SQL Project: Dukaan (Store) Ka Mukammal Database

The whole series in one project: a store's products, customers, orders and order items — schema, data, sales reports and stock alerts.

Introduction

Eleven lessons of tools. Now we build something real with all of them: the complete database of a small **dukaan** (store). The owner needs four things stored — **products** (what he sells), **customers** (who buys), **orders** (each purchase visit), and **order items** (the lines inside each purchase) — and three questions answered every evening: *how much did each order total? which product sold most? what is about to run out?*

Every idea from the series gets its moment: CREATE + types (DB.2), CRUD (DB.3), keys and design (DB.6), constraints (DB.7), JOINs (DB.5), filtering and sorting (DB.4), grouped report questions (DB.8's spirit — questions inside the question), and one view for the evening report (DB.11). Check the arithmetic by hand as you go — a project you cannot verify is just typing.

(Roman Urdu: Ye poori series ka imtihan hai — ek asli dukaan, chaar tables, aur shaam wale teen sawal.)

Key points

- **Design before typing:** the store's facts split into four one-topic tables. A product's name and price live once in `products`; an order stores *who* and *when*; the *what and how many* of each purchase line lives in `order_items`. Storing the product's price inside every order line *as well* would create the two-homes-one-fact trap from DB.6.
- **The linking shape:** `orders.customer_id` → `customers.id`; `order_items.order_id` → `orders.id`; `order_items.product_id` → `products.id`. Three foreign keys are the whole skeleton — every report below is a JOIN travelling on them.
- **GROUP BY is the report-maker:** `SUM(quantity * price)` per order, `SUM(quantity)` per product — GROUP BY folds many rows into one answer per group. It is the natural partner of JOINs in every sales report ever written.

  (Roman Urdu: JOIN tables ko jorta hai, GROUP BY jama karke jawab banata hai.)
- **An order's total is never stored:** it is *computed* from its items, every time. Store facts; compute answers — the DB.11 rule, applied.
- **Stock alerts are just WHERE:** "about to run out" = `stock < 30`. Business language translates to SQL one condition at a time; that translation *is* the developer's job.

Code examples

Example 1 — Build the dukaan: schema and data

```
CREATE TABLE products (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(60) NOT NULL,
  price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
  stock INT NOT NULL DEFAULT 0
);

CREATE TABLE customers (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  phone VARCHAR(15),
  city VARCHAR(50)
);

CREATE TABLE orders (
  id INT PRIMARY KEY AUTO_INCREMENT,
  customer_id INT NOT NULL,
  order_date DATE NOT NULL,
  FOREIGN KEY (customer_id) REFERENCES customers(id)
);

CREATE TABLE order_items (
  id INT PRIMARY KEY AUTO_INCREMENT,
  order_id INT NOT NULL,
  product_id INT NOT NULL,
  qty INT NOT NULL CHECK (qty > 0),
  FOREIGN KEY (order_id) REFERENCES orders(id),
  FOREIGN KEY (product_id) REFERENCES products(id)
);

INSERT INTO products (name, price, stock) VALUES
  ('Chawal 5kg', 850, 40), ('Cheeni 1kg', 160, 100),
  ('Aata 10kg', 1450, 25), ('Chai Patti 500g', 620, 60);

INSERT INTO customers (name, phone, city) VALUES
  ('Ahmed Raza', '0300-1112233', 'Sahiwal'),
  ('Fatima Bibi', '0321-4455667', 'Okara'),
  ('Rashid Mehmood', '0333-7788990', 'Sahiwal');

INSERT INTO orders (customer_id, order_date) VALUES
  (1, '2026-10-01'), (2, '2026-10-02'), (1, '2026-10-03');
```

```
INSERT INTO order_items (order_id, product_id, qty) VALUES
  (1, 1, 2), (1, 3, 1), (2, 2, 5), (3, 4, 2), (3, 1, 1);

SHOW TABLES;
```

What it does: Four tables with keys, foreign keys and the constraints that matter (prices never negative, quantities always above zero, names never empty). The data: 4 products, 3 customers, 3 orders — Ahmed bought twice (orders 1 and 3), Fatima once (order 2) — and 5 order lines between them.

What you will see:

```
Tables_in_shop
customers
order_items
orders
products
4 rows in set
```

The shop exists. Note order 1's lines: 2 × Chawal and 1 × Aata. Order 3's lines: 2 × Chai Patti and 1 × Chawal. Keep these in mind — the next example's totals must match them exactly.

Example 2 — The evening report: total of every order

```
SELECT o.id AS order_no, c.name AS customer,
  SUM(oi.qty * p.price) AS total
FROM orders o
JOIN customers c ON c.id = o.customer_id
JOIN order_items oi ON oi.order_id = o.id
JOIN products p ON p.id = oi.product_id
GROUP BY o.id, c.name
ORDER BY o.id;
```

What it does: Three joins walk the skeleton — order → its customer, order → its items, item → its product (for the price). GROUP BY folds each order's lines into one row, and SUM adds `qty × price` across them. Check by hand: order 1 = 2×850 + 1×1450 = 1700 + 1450 = **3150**; order 2 = 5×160 = **800**; order 3 = 2×620 + 1×850 = 1240 + 850 = **2090**.

