Task 1: Online Bookstore Sales Contest

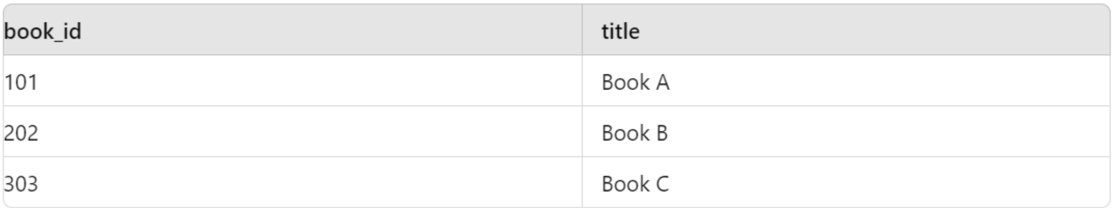
An online bookstore conducted a 15-day sales contest from April 1, 2024, to April 15, 2024. Participants' data is stored in two tables:

- Books Table: Contains `book_id` (ID of the book) and `title` (title of the book).

- Sales Table: Contains `sale_date` (date of the sale), `sale_id` (ID of the sale), `book_id` (ID of the book sold), and `quantity` (number of copies sold).

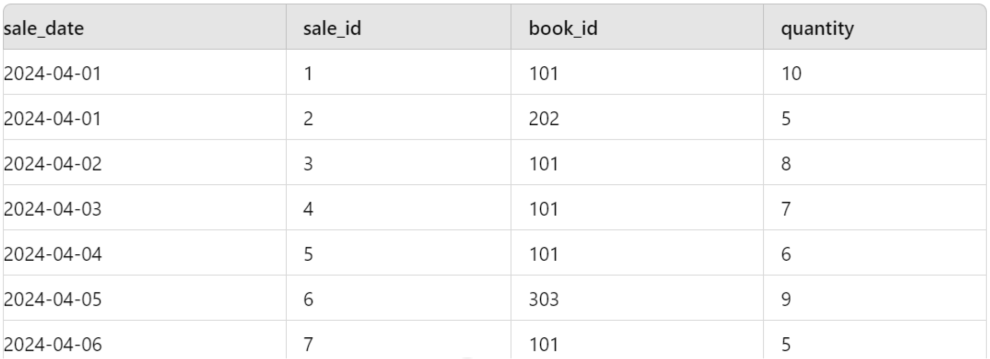
Sample Data:

Books Table:



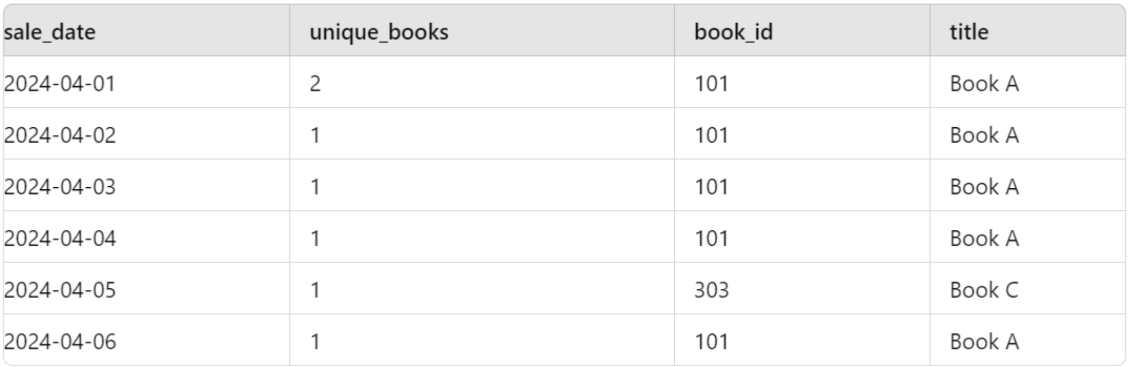
book_id	title
101	Book A
202	Book B
303	Book C

Sales Table:



sale_date	sale_id	book_id	quantity
2024-04-01	1	101	10
2024-04-01	2	202	5
2024-04-02	3	101	8
2024-04-03	4	101	7
2024-04-04	5	101	6
2024-04-05	6	303	9
2024-04-06	7	101	5

Expected Output:



sale_date	unique_books	book_id	title
2024-04-01	2	101	Book A
2024-04-02	1	101	Book A
2024-04-03	1	101	Book A
2024-04-04	1	101	Book A
2024-04-05	1	303	Book C
2024-04-06	1	101	Book A

1. List all sales by date, including `sale_date`, `sale_id`, `book_id`, and `quantity`.
2. Count the number of unique books sold each day.
3. Find the `book_id` and `title` of the book with the maximum number of sales each day.
4. If multiple books have the same number of maximum sales, select the one with the lowest `book_id`.
5. Create a summary of the total sales across all days, including the count of total sales and the number of unique books sold.
6. Calculate the daily sales rate as the ratio of books sold to the total number of books available.
7. Analyze the trend of sales over the contest period to identify peaks and dips.
8. Identify books that made sales on all days of the contest.
9. Determine the number of days each book was sold (made at least one sale).
10. Count the number of sales made for each book each day.
11. Identify the highest quantity sold for any book each day.
12. List books that had multiple sales on any given day.
13. Find books with sales quantities below a certain threshold (e.g., 5 copies).
14. Provide a final summary of the contest, including total sales, unique books sold, and highest selling book.

