

Set 5 Solutions

Database: **shopdb**

Table 1: **customers**

Columns:

- **customer_id** → Primary Key, auto-increment.
- **first_name** → Cannot be NULL.
- **last_name** → Cannot be NULL.
- **email** → Must be unique, cannot be NULL.
- **phone** → Must be unique, cannot be NULL.
- **city** → Optional.
- **created_at** → Cannot be NULL (stores date when customer was created).

Constraints:

- **customer_id** is **Primary Key**.
 - **email** is **Unique**.
 - **phone** is **Unique**.
-

Table 2: **products**

Columns:

- **product_id** → Primary Key, auto-increment.
- **name** → Cannot be NULL.
- **category** → Cannot be NULL.
- **price** → Cannot be NULL, must be greater than 0.
- **stock** → Cannot be NULL, must be greater than or equal to 0.

Constraints:

- **product_id** is **Primary Key**.
 - Add a **Check Constraint** for **price > 0**.
 - Add a **Check Constraint** for **stock >= 0**.
-

Table 3: **employees**

Columns:

- **employee_id** → Primary Key, auto-increment.
- **first_name** → Cannot be NULL.
- **last_name** → Cannot be NULL.
- **role** → Cannot be NULL.
- **hire_date** → Cannot be NULL.
- **salary** → Cannot be NULL, must be greater than 0.

Constraints:

- `employee_id` is **Primary Key**.
- Add a **Check Constraint** for `salary > 0`.

Table 4: `orders`

Columns:

- `order_id` → Primary Key, auto-increment.
- `customer_id` → Foreign Key referencing `customers(customer_id)`.
- `product_id` → Foreign Key referencing `products(product_id)`.
- `quantity` → Cannot be NULL, must be greater than 0.
- `order_date` → Cannot be NULL.
- `status` → Cannot be NULL (values like `Pending`, `Shipped`, `Delivered`, `Cancelled`).
- `total` → Cannot be NULL, must be greater than 0.

Constraints:

- `order_id` is **Primary Key**.
- `customer_id` is **Foreign Key** (links to `customers`).
- `product_id` is **Foreign Key** (links to `products`).
- Add a **Check Constraint** for `quantity > 0`.
- Add a **Check Constraint** for `total > 0`.

Sample Data (4–5 rows per table)

`customers`

customer_id	first_name	last_name	email	phone	city	created_at
1	Rohan	Mehta	rohan@gmail.com	9876543210	Mumbai	2024-01-05
2	Priya	Sharma	priya@gmail.com	9123456780	Pune	2024-01-10
3	Aarav	Patel	aarav@gmail.com	9988776655	Surat	2024-02-15
4	Neha	Singh	neha@gmail.com	9112233445	Thane	2024-03-20
5	Karan	Desai	karan@gmail.com	9001122334	Ahmedabad	2024-04-01

`products`

product_id	name	category	price	stock
1	Laptop	Electronics	55000.0	15
2	Office Chair	Furniture	4500.0	40
3	T-Shirt	Clothing	799.0	100
4	Mixer Grinder	Kitchen	3200.0	25
5	Cricket Bat	Sports	2500.0	30

`employees`

employee_id	first_name	last_name	role	hire_date	salary
1	Ankit	Joshi	Sales	2023-01-15	30000.00

employee_id	first_name	last_name	role	hire_date	salary
2	Sneha	Kapoor	Delivery	2023-02-10	22000.00
3	Ramesh	Iyer	Manager	2022-12-01	45000.00
4	Pooja	Shetty	Support	2023-03-20	25000.00
5	Vishal	Choudhary	Accountant	2023-04-05	28000.00

orders

order_id	customer_id	product_id	quantity	order_date	status	total
1	1	1	1	2024-05-01	Delivered	55000.0
2	2	3	2	2024-05-02	Shipped	1598.0
3	3	5	1	2024-05-05	Pending	2500.0
4	4	2	3	2024-05-07	Delivered	13500.0
5	5	4	2	2024-05-08	Cancelled	6400.0

👉 Tasks:

1. Write `CREATE DATABASE` , `CREATE TABLE` queries with the above columns and constraints.
2. Insert the 5 rows shown above into each table.
3. Insert more rows (up to 30) with different realistic values.
4. Practice `INSERT` , `UPDATE` , `ALTER` .

Questions

1. Find the **top 3 customers by total spending** (JOIN + GROUP BY + ORDER BY).
2. Show products that have **never been ordered** (subquery + NOT IN).
3. Find orders where the **total is above the overall average order total** (subquery + filter).
4. Display each customer's **most recent order** (correlated subquery + join).
5. Find employees earning more than the **average salary within their own role** (correlated subquery).

Solutions

Set 5 – Mixed Business Insights

1. Find the top 3 customers by total spending.

```
SELECT c.name, SUM(oi.quantity * p.price) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY c.customer_id
ORDER BY total_spent DESC
LIMIT 3;
```

2. Show products never ordered.

```
SELECT p.product_name
FROM products p
```

```
WHERE p.product_id NOT IN (SELECT DISTINCT product_id FROM order_items);
```

3. Find orders above the average order total.

```
SELECT o.order_id, SUM(oi.quantity * p.price) AS order_total
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY o.order_id
HAVING order_total > (
    SELECT AVG(order_totals.total_amount)
    FROM (
        SELECT SUM(oi.quantity * p.price) AS total_amount
        FROM order_items oi
        JOIN products p ON oi.product_id = p.product_id
        GROUP BY oi.order_id
    ) AS order_totals
);
```

4. Display each customer's most recent order.

```
SELECT c.name, o.order_id, o.order_date
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_date = (
    SELECT MAX(o2.order_date)
    FROM orders o2
    WHERE o2.customer_id = c.customer_id
);
```

5. Find employees earning more than the average salary of their role.

```
SELECT e.name, e.role, e.salary
FROM employees e
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e2.role = e.role
);
```