Week 5 Assessment Scenarios

Databricks scenario

✈️ Assessment Scenario: Flight Data Analytics with PySpark

Context

You are working as a **Data Engineer** for an aviation analytics company. The company wants to monitor live flight data using the **OpenSky API**. Your task is to build a PySpark pipeline that ingests streaming flight data and computes summary statistics.

The pipeline has two main tables:

1. **`ingest_flights`** → streams raw flight data from OpenSky.
2. **`flights_stats`** → aggregates statistics such as number of events, distinct aircraft, and maximum velocity.

Tasks for Learners

1. **Data Ingestion**
   - Register the OpenSky data source.
   - Create a streaming table `ingest_flights` that continuously loads flight data.
2. **Data Aggregation**
   - Build the `flights_stats` table to compute:
   - Total number of flight events (`num_events`)
   - Distinct aircraft identifiers (`distinct_aircraft`)
   - Maximum velocity observed (`max_velocity`)
3. **Data Validation**
   - Verify that the streaming ingestion is working by checking schema and sample rows.
   - Ensure that aggregation updates dynamically as new data arrives.
4. **Visualization**

- Plot the aggregated statistics to monitor flight activity.
- Example using **Matplotlib / Seaborn** after converting Spark DataFrame to Pandas:

Evaluation Criteria

- Correctness of ingestion pipeline setup.
- Accuracy of aggregation logic.
- Ability to visualize and interpret results.
- Clarity of code annotations and explanation.

GitHub Link:

Databricks_solution

Assessment Scenario 2 : Prompt Engineering

Develop a 30-day go-to-market strategy for a B2B SaaS tool targeting mid-size companies.

Draft an internal memo addressing a data breach incident, keeping staff informed without causing panic.

Github Link:

Prompt_solutions.pdf

Assessment Scenario 3: Transfer Learning for Land Cover Classification

Context

You are part of a geospatial AI team working with satellite imagery to monitor land use changes. Your task is to build a model that classifies satellite images into three categories:

- **Urban**
- **Forest**
- **Water**

Due to limited labeled data and compute resources, training a deep CNN from scratch is not practical.

Challenge

Apply **transfer learning** using a pre-trained ResNet50 model (trained on ImageNet) to classify satellite images into the three land cover types.

Github Link :

ResNet50.py

Assessment Scenario 4 for SQL for Enterprise Data Management

Core Queries & Joins • DBMS Operations • Query Optimization

E-Commerce Analytics & Order Management System

Table of Contents

Project Overview

Build a complete enterprise data management system for **"TechMart"** - an online electronics retailer. You'll design the database, write complex queries, optimize performance, and generate business insights.

Phase 1: Database Design & Schema Creation

Business Requirements

TechMart needs to track:

- Customers and their orders
- Product inventory across multiple warehouses
- Order fulfillment and shipping
- Employee performance
- Supplier relationships
- Product reviews and ratings

Schema Design

```
-- Create Database
CREATE DATABASE techmart_db;
USE techmart_db;
```

1. CUSTOMERS Table

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),
    registration_date DATE NOT NULL,
    loyalty_tier ENUM('Bronze', 'Silver', 'Gold', 'Platinum') DEFAULT
'Bronze',
    total_spent DECIMAL(10,2) DEFAULT 0,
    INDEX idx_email (email),
    INDEX idx_loyalty (loyalty_tier)
);
```

2. ADDRESSES Table (One-to-Many with Customers)

```
CREATE TABLE addresses (
    address_id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    customer_id INT NOT NULL,
    address_type ENUM('Billing', 'Shipping') NOT NULL,
    street_address VARCHAR(200) NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(50) NOT NULL,
    postal_code VARCHAR(20) NOT NULL,
    country VARCHAR(50) NOT NULL DEFAULT 'USA',
    is_default BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE
CASCADE,
    INDEX idx_customer (customer_id)
);
```

3. CATEGORIES Table

```
CREATE TABLE categories (
    category_id INT PRIMARY KEY AUTO_INCREMENT,
    category_name VARCHAR(100) NOT NULL,
    parent_category_id INT NULL,
    description TEXT,
    FOREIGN KEY (parent_category_id) REFERENCES categories(category_id)
);
```

4. SUPPLIERS Table

