SQL Case Studies by Data In Motion, LLC

Overview:

I have used PostgreSQL database in these case studies. I have been exposed to the following areas of SQL:

- Basic aggregations
- CASE WHEN statements
- Window functions
- Joins
- Date time functions
- CTEs

Case Study 1: Tiny Shop Sales

This case study focuses on analyzing the sales of Tiny Shop over time in PostgreSQL.

Tables used:

- Customers – information about the customers
- Products – information about the products
- Orders – information at order level
- Order_Items – information at item level in each order

```
CREATE TABLE customers (
    customer_id integer PRIMARY KEY,
    first_name varchar(100),
    last_name varchar(100),
    email varchar(100)
);

CREATE TABLE products (
    product_id integer PRIMARY KEY,
    product_name varchar(100),
    price decimal
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    customer_id integer,
    order_date date
);

CREATE TABLE order_items (
    order_id integer,
    product_id integer,
    quantity integer
);
```

Queries:

```
--1) Which product has the highest price? Only return a single row.
select product_name, price from products where price = (select max(price) from
products);

--2) Which customer has made the most orders?
with cte as (
select o.customer_id, c.first_name||' '||c.last_name as customer_name,
count(o.order_id) as no_of_orders,
dense_rank() over (order by count(o.order_id) desc) as r from customers c
inner join orders o on c.customer_id = o.customer_id
group by o.customer_id, c.first_name||' '||c.last_name
)
select customer_id, customer_name, no_of_orders from cte where r = 1;

--3) What's the total revenue per product?
select o.product_id, sum(o.quantity * p.price) as revenue from order_items o
inner join products p on o.product_id = p.product_id
group by o.product_id order by o.product_id;

--4) Find the day with the highest revenue.
with cte as (
select min(TO_CHAR(o.order_date, 'DD/MM/YYYY')) as order_date, sum(oi.quantity
* p.price) as revenue,
dense_rank() over (order by sum(oi.quantity * p.price) desc) as r from orders
o inner join order_items oi on o.order_id = oi.order_id
inner join products p on oi.product_id = p.product_id group by o.order_date
)
select order_date, revenue from cte where r = 1;

--5) Find the first order (by date) for each customer.
select customer_id, min(TO_CHAR(order_date, 'DD/MM/YYYY')) as first_order_date
from orders group by customer_id order by customer_id;

--6) Find the top 3 customers who have ordered the most distinct products
select o.customer_id, c.first_name||' '||c.last_name as customer_name,
count(distinct oi.product_id) as unique_products
from customers c inner join orders o on c.customer_id = o.customer_id inner
join order_items oi on o.order_id = oi.order_id
group by o.customer_id, c.first_name||' '||c.last_name order by count(distinct
oi.product_id) desc limit 3;

--7) Which product has been bought the least in terms of quantity?
select a.product_id, p.product_name, a.total_quantity from (select product_id,
sum(quantity) as total_quantity,
```

```
dense_rank() over (order by sum(quantity)) as rank from order_items group by
product_id) a inner join products p on a.product_id = p.product_id
where rank = 1 order by a.product_id;

--8) What is the median order total?
with order_totals as (
select row_number() over(order by sum(o.quantity * p.price)) as row_id,
o.order_id, sum(o.quantity * p.price) as order_price,
(select count(1) from orders) as ct
from order_items o inner join products p on o.product_id = p.product_id group
by o.order_id
)
select round(avg(order_price),2) as median from order_totals where row_id
between ct/2.0 and (ct/2.0) + 1;

--9) For each order, determine if it was 'Expensive' (total over 300),
'Affordable' (total over 100), or 'Cheap'.
select o.order_id, sum(o.quantity * p.price) as order_price,
CASE
when sum(o.quantity * p.price) > 300 then 'Expensive'
when sum(o.quantity * p.price) <= 100 then 'Cheap'
else 'Affordable' end as purchase_type
from order_items o inner join products p on o.product_id = p.product_id group
by o.order_id order by o.order_id;

--10) Find customers who have ordered the product with the highest price.
select c.customer_id, c.first_name||' '||c.last_name as customer_name,
p.product_name, p.price from customers c inner join orders o
on c.customer_id = o.customer_id inner join order_items oi on o.order_id =
oi.order_id inner join products p on oi.product_id = p.product_id
where p.price = (select max(price) from products) order by c.customer_id;
```

Insights derived:

- The most expensive product is Product M and is bought by the customers Ivy Jones and Sophia Thomas.
- Customers named John Doe, Jane Smith, Bob Johnson made the most orders.
- The highest revenue was collected on 16/05/2023.
- Customers named Jane Smith, Bob Johnson, John Doe have ordered the most distinct products.
- The products Product D, Product E, Product G, Product H, Product I, Product K, Product L are bought the least in terms of quantity.
- The median order total is 112.50
- There are 8 Cheap orders, 7 Affordable and 1 Expensive orders placed.

Case Study 2: Human Resources

This case study focuses on analyzing data related to employees in the department and the ongoing projects.

Tables used:

- Departments – information about the departments
- Employees – information about the employees
- Projects – information about the projects

```
CREATE TABLE departments (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    manager_id INT
);

CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    hire_date DATE,
    job_title VARCHAR(50),
    department_id INT REFERENCES departments(id)
);

CREATE TABLE projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    start_date DATE,
    end_date DATE,
    department_id INT REFERENCES departments(id)
);
```

Queries:

```
--1. Find the longest ongoing project for each department.
with cte as (
select p.name as project_name, p.start_date, p.end_date,
(p.end_date - p.start_date) as duration_in_days, dense_rank() over (partition
by d.id order by (p.end_date - p.start_date) desc) as rank,
p.department_id, d.name as department_name
from projects p inner join departments d on p.department_id = d.id
)
select project_name, start_date, end_date, duration_in_days, department_id,
department_name from cte where rank = 1;
```

```sql
--2. Find all employees who are not managers.
select id, name from employees where id not in (select id from departments);

--3. Find all employees who have been hired after the start of a project in
their department.
select e.name as emp_name, e.job_title, TO_CHAR(e.hire_date, 'DD/MM/YYYY') as
hire_date, p.name as project_name,
TO_CHAR(p.start_date, 'DD/MM/YYYY') as project_start_date, p.department_id
from employees e inner join projects p
on e.department_id = p.department_id where e.hire_date > p.start_date;

--4. Rank employees within each department based on their hire date (earliest
hire gets the highest rank).
select id, name as emp_name, department_id, hire_date,
dense_rank() over (partition by department_id order by hire_date) as rank from
employees;

--5. Find the duration between the hire date of each employee and the hire
date of the next employee hired in the same department.
with cte as (
select department_id, name as emp_name, hire_date,
lead(name) over (partition by department_id order by hire_date) as
next_hired_emp_name,
lead(hire_date) over (partition by department_id order by hire_date) as
next_hire_date from employees
)
select *, (next_hire_date - hire_date) as duration_in_days from cte;
```

Insights derived:

- HR Project 1, IT Project 1, Sales Project 1 are the longest ongoing projects in HR, IT, Sales departments respectively.
- Employees Bob Miller, Charlie Brown, Dave Davis are not managers.
- Employee Dave Davis have been hired after the start of a project in their department.