Task 2: SmartBuy Database

We have a database example for SmartBuy website. This database includes tables for customers, orders, products, order details, employees, departments, projects, and employee projects.

Tables and Sample Data: Use these following SQL to generate the data and sample data.

- **Table 1: `customers`**

```
CREATE TABLE customers (
   customer_id INT PRIMARY KEY,
   customer_name VARCHAR(100),
   status VARCHAR(10)
);
```

```
INSERT INTO customers (customer_id, customer_name, status) VALUES
(1, 'Alice', 'active'),
(2, 'Bob', 'inactive'),
(3, 'Charlie', 'active'),
(4, 'Diana', 'active'),
(5, 'Eve', 'inactive');
```

- **Table 2: `orders`**

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    shipping_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

INSERT INTO orders (order_id, customer_id, order_date, shipping_date) VALUES
(1, 1, '2023-01-01', '2023-01-05'),
(2, 2, '2023-01-02', NULL),
(3, 3, '2023-01-03', '2023-01-06'),
(4, 4, '2023-01-04', NULL),
(5, 1, '2023-01-05', '2023-01-07');
```

- **Table 3: `products`**

```
CREATE TABLE products (
   product_id INT PRIMARY KEY,
   product_name VARCHAR(100)
);

INSERT INTO products (product_id, product_name) VALUES
(1, 'Laptop'),
(2, 'Smartphone'),
(3, 'Tablet'),
(4, 'Monitor'),
(5, 'Keyboard');
```

- **Table 4: `order_details`**

```
CREATE TABLE order_details (
   order_detail_id INT PRIMARY KEY,
   order_id INT,
   product_id INT,
   quantity INT,
   price DECIMAL(10, 2),
   FOREIGN KEY (order_id) REFERENCES orders(order_id),
   FOREIGN KEY (product_id) REFERENCES products(product_id)
);

INSERT INTO order_details (order_detail_id, order_id, product_id, quantity, price)
VALUES
(1, 1, 1, 1, 1000.00),
(2, 1, 2, 2, 500.00),
(3, 2, 3, 1, 300.00),
(4, 3, 1, 1, 1000.00),
(5, 3, 4, 2, 150.00),
(6, 4, 5, 3, 50.00),
(7, 5, 2, 1, 500.00);
```

- **Table 5: `employees`**

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100),
    manager_id INT,
    department_id INT,
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id),
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

INSERT INTO employees (employee_id, employee_name, manager_id, department_id) VALUES
(1, 'John', NULL, 1),
(2, 'Sara', 1, 1),
(3, 'Mike', 1, 2),
(4, 'Kate', 2, 1),
(5, 'Tom', 3, 2);
```

- **Table 6: `departments`**

```
CREATE TABLE departments (
   department_id INT PRIMARY KEY,
   department_name VARCHAR(100)
);

INSERT INTO departments (department_id, department_name) VALUES
(1, 'Sales'),
(2, 'Engineering'),
(3, 'HR');
```

- **Table 7: `projects`**

```
CREATE TABLE projects (
   project_id INT PRIMARY KEY,
   project_name VARCHAR(100)
);

INSERT INTO projects (project_id, project_name) VALUES
(1, 'Project A'),
(2, 'Project B'),
(3, 'Project C');
```

- **Table 7: `employee_projects`**

```
```

```
CREATE TABLE employee_projects (
    employee_id INT,
    project_id INT,
    PRIMARY KEY (employee_id, project_id),
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
    FOREIGN KEY (project_id) REFERENCES projects(project_id)
);

INSERT INTO employee_projects (employee_id, project_id) VALUES
(1, 1),
(2, 1),
(3, 2),
(4, 3),
(5, 2);
```

Needed Relations:

1. **customers and orders**: One-to-Many relationship. One customer can have multiple orders.
2. **orders and order_details**: One-to-Many relationship. One order can have multiple order details.
3. **products and order_details**: One-to-Many relationship. One product can appear in multiple order details.
4. **employees and departments**: Many-to-One relationship. Many employees can belong to one department.
5. **employees and projects**: Many-to-Many relationship. Many employees can work on many projects, which is resolved by the employee_projects table.

Task 1: Retrieve a list of all customers along with their corresponding orders.

Concepts to use: INNER JOIN, SELECT

Task 2: Get a list of all products and their associated order details, including products that haven't been ordered.

Concepts to use: LEFT JOIN, IS NULL

Task 3: Find all employees and their associated department names. Include departments with no employees.

Concepts to use: RIGHT JOIN, SELECT

Task 4: Calculate the total sales amount for each product.

Concepts to use: SUM, GROUP BY, INNER JOIN

Task 5: Retrieve a list of all orders with customer names, product names, and order quantities.

Concepts to use: Multiple JOINs, SELECT

Task 6: Find customers who have placed more than five orders.

Concepts to use: INNER JOIN, COUNT

Task 7: Get a list of all employees and the projects they are working on, including employees not assigned to any project and projects with no assigned employees.

Concepts to use: FULL OUTER JOIN, COALESCE

Task 8: Create a list of all active and inactive customers along with their orders, separating them using a status column.

Concepts to use: UNION, INNER JOIN, SELECT

Task 9: Retrieve a list of all orders with their status ('Shipped', 'Pending', 'Canceled') based on the order date and shipping date.

Concepts to use: CASE, INNER JOIN, SELECT