Step 1: Identify Problems in Raw Table

Based on the `orders_raw` sample, we have several major issues:

1. **Data Duplication (Redundancy):**
    - **Example:** John Smith's name and city are repeated three times.
    - **Problem:** If John moves to "Chicago," we have to update multiple rows. If we miss one, the data becomes inconsistent.
2. **Update Anomalies:**
    - If the price of a "Laptop" changes, we have to find every order containing a laptop and update it.
3. **Insert Anomalies:**
    - We cannot add a new product to our system until someone actually orders it, because we have nowhere to store product info except in an "order" row.
4. **Delete Anomalies:**
    - If we delete "Jane Doe's" only order (1002), we lose all record that Jane Doe ever existed as a customer.
5. **Partial Dependencies:**
    - `product_price` depends only on the `product_name`, not the `order_id`.
6. **Transitive Dependencies:**
    - `customer_city` depends on `customer_name`, which depends on `order_id`.

Step 2: First Normal Form (1NF)

Goal: Ensure atomic values and unique rows.

Changes Made:

- Ensured every cell has one value (no "Laptop, Mouse" in one cell).
- Identified that the unique "Key" for a row is actually a combination of `order_id` and `product_name`.

1NF Schema (`orders_1nf`): | order_id (PK) | product_name (PK) | customer_name | customer_city | product_price | quantity | | :--- | :--- | :--- | :--- | :--- | :--- | | 1001 | Laptop | John Smith | New York | 1200.00 | 1 | | 1001 | Mouse | John Smith | New York | 25.00 | 2 | | 1002 | Keyboard | Jane Doe | Los Angeles | 75.00 | 1 |

Step 3: Second Normal Form (2NF)

Goal: Remove **Partial Dependencies**. Non-key attributes must depend on the *whole* primary key.

In 1NF, `customer_name` depends only on `order_id`, while `product_price` depends only on `product_name`. We must split these out.

2NF Schema:

- **`customers`**: `customer_id (PK)`, `customer_name`, `customer_city`
- **`products`**: `product_id (PK)`, `product_name`, `product_price`
- **`orders`**: `order_id (PK)`, `customer_id (FK)`
- **`order_items`**: `order_id (PK, FK)`, `product_id (PK, FK)`, `quantity`

What changed? We created a "Junction Table" (`order_items`). Now, product information is stored once, and order info is stored once. The partial dependency is gone.

Step 4: Third Normal Form (3NF)

Goal: Remove **Transitive Dependencies**. Attributes should depend on "The Key, the Whole Key, and Nothing but the Key."

In our 2NF `customers` table, the `city` might be repeated for many customers. If we want to store city-specific data (like Sales Tax), it's better to isolate it.

3NF Schema:

1. **`cities`**: `city_id (PK)`, `city_name`
2. **`customers`**: `customer_id (PK)`, `customer_name`, `city_id (FK)`
3. **`products`**: `product_id (PK)`, `product_name`, `product_price`
4. **`orders`**: `order_id (PK)`, `customer_id (FK)`, `order_date`
5. **`order_items`**: `order_id (PK, FK)`, `product_id (PK, FK)`, `quantity`, `unit_price_at_sale`

Improvements:

- **Integrity:** We added `unit_price_at_sale` to `order_items`. This is a 3NF best practice because even if the *current* product price changes in the `products` table, the *historical* price paid by the customer is preserved.

Table	Primary Key	Foreign Keys	Purpose
cities	city_id	None	Single source of truth for locations.
customers	customer_id	city_id	Stores unique identity of users.
products	product_id	None	Catalog of items and current prices.
orders	order_id	customer_id	Header-level info for a transaction.
order_items	order_id, product_id	order_id, product_id	Line-level details (The "Many-to-Many" link).

Anomaly Resolution

- **Update:** Changing a product price now only requires 1 row update in `products`.
- **Delete:** Deleting an order no longer deletes the Customer from the `customers` table.
- **Insert:** We can add new cities or products before any orders exist.