```
CREATE TABLE suppliers (
    supplier_id INT PRIMARY KEY AUTO_INCREMENT,
    supplier_name VARCHAR(100) NOT NULL,
    contact_name VARCHAR(100),
    email VARCHAR(100),
    phone VARCHAR(20),
    country VARCHAR(50),
    rating DECIMAL(3,2) CHECK (rating BETWEEN 0 AND 5),
    INDEX idx_rating (rating)
);
```

5. PRODUCTS Table

```
CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(200) NOT NULL,
    category_id INT NOT NULL,
    supplier_id INT NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL,
    cost_price DECIMAL(10,2) NOT NULL,
```

```
    stock_quantity INT DEFAULT 0,
    reorder_level INT DEFAULT 10,
    is_active BOOLEAN DEFAULT TRUE,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(category_id),
    FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id),
    INDEX idx_category (category_id),
    INDEX idx_price (unit_price),
    INDEX idx_stock (stock_quantity)
);
```

6. WAREHOUSES Table

```
CREATE TABLE warehouses (
    warehouse_id INT PRIMARY KEY AUTO_INCREMENT,
    warehouse_name VARCHAR(100) NOT NULL,
    location VARCHAR(100) NOT NULL,
    capacity INT NOT NULL,
    manager_name VARCHAR(100)
);
```

7. INVENTORY Table (Many-to-Many: Products & Warehouses)

```
CREATE TABLE inventory (
    inventory_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT NOT NULL,
    warehouse_id INT NOT NULL,
    quantity INT DEFAULT 0,
    last_restocked DATE,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (warehouse_id) REFERENCES warehouses(warehouse_id),
    UNIQUE KEY unique_product_warehouse (product_id, warehouse_id),
    INDEX idx_quantity (quantity)
);
```

8. ORDERS Table

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    shipping_address_id INT NOT NULL,
    order_status ENUM('Pending', 'Processing', 'Shipped', 'Delivered',
'Cancelled') DEFAULT 'Pending',
    total_amount DECIMAL(10,2) NOT NULL,
    discount_amount DECIMAL(10,2) DEFAULT 0,
    tax_amount DECIMAL(10,2) DEFAULT 0,
    payment_method ENUM('Credit Card', 'Debit Card', 'PayPal', 'Bank
Transfer') NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (shipping_address_id) REFERENCES addresses(address_id),
    INDEX idx_customer (customer_id),
    INDEX idx_order_date (order_date),
    INDEX idx_status (order_status)
);
```

9. ORDER_ITEMS Table

```
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL,
    discount DECIMAL(10,2) DEFAULT 0,
    subtotal DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    INDEX idx_order (order_id),
    INDEX idx_product (product_id)
);
```

10. SHIPMENTS Table

```
CREATE TABLE shipments (
    shipment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT NOT NULL,
    warehouse_id INT NOT NULL,
    shipment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    carrier VARCHAR(50),
    tracking_number VARCHAR(100),
    estimated_delivery DATE,
    actual_delivery DATE,
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (warehouse_id) REFERENCES warehouses(warehouse_id),
    INDEX idx_order (order_id)
);
```

11. EMPLOYEES Table

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    hire_date DATE NOT NULL,
    department ENUM('Sales', 'Support', 'Warehouse', 'Management') NOT
NULL,
    salary DECIMAL(10,2),
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id),
```

```
    INDEX idx_department (department)
);
```

12. REVIEWS Table

```
CREATE TABLE reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,
    order_id INT NOT NULL,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_verified_purchase BOOLEAN DEFAULT TRUE,
    helpful_count INT DEFAULT 0,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    INDEX idx_product (product_id),
    INDEX idx_rating (rating)
);
```

Phase 2: Sample Data Insertion

All sample data is ready to use. Simply copy and paste the INSERT statements below into your SQL client.

Insert Categories

```
INSERT INTO categories (category_name, parent_category_id, description)
VALUES
('Electronics', NULL, 'Electronic devices and accessories'),
('Computers', 1, 'Desktop and laptop computers'),
('Mobile Devices', 1, 'Smartphones and tablets'),
('Accessories', 1, 'Electronic accessories'),
('Audio', 1, 'Audio equipment and headphones');
```

Insert Suppliers

