount** track engagement.
  + **postImageURL** holds the optional image.
  + **postType** differentiates content (Outfit, Question, Advice).
  + **userID** links the post to the creator.
* **Why It’s Important:**
  + Encourages a community-based engagement model.
  + Enables users to interact and share fashion ideas.

**8) Likes Table**

* **Purpose:** Tracks likes on user posts.
* **How It Works:**
  + **likeID** uniquely identifies each like.
  + **postID** links it to a specific post.
  + If a post is deleted, all associated likes are also removed.
* **Why It’s Important:**
  + Helps measure user engagement.
  + Highlights trending posts.

**9) Comments Table**

* **Purpose:** Stores comments on social posts.
* **How It Works:**
  + **commentID** uniquely identifies each comment.
  + **commentText** holds the content.
  + **commentDate** records when it was made.
  + **postID** links it to the respective post.
* **Why It’s Important:**
  + Enhances discussion and engagement.
  + Creates a dynamic community interaction.

**10) Outfits Table (NEW)**

* **Purpose:** Stores user-created outfits, which consist of multiple wardrobe items.
* **How It Works:**
  + **outfitID** uniquely identifies each outfit.
  + **outfitName** allows users to label outfits (e.g., "Casual Friday").
  + **createdDate** tracks when it was created.
  + **userID** links the outfit to the creator.
* **Why It’s Important:**
  + Allows users to pre-plan looks.
  + Helps organize and save favorite outfit combinations.

**11) OutfitItems Table (NEW)**

* **Purpose:** Links outfits to specific wardrobe items (Many-to-Many).
* **How It Works:**
  + **outfitID** references an outfit.
  + **itemID** references a wardrobe item.
* **Why It’s Important:**
  + Ensures that each outfit can consist of multiple clothing pieces.
  + Enables customization of outfits using stored wardrobe items.

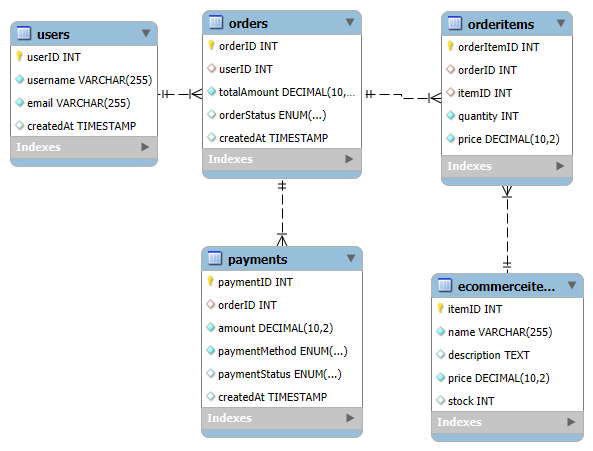
**12) Collections Table (Previously OutfitCollection)**

* **Purpose:** Allows users to create fashion collections (e.g., "Summer Looks", "Business Casual").
* **How It Works:**
  + **collectionID** uniquely identifies each collection.
  + **collectionName** allows naming of the collection.
  + **userID** links the collection to a specific user.
  + **createdDate** tracks when the collection was made.
* **Why It’s Important:**
  + Helps users organize outfits into themed groups.
  + Allows users to store and revisit curated looks.

**13) CollectionOutfits Table (NEW)**

* **Purpose:** Links outfits to specific collections (Many-to-Many).
* **How It Works:**
  + **collectionID** references a collection.
  + **outfitID** references an outfit.
* **Why It’s Important:**
  + Allows outfits to belong to multiple collections.
  + Enables users to group outfits by style or season.

**Database 2:** Transactions (orders, purchases) are in one SQL database, ensuring structured consistency.

****

1. **Users Table**

* **Purpose**: Stores information about users who place orders.
* **How It Works**:
  + Each user has a **unique userID**.
  + The **email** is unique for each user.
  + createdAt records the timestamp of when the user was added to the system.
  + Users in this table can **place orders and make payments**.
* **Why It’s Important**:
  + Serves as the **central reference** for user purchases and transactions.
  + Ensures users can be **tracked across orders and payments**.

1. **Orders Table**

* **Purpose**: Stores information about **user purchases**.
* **How It Works**:
  + Each order has a **unique orderID**.
  + userID links the order to a **specific user**.
  + totalAmount stores the **total price of the order**.
  + orderStatus tracks whether the order is **Pending, Shipped, Delivered, or Cancelled**.
  + createdAt records when the order was placed.
* **Why It’s Important**:
  + Ensures each order is **linked to a user**.
  + Tracks **order status** to facilitate **shipment and delivery management**.
  + Stores the **total amount** to be paid for each order.

