```
use mydb;

-- 1. Aggregation & Filtering
-- Given a Sales table with columns:
-- sale_id (INT, PRIMARY KEY),
-- product_name (VARCHAR),
-- quantity (INT),
-- price (DECIMAL),
-- sale_date (DATE)

create table Sales (
    sale_id int primary key,
    product_name varchar(50),
    quantity int,
    price decimal(10,2),
    sale_date date
);

insert into sales values
(1, 'Laptop', 10, 800.00, '2024-03-01'),
(2, 'Smartphone', 15, 600.00, '2024-03-02'),
(3, 'Tablet', 8, 300.00, '2024-03-03'),
(4, 'Headphones', 20, 100.00, '2024-03-04'),
(5, 'Smartwatch', 6, 250.00, '2024-03-05');

-- Write a SQL query to:
-- Find the top 3 best-selling products (based on total revenue = quantity * price).
-- Only include products that have been sold more than 5 times.
select product_name, sum(price * quantity) as total_revenue from sales
group by product_name
having sum(quantity) > 5
order by total_revenue desc
limit 3;



-- 2. Subqueries & Joins
-- You have two tables:
-- Employees (id, name, department_id, salary),
-- Departments (department_id, department_name).

create table departments (
    department_id int primary key,
    department_name varchar(50)
);

insert into departments values
(1, 'Engineering'),
(2, 'Marketing'),
(3, 'Sales');

create table employees (
    id int primary key,
    name varchar(50),
    department_id int,
```

```
    salary decimal(10,2),
    foreign key (department_id) references Departments(department_id)
);

insert into employees values
(1, 'Alice', 1, 90000.00),
(2, 'Bob', 1, 95000.00),
(3, 'Charlie', 2, 70000.00),
(4, 'David', 3, 80000.00),
(5, 'Eve', 3, 85000.00);

-- Write a SQL query to retrieve the name of the highest-paid employee in each department.
 select e.name, sum(e.salary) as high_Salary from employees e
 join departments d on e.department_id = d.department_id
 group by e.name
 order by high_salary desc
 limit 1;

-- 3. Foreign Key Constraint & Handling Deletions
-- You are designing a database with the following tables:
-- Customers (customer_id, name, email)
-- Orders (order_id, customer_id, order_date, total_amount), where customer_id is a foreign key referencing Customers(customer_id).

-- Write the SQL queries to:
-- 1. Create both tables with the necessary constraints.
create table customers(
 customer_id int auto_increment primary key,
 name varchar(50),
 email varchar(50));

create table orders (
 order_id int,
 customer_id int,
 order_date date,
 foreign key (customer_id) references customers(customer_id));

-- 2. Prevent the deletion of a customer if they have existing orders, and explain how you would handle this scenario if deletion is required.
-- answer:
-- since orders referenced to customers first orders table should be deleted then customers if
-- we try to delete customers it would cause an error since it referenced to orders.

-- 4. Recursive CTE for Hierarchical Data
-- Consider the Employees table with:
-- id (INT, PRIMARY KEY),
-- name (VARCHAR),
-- manager_id (INT, FOREIGN KEY referencing id).
alter table employees
add column manager_id int,
add constraint fk_manager foreign key (manager_id) references employees(id);
```

```
insert into employees (id, name, manager_id) values
(6, 'Frank', 4),
(7, 'Grace', 5);

-- Write a recursive CTE to find all employees who report (directly or indirectly) to a
specific manager (e.g., manager_id = 3).

with recursive emp_cte as (
-- direct report
    select id, name, department_id, salary, manager_id
    from employees
    where manager_id = 3

    union all

    select e.id, e.name, e.department_id, e.salary, e.manager_id
    from employees e
    join emp_cte ec on e.manager_id = ec.id
)
select * from emp_cte;

-- 5. Complex Filtering & Grouping with HAVING
-- Given a Payments table with columns:
-- payment_id (INT, PRIMARY KEY),
-- customer_id (INT, FOREIGN KEY),
-- amount (DECIMAL),
-- payment_date (DATE).
create table payments (
    payment_id int primary key,
    customer_id int,
    amount decimal(10,2),
    payment_date date,
    foreign key (customer_id) references customers(customer_id));

-- Write a SQL query to:
-- Find customers who have made more than 3 payments in the last 6 months.
-- The total sum of their payments should be greater than $500.
select customer_id
from payments
where payment_date >= date_sub(curdate(), interval 6 month)
group by customer_id
having count(payment_id) > 3 and sum(amount) > 500;
```