```
INSERT INTO suppliers (supplier_name, contact_name, email, phone, country,
rating) VALUES
('TechSupply Inc', 'John Smith', 'john@techsupply.com', '555-0101', 'USA',
4.5),
('GlobalElectronics', 'Maria Garcia', 'maria@global.com', '555-0102',
'China', 4.2),
('QuickShip Electronics', 'David Lee', 'david@quickship.com', '555-0103',
'Taiwan', 4.8),
('Premium Parts Co', 'Sarah Johnson', 'sarah@premiumparts.com', '555-0104',
'Germany', 4.6);
```

Insert Products

```
INSERT INTO products (product_name, category_id, supplier_id, unit_price,
cost_price, stock_quantity, reorder_level) VALUES
('Dell XPS 15 Laptop', 2, 1, 1499.99, 1100.00, 25, 5),
('MacBook Pro 14"', 2, 1, 1999.99, 1500.00, 15, 5),
('iPhone 15 Pro', 3, 2, 999.99, 750.00, 50, 10),
('Samsung Galaxy S24', 3, 2, 899.99, 680.00, 40, 10),
('Sony WH-1000XM5 Headphones', 5, 3, 349.99, 220.00, 60, 15),
('Logitech MX Master 3', 4, 3, 99.99, 65.00, 100, 20),
('USB-C Hub', 4, 4, 49.99, 25.00, 150, 30),
('Samsung 27" Monitor', 4, 2, 299.99, 200.00, 35, 10);
```

Insert Warehouses

```
INSERT INTO warehouses (warehouse_name, location, capacity, manager_name)
VALUES
('West Coast Hub', 'Los Angeles, CA', 10000, 'Mike Anderson'),
```

```
('East Coast Hub', 'New York, NY', 8000, 'Jennifer White'),
('Central Hub', 'Chicago, IL', 12000, 'Robert Brown');
```

Insert Inventory

```
INSERT INTO inventory (product_id, warehouse_id, quantity, last_restocked)
VALUES
(1, 1, 10, '2024-01-15'),
(1, 2, 15, '2024-01-20'),
(2, 1, 8, '2024-01-18'),
(3, 2, 25, '2024-01-22'),
(3, 3, 25, '2024-01-22'),
(4, 1, 20, '2024-01-20'),
(5, 2, 30, '2024-01-25'),
(6, 3, 50, '2024-01-10'),
(7, 1, 75, '2024-01-12'),
(8, 2, 20, '2024-01-28');
```

Insert Customers

```
INSERT INTO customers (first_name, last_name, email, phone,
registration_date, loyalty_tier, total_spent) VALUES
('Alice', 'Johnson', 'alice.j@email.com', '555-1001', '2023-06-15', 'Gold',
5200.00),
('Bob', 'Williams', 'bob.w@email.com', '555-1002', '2023-08-20', 'Silver',
2800.00),
('Carol', 'Davis', 'carol.d@email.com', '555-1003', '2024-01-10', 'Bronze',
450.00),
('David', 'Miller', 'david.m@email.com', '555-1004', '2023-03-05',
'Platinum', 12500.00),
('Emma', 'Wilson', 'emma.w@email.com', '555-1005', '2023-11-12', 'Silver',
1900.00);
```

Insert Addresses

```
INSERT INTO addresses (customer_id, address_type, street_address, city,
state, postal_code, is_default) VALUES
(1, 'Shipping', '123 Main St', 'Los Angeles', 'CA', '90001', TRUE),
(1, 'Billing', '123 Main St', 'Los Angeles', 'CA', '90001', TRUE),
(2, 'Shipping', '456 Oak Ave', 'New York', 'NY', '10001', TRUE),
(3, 'Shipping', '789 Pine Rd', 'Chicago', 'IL', '60601', TRUE),
(4, 'Shipping', '321 Elm St', 'Houston', 'TX', '77001', TRUE),
(5, 'Shipping', '654 Maple Dr', 'Phoenix', 'AZ', '85001', TRUE);
```

Insert Employees

```
INSERT INTO employees (first_name, last_name, email, hire_date, department,
salary, manager_id) VALUES
('James', 'Taylor', 'james.t@techmart.com', '2020-01-15', 'Management',
95000.00, NULL),
('Lisa', 'Anderson', 'lisa.a@techmart.com', '2021-03-20', 'Sales',
65000.00, 1),
('Tom', 'Martinez', 'tom.m@techmart.com', '2022-06-10', 'Support',
55000.00, 1),
('Amy', 'Thomas', 'amy.t@techmart.com', '2021-09-05', 'Warehouse',
48000.00, 1);
```

Insert Orders