1. **Ecommerce Items Table**

* **Purpose**: Stores **products available for purchase**.
* **How It Works**:
  + Each item has a **unique itemID**.
  + name and description store **product details**.
  + price indicates the **cost of the item**.
  + stock keeps track of **available inventory**.
* **Why It’s Important**:
  + Provides a **list of available items** for users to order.
  + Tracks **inventory levels** to prevent overselling.
  + Helps in **pricing calculations** for order totals.

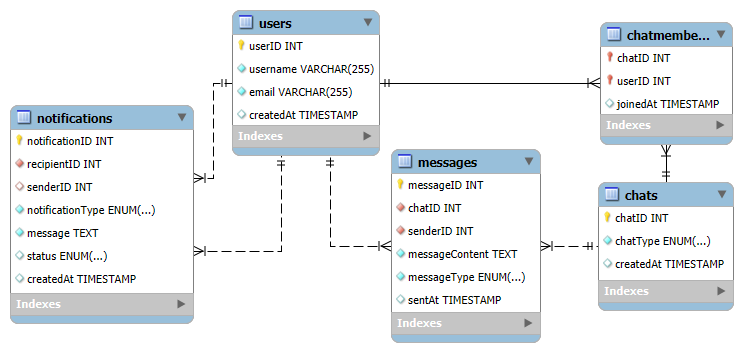
1. **Order Items Table (Many-to-Many Relationship: Orders ↔ Items)**

* **Purpose**: Connects **Orders** and **Ecommerce Items**, since **one order can contain multiple items, and one item can appear in multiple orders**.
* **How It Works**:
  + Each record has a **unique orderItemID**.
  + orderID links to the **Orders table**.
  + itemID links to the **Ecommerce Items table**.
  + quantity stores the **number of units** of an item in an order.
  + price captures the **cost per item** at the time of purchase.
* **Why It’s Important**:
  + Ensures each order can contain **multiple items**.
  + Tracks **how many units** of an item are purchased in each order.
  + Helps in **calculating the total cost of an order**.

1. **Payments Table**

* **Purpose**: Stores **payment transactions** for orders.
* **How It Works**:
  + Each payment has a **unique paymentID**.
  + orderID links the payment to a **specific order**.
  + amount represents the **payment total**.
  + paymentMethod tracks whether the payment was made using **Credit Card, PayPal, or Cryptocurrency**.
  + paymentStatus indicates whether the payment is **Pending, Completed, or Failed**.
  + createdAt records when the payment was made.
* **Why It’s Important**:
  + Ensures every order has an **associated payment record**.
  + Tracks **failed or pending transactions**.
  + Helps in **managing refunds and chargebacks**.

**Database 3:** Notifications and chats



1. **Users Table**

* **Purpose**: Stores basic user information for reference in chats and notifications.
* **How It Works**: Every user has a unique userID, username, and email.
* **Why It’s Important**: This table ensures that both the chat system and notification system can correctly identify users.

1. **Notifications Table**

* **Purpose**: Stores all real-time notifications (follows, likes, comments, messages, system alerts).
* **How It Works**:
  + Each notification has a unique notificationID.
  + recipientID is the user who will receive the notification.
  + senderID (optional) is the user who triggered the event (e.g., someone liked your post).
  + notificationType defines whether it’s a **Follow**, **Like**, **Comment**, **Message**, or **System** notification.
  + message contains the notification text (e.g., "John liked your post").
  + status shows whether the user has **Read** or **Unread** the notification.
  + createdAt records when the notification was sent.
* **Why It’s Important**: This table ensures that users receive alerts in real time about important events.

1. **Chats Table**

* **Purpose**: Stores information about all chat conversations.
* **How It Works**:
  + Each conversation has a unique chatID.
  + chatType defines whether the chat is **Private (One-to-One)** or **Group (Multiple users in one chat room)**.
  + createdAt records when the chat was created.
* **Why It’s Important**: This table is the foundation for all chat conversations. Without it, there would be no structure for messages.

1. **ChatMembers Table**

* **Purpose**: Connects users to chats. It defines which users are part of which chat.
* **How It Works**:
  + chatID links to a chat session.
  + userID links to a specific user who is part of the chat.
  + joinedAt records when the user joined the chat.
* **Why It’s Important**:
  + It allows **group chats** by linking multiple users to the same chatID.
  + It also ensures that only authorized users can send and receive messages in a specific chat.

1. **Messages Table**

* **Purpose**: Stores all chat messages sent by users.
* **How It Works**:
  + Each message has a unique messageID.
  + chatID links the message to a specific chat session.
  + senderID identifies which user sent the message.
  + messageContent stores the actual text, image, or media of the message.
  + messageType defines whether the message is **Text**, **Image**, **Video**, **GIF**, or **Sticker**.
  + sentAt records when the message was sent.
* **Why It’s Important**:
  + This table ensures that chat history is saved and accessible.
  + Different message types allow **rich media conversations** instead of just plain text.
  + Every message is linked to a **specific chat** so conversations stay organized.