What you will see:

```
+----------+---------------+---------+
| order_no | customer      | total   |
+----------+---------------+---------+
|        1 | Ahmed Raza    | 3150.00 |
|        2 | Fatima Bibi   |  800.00 |
|        3 | Ahmed Raza    | 2090.00 |
+----------+---------------+---------+
3 rows in set
```

Every total matches the hand-check to the rupee. This single query pattern — join the chain, group by the parent, sum the lines — is the beating heart of every sales dashboard, invoice screen and shop report you will ever build.

Example 3 — Best seller, and the stock alert

```
SELECT p.name, SUM(oi.qty) AS sold
FROM order_items oi
JOIN products p ON p.id = oi.product_id
GROUP BY p.name
ORDER BY sold DESC;

SELECT name, stock FROM products
WHERE stock < 30;

CREATE VIEW order_totals AS
SELECT o.id AS order_no, c.name AS customer,
  SUM(oi.qty * p.price) AS total
FROM orders o
JOIN customers c ON c.id = o.customer_id
JOIN order_items oi ON oi.order_id = o.id
JOIN products p ON p.id = oi.product_id
GROUP BY o.id, c.name;
```

What it does: The first query ranks products by total quantity sold: Cheeni 5 (all in Fatima's one order), Chawal 3 (2 + 1 across Ahmed's two orders), Chai Patti 2, Aata 1. The second is the stock alert — one honest WHERE. The third saves Example 2's report as a **view** (DB.11), so tomorrow evening the owner types `SELECT * FROM order_totals;` and nothing more.

What you will see:

```
+-----------------+------+
| name            | sold |
+-----------------+------+
| Cheeni 1kg      |    5 |
| Chawal 5kg      |    3 |
| Chai Patti 500g |    2 |
| Aata 10kg       |    1 |
+-----------------+------+
4 rows in set

+-----------+-------+
| name      | stock |
+-----------+-------+
| Aata 10kg |    25 |
+-----------+-------+
1 row in set
```

The alert names exactly one product: Aata, 25 bags, below the 30 line — and it is also the slowest seller, so the owner's real decision (reorder small, or stop?) is an informed one. That is what a database is *for*: not storing the shop, but answering it.

Quick recap

- Four tables, three foreign keys: **products, customers, orders, order_items** — one topic each, facts stored once, totals computed, never stored.
- The report pattern of a lifetime: **JOIN the chain → GROUP BY the parent → SUM the lines.**
- Best-seller and stock-alert questions are the same tools pointed at business questions.
- The evening report becomes a **VIEW**, so the hard query is written once and asked forever.

Common mistakes

- **Storing the order total as a column:** the day an item is corrected, the stored total and the real items disagree — and nobody knows which is true. Compute totals; store only facts.
- **Putting customer details inside orders:** name and phone copied onto every order means Ahmed's new phone number must be fixed in a dozen places. Customers live in `customers`; orders carry the id (DB.6, rule 2).
  (Roman Urdu: Gahak ki tafseel ek jagah, order mein sirf us ka number — warna har tabdeeli das jagah karni padegi.)
- **Skipping the hand-check:** a report query nobody verified is a rumour with decimals. The first run of every report gets checked against two or three hand-computed answers, like Example 2's.
- **Forgetting the qty CHECK:** without `CHECK (qty > 0)`, a line of −3 quietly *reduces* an order's total in the report. Project data deserves project-grade constraints (DB.7).

Practice

1. Add a fourth order: Rashid buys 1 × Aata and 2 × Cheeni on 2026-10-04. Write all the INSERTs (order first, then its items), then state order 4's total before running the report query — and check.
2. Write the query for each customer's **total spending** across all orders (customer name + one sum), biggest spender first. Who tops the list after task 1's order lands?
3. The owner asks: "which products has Fatima bought, and how many of each?" Write the query (join the chain, filter by customer, group by product). Then turn it into a view named `customer_purchases` — what changes in your query, and what stays the same?

Practice quiz

1. In the Dukaan database, what are the three foreign keys of the skeleton?

- products.id, customers.id, orders.id
- orders.customer_id to customers.id, order_items.order_id to orders.id, order_items.product_id to products.id
- Only order_items.order_id
- There are no foreign keys, the tables are separate

Answer: orders.customer_id to customers.id, order_items.order_id to orders.id, order_items.product_id to products.id

2. Should an order's total be stored as a column?

- Yes, storing it is faster and always correct
- No, it is computed from its items every time with SUM(quantity * price) grouped by order; store facts, compute answers, or a corrected item and the stored total disagree
- Yes, but only for big orders
- Only if the order has one item

Answer: No, it is computed from its items every time with SUM(quantity * price) grouped by order; store facts, compute answers, or a corrected item and the stored total disagree

3. In Dukaan language, what does "about to run out" translate to in SQL?

- stock = 0
- stock < 30, a stock alert is just a WHERE condition
- stock > 30
- ORDER BY stock

Answer: stock < 30, a stock alert is just a WHERE condition

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/sql-project-dukaan · More free tools and notes at https://scopetreasure.com