```
INSERT INTO orders (customer_id, order_date, shipping_address_id,
order_status, total_amount, discount_amount, tax_amount, payment_method)
VALUES
(1, '2024-01-15 10:30:00', 1, 'Delivered', 1649.98, 50.00, 132.00, 'Credit
Card'),
(2, '2024-01-20 14:15:00', 3, 'Delivered', 1099.98, 0.00, 88.00, 'PayPal'),
(1, '2024-01-25 09:45:00', 1, 'Shipped', 449.98, 0.00, 36.00, 'Credit
Card'),
(3, '2024-02-01 16:20:00', 4, 'Processing', 2099.98, 100.00, 160.00, 'Debit
Card'),
(4, '2024-02-03 11:00:00', 5, 'Pending', 3499.95, 200.00, 264.00, 'Bank
Transfer');
```

Insert Order Items

```
INSERT INTO order_items (order_id, product_id, quantity, unit_price,
discount, subtotal) VALUES
(1, 1, 1, 1499.99, 50.00, 1449.99),
(1, 6, 2, 99.99, 0.00, 199.98),
(2, 4, 1, 899.99, 0.00, 899.99),
(2, 6, 2, 99.99, 0.00, 199.98),
(3, 5, 1, 349.99, 0.00, 349.99),
(3, 7, 2, 49.99, 0.00, 99.98),
(4, 2, 1, 1999.99, 100.00, 1899.99),
(4, 8, 1, 299.99, 0.00, 299.99),
(5, 1, 2, 1499.99, 100.00, 2799.98),
(5, 5, 2, 349.99, 0.00, 699.98);
```

Insert Shipments

```
INSERT INTO shipments (order_id, warehouse_id, shipment_date, carrier,
tracking_number, estimated_delivery, actual_delivery) VALUES
(1, 1, '2024-01-16 08:00:00', 'FedEx', 'FDX123456789', '2024-01-20', '2024-
01-19'),
(2, 2, '2024-01-21 09:30:00', 'UPS', 'UPS987654321', '2024-01-25', '2024-
01-24'),
(3, 2, '2024-01-26 10:15:00', 'USPS', 'USPS456789123', '2024-01-30', NULL);
```

Insert Reviews

```
INSERT INTO reviews (product_id, customer_id, order_id, rating,
review_text, helpful_count) VALUES
(1, 1, 1, 5, 'Excellent laptop! Fast delivery and great performance.', 12),
(6, 1, 1, 4, 'Good mouse, but a bit pricey.', 5),
```

```
(4, 2, 2, 5, 'Love this phone! Best upgrade ever.', 8),
(5, 1, 3, 5, 'Amazing sound quality. Worth every penny!', 15);
```

🔍 Phase 3: Core SQL Queries & Business Analytics

3.1 Basic SELECT with Filtering

Q1: Find all products below reorder level

```
SELECT
    p.product_name,
    p.stock_quantity,
    p.reorder_level,
    s.supplier_name,
    (p.reorder_level - p.stock_quantity) AS units_to_order
FROM products p
JOIN suppliers s ON p.supplier_id = s.supplier_id
WHERE p.stock_quantity < p.reorder_level
ORDER BY units_to_order DESC;
```

Q2: Find customers who haven't ordered in the last 30 days

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    c.email,
    c.loyalty_tier,
    MAX(o.order_date) AS last_order_date,
    DATEDIFF(CURDATE(), MAX(o.order_date)) AS days_since_order
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
HAVING MAX(o.order_date) < DATE_SUB(CURDATE(), INTERVAL 30 DAY)
    OR MAX(o.order_date) IS NULL
ORDER BY days_since_order DESC;
```

3.2 INNER JOINs - Transactional Queries

Q3: Complete order details with customer and product information

```
SELECT
    o.order_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    c.email,
    o.order_date,
    p.product_name,
    oi.quantity,
    oi.unit_price,
    oi.subtotal,
    o.order_status
```

```
FROM orders o
INNER JOIN customers c ON o.customer_id = c.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 o.order_date >= '2024-01-01'
ORDER BY o.order_date DESC, o.order_id;
```

