**Key Entities and Attributes**

We identify the following main entities (with their key attributes) in the food-delivery database:

* **Account (User)**: Holds customer info. Attributes include AccountID (PK), FullName, Email, Phone, and credentials (e.g. PasswordHash). We allow multiple addresses and payment methods per account (see below).
* **Address**: If users can save several delivery addresses, we model a separate Address table. Attributes: AddressID (PK), AccountID (FK→Account), plus Street, City, State, ZipCode, etc. (One account can have many addresses.)
* **Category**: Groups products (e.g. “Organic”, “Snacks”, “Electronics”, etc.). Attributes: CategoryID (PK) and Name. Every product belongs to exactly one category, and each category can contain many products.
* **Product**: Items for sale. Attributes: ProductID (PK), Name, Description, Price, StockLevel (or Quantity in stock), ImageURL (optional), and CategoryID (FK→Category). We include a flag or quantity to indicate “in stock” status. Each product is linked to one Category.
* **Order**: Represents a customer’s placed order. Attributes: OrderID (PK), AccountID (FK→Account), OrderDate, DeliveryAddress (FK→Address or explicit text), DeliveryStatus (ENUM: e.g. “Preparing”, “Out for Delivery”, “Delivered”), PaymentStatus (ENUM: “Pending”, “Success”, “Failed”), and totals/amounts (TotalAmount). We typically store DeliveryAddress either by referencing one of the Account’s addresses or by copying an address at order time. Each order is placed by one account.
* **OrderItem** (junction table): Links Orders and Products to support many-to-many. Attributes: OrderItemID (PK), OrderID (FK→Order), ProductID (FK→Product), Quantity, and ItemPrice (price at time of order). One order can have many OrderItems, and each OrderItem refers to one product. We compute the order’s total by summing Quantity \* ItemPrice over its OrderItems.
* **PaymentInstrument**: Stores a user’s saved payment methods. Attributes: InstrumentID (PK), AccountID (FK→Account), Type (e.g. “Visa”, “MasterCard”), MaskedNumber (e.g. “\*\*\*\* \*\*\*\* \*\*\*\* 1234”), and ExpiryDate. Each account can link multiple instruments (cards) to use for online payments.
* **PaymentHistory**: Logs each payment transaction (for auditing). Attributes might include PaymentID (PK), OrderID (FK→Order), AccountID (FK→Account), InstrumentID (FK→PaymentInstrument, nullable if COD), Amount, PaymentDateTime, and Status (e.g. Success/Failed). An order may generate one or more payment records (e.g. retries), so this is one-to-many with Order (similar to a user making multiple payments).
* **CustomerServiceTicket**: Handles customer issues. Attributes: TicketID (PK), AccountID (FK→Account), OrderID (FK→Order, nullable if general), ServiceType (ENUM: “Order Status”, “Defective Product”, “Refund”, etc.), Status (ENUM: “Open”, “Resolved”, “Pending”), CreatedAt, and UpdatedAt. One account can open many tickets, and a ticket may be linked to a specific order.

**Relationships and Cardinalities**

From the above entities, the key relationships are:

* **Account – Address**: One-to-many. An account can have multiple saved addresses.
* **Account – Order**: One-to-many. Each order is placed by exactly one account, but an account can place many orders.
* **Account – PaymentInstrument**: One-to-many. A user can link multiple payment cards/instruments.
* **Account – PaymentHistory**: One-to-many. A user may have many payment records (transactions).
* **Category – Product**: One-to-many. Each product belongs to exactly one category, and each category can include multiple products.
* **Order – OrderItem**: One-to-many. Each order can have multiple order items, and each order item is associated with one order.
* **Product – OrderItem**: One-to-many. Each product can appear in many order items, each item includes one product. (Equivalently, Order–Product is many-to-many via OrderItem.)
* **Order – PaymentHistory**: One-to-many. An order may generate multiple payment records (e.g. partial payments or retries).
* **Order – Address**: Many-to-one (if order stores a FK to one address). Each order is delivered to one address (either chosen from the user’s addresses or entered at time).
* **CustomerServiceTicket – Account**: One-to-many. A user can have multiple support tickets.
* **CustomerServiceTicket – Order**: Many-to-one (optional). Each ticket is linked to at most one order when applicable.

These cardinalities align with standard e-commerce schemas. For example, “[38] shows that one user can place many orders (one-to-many) and one order can contain many items via a junction table”. Likewise, each product’s link to exactly one category is the usual pattern.

**ER Diagram Illustration**

*Figure: Example Entity-Relationship diagram for an e-commerce system (illustration).* The diagram above (from a typical e-commerce design) highlights that a user/account can have multiple addresses and payment methods, and that orders involve multiple products via an OrderItem table. In our design, we follow the same pattern: each **OrderItem** record joins one Order and one Product (capturing quantity and price). The **Account** entity (user) can link to many **Address** records and many **PaymentInstrument** records, reflecting that one user may save several delivery addresses and cards.

Each **Order** is placed by one Account (one-to-many) and may contain multiple OrderItems. We also include delivery-related fields (e.g. DeliveryStatus, DeliveredAt) and payment-status fields in the Order table, allowing dynamic updates of order state. The **PaymentHistory** table logs each transaction for an Order, enabling an audit of past payments beyond the order’s current status. The **CustomerServiceTicket** table links back to Account and (optionally) Order for issue tracking.