Q4: Revenue by product category

```
SELECT
    cat.category_name,
    COUNT(DISTINCT oi.order_id) AS total_orders,
    SUM(oi.quantity) AS units_sold,
    SUM(oi.subtotal) AS total_revenue,
    AVG(oi.unit_price) AS avg_selling_price,
    SUM(oi.subtotal - (p.cost_price * oi.quantity)) AS profit
FROM categories cat
INNER JOIN products p ON cat.category_id = p.category_id
INNER JOIN order_items oi ON p.product_id = oi.product_id
INNER JOIN orders o ON oi.order_id = o.order_id
WHERE o.order_status != 'Cancelled'
GROUP BY cat.category_id, cat.category_name
ORDER BY total_revenue DESC;
```

3.3 LEFT/RIGHT JOINs - Finding Gaps

Q5: Products with no sales (LEFT JOIN)

```
SELECT
    p.product_id,
    p.product_name,
    p.stock_quantity,
    c.category_name,
    COUNT(oi.order_item_id) AS times_ordered
FROM products p
LEFT JOIN order_items oi ON p.product_id = oi.product_id
LEFT JOIN categories c ON p.category_id = c.category_id
GROUP BY p.product_id, p.product_name, p.stock_quantity, c.category_name
HAVING times_ordered = 0
ORDER BY p.stock_quantity DESC;
```

Q6: Customers without orders in 2024

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    c.email,
    c.registration_date,
    c.loyalty_tier,
    COUNT(o.order_id) AS order_count_2024
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
    AND YEAR(o.order_date) = 2024
GROUP BY c.customer_id
HAVING order_count_2024 = 0;
```

3.4 SELF JOINs - Hierarchical Data

Q7: Employee hierarchy - managers and their subordinates

```
SELECT
    e1.employee_id,
    CONCAT(e1.first_name, ' ', e1.last_name) AS employee_name,
    e1.department,
    CONCAT(e2.first_name, ' ', e2.last_name) AS manager_name,
    e2.department AS manager_department
FROM employees e1
LEFT JOIN employees e2 ON e1.manager_id = e2.employee_id
ORDER BY e2.employee_id, e1.employee_id;
```

3.6 Subqueries - Advanced Analytics

Q11: Find top customers (above average spending)

```
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    c.loyalty_tier,
    COUNT(o.order_id) AS total_orders,
    SUM(o.total_amount) AS total_spent,
    (SELECT AVG(total_amount) FROM orders) AS avg_order_value,
    SUM(o.total_amount) - (SELECT AVG(total_amount) FROM orders) *
COUNT(o.order_id) AS vs_average
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
HAVING total_spent > (SELECT AVG(total_spent) FROM customers)
ORDER BY total_spent DESC;
```

3.7 Window Functions - Advanced Analytics

Q14: Running total of daily revenue

```
SELECT
    DATE(order_date) AS order_day,
    COUNT(order_id) AS orders_count,
    SUM(total_amount) AS daily_revenue,
    SUM(SUM(total_amount)) OVER (ORDER BY DATE(order_date)) AS
running_total,
    AVG(SUM(total_amount)) OVER (ORDER BY DATE(order_date)
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg_3day
FROM orders
WHERE order_status != 'Cancelled'
GROUP BY DATE(order_date)
ORDER BY order_day;
```

Phase 4: Query Optimization

4.1 EXPLAIN and Query Analysis

```
-- Analyze a slow query
EXPLAIN SELECT
    p.product_name,
    COUNT(oi.order_item_id) AS times_ordered,
    SUM(oi.quantity) AS total_quantity
FROM products p
LEFT JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name
ORDER BY times_ordered DESC;
```

4.2 Index Optimization

```
-- Create composite indexes for frequent queries
CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);
CREATE INDEX idx_order_items_product_order ON order_items(product_id,
order_id);
CREATE INDEX idx_reviews_product_rating ON reviews(product_id, rating);

-- Analyze index usage
SHOW INDEX FROM orders;
SHOW INDEX FROM products;
```

4.3 Query Rewriting for Performance

```
-- SLOW: Using OR conditions
SELECT * FROM products
WHERE category_id = 2 OR category_id = 3 OR category_id = 5;

-- FAST: Using IN clause
SELECT * FROM products
WHERE category_id IN (2, 3, 5);

-- SLOW: Function on indexed column
SELECT * FROM orders
WHERE YEAR(order_date) = 2024;

-- FAST: Range condition
SELECT * FROM orders
WHERE order_date >= '2024-01-01' AND order_date < '2025-01-01';
```

🎯 Phase 5: Advanced DBMS Operations

5.1 Transactions and ACID Properties

```
-- Transaction Example: Process a new order
START TRANSACTION;

-- 1. Insert order
INSERT INTO orders (customer_id, shipping_address_id, order_status,
total_amount, payment_method)
VALUES (1, 1, 'Pending', 1549.98, 'Credit Card');

SET @new_order_id = LAST_INSERT_ID();

-- 2. Insert order items
INSERT INTO order_items (order_id, product_id, quantity, unit_price,
subtotal)
VALUES
    (@new_order_id, 1, 1, 1499.99, 1499.99),
    (@new_order_id, 7, 1, 49.99, 49.99);

-- 3. Update inventory
UPDATE products SET stock_quantity = stock_quantity - 1 WHERE product_id =
1;
UPDATE products SET stock_quantity = stock_quantity - 1 WHERE product_id =
7;

-- 4. Update customer total spent
UPDATE customers
SET total_spent = total_spent + 1549.98
WHERE customer_id = 1;

COMMIT;
```

5.4 Views for Data Security and Simplification

```
-- Create view for customer order summary (hides sensitive data)
CREATE VIEW vw_customer_order_summary AS
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    c.loyalty_tier,
    COUNT(DISTINCT o.order_id) AS total_orders,
    SUM(o.total_amount) AS lifetime_value,
    AVG(o.total_amount) AS avg_order_value,
    MAX(o.order_date) AS last_order_date
FROM customers c
```

```
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_status != 'Cancelled'
GROUP BY c.customer_id;

-- Use the view
SELECT * FROM vw_customer_order_summary
WHERE lifetime_value > 1000
ORDER BY lifetime_value DESC;
```

Phase 6: Business Intelligence Queries

Q18: Monthly sales dashboard

```
SELECT
    DATE_FORMAT(o.order_date, '%Y-%m') AS month,
    COUNT(DISTINCT o.order_id) AS total_orders,
    COUNT(DISTINCT o.customer_id) AS unique_customers,
    SUM(o.total_amount) AS revenue,
    AVG(o.total_amount) AS avg_order_value,
    SUM(CASE WHEN o.order_status = 'Cancelled' THEN 1 ELSE 0 END) AS
cancelled_orders,
    ROUND(SUM(CASE WHEN o.order_status = 'Cancelled' THEN 1 ELSE 0 END) *
100.0 / COUNT(*), 2)
        AS cancellation_rate
FROM orders o
GROUP BY DATE_FORMAT(o.order_date, '%Y-%m')
ORDER BY month DESC;
```

🎓 Phase 7: Practice Challenges

Challenge 1: Inventory Management

Write queries to:

1. 1. Identify products that need reordering across all warehouses
2. 2. Calculate the cost of restocking all products below reorder level
3. 3. Recommend warehouse transfers to balance inventory

Challenge 2: Customer Analytics

4. 1. Create a customer cohort analysis by registration month
5. 2. Calculate customer churn rate
6. 3. Identify customers likely to upgrade loyalty tiers

Challenge 3: Revenue Optimization

7. 1. Find the most profitable product combinations
8. 2. Analyze discount effectiveness on revenue
9. 3. Calculate revenue per warehouse

Challenge 4: Performance Tuning

10. 1. Optimize the slowest queries using EXPLAIN
11. 2. Create appropriate indexes
12. 3. Rewrite a subquery using JOINs for better performance

🎯 Learning Outcomes Checklist

After completing this project, you should be able to:

- Design normalized database schemas with proper relationships
- Write complex SELECT queries with multiple JOINs
- Use INNER, LEFT, RIGHT, and SELF JOINs appropriately
- Apply aggregate functions with GROUP BY and HAVING
- Write subqueries and correlated subqueries
- Use window functions for analytics
- Create and use CTEs for readable queries
- Optimize queries using EXPLAIN and indexes
- Implement transactions for data integrity
- Create stored procedures and triggers
- Build views for data abstraction
- Perform business intelligence analysis

🚀 Next Steps

13. Extend the project: Add more features like promotions, wish lists, or gift cards
14. Performance testing: Load test with larger datasets (100K+ orders)
15. Security: Implement role-based access control
16. Reporting: Create complex analytical dashboards
17. Real-world integration: Connect to a Python/Java application

Good luck with your SQL mastery journey! 🎉

Github Link :

SQL_queries_solution