Discount Rule Engine Using Scala Functional Programming

Prepared By : Ahmed Atef

Presented To : Eng. Youssef Etman

Problem Statement

A large retail store requires a rule engine that evaluates order transactions for discounts based on a set of qualifying rules. The discounts are calculated based on specific conditions such as product expiry, product type, special events, and quantity purchased.

Qualifying Rules and Calculation Rules

1. **Less than 30 days remaining for product expiry**
    - Discounts based on remaining days:
        - 29 days: 1% discount
        - 28 days: 2% discount
        - ...and so on.
2. **Cheese and wine products on sale**
    - Cheese: 10% discount
    - Wine: 5% discount
3. **Products sold on March 23rd**
    - Special discount: 50%
4. **More than 5 units of the same product**
    - 6-9 units: 5% discount
    - 10-14 units: 7% discount
    - More than 15 units: 10% discount

Discount Calculation

- Transactions that do not qualify for any discount receive a 0% discount.
- Transactions qualifying for multiple discounts receive the top 2 discounts, averaged.

Technical Approach

1. **Functional Programming**: Core logic implemented using pure functions, emphasizing immutability and predictability.
2. **Data Processing**:
    - Read order transactions from a CSV file.
    - Apply qualifying rules to determine discounts.
    - Calculate final prices after applying discounts.
3. **Database Interaction**:
    - Insert processed data into a database table.
    - Utilize JDBC for database connectivity.
4. **Logging**:
    - Log engine events in a file with timestamp and log level.
5. **Documentation**:
    - Ensure code is well-commented and adheres to functional programming principles.
    - Emphasize readability, clarity, and self-explanation of the codebase.

Implemented Discount Rules

1. **Day Remaining Qualifying Rule**:
    - Function: **less_than_30_qualifier_using_days_between**
    - Description: Checks if the remaining days for a product to expire is less than 30.
    - Functionality: Determines the number of remaining days and evaluates if it meets the qualifying condition.
2. **On-Sale Products Qualifying Rule**:
    - Function: **cheese_and_wine_qualifier**
    - Description: Identifies whether a product is eligible for discount based on being a wine or cheese product.
    - Functionality: Determines if the product name starts with "Wine" or "Cheese" and applies the appropriate discount.
3. **Special Discount for Products Sold on 23rd of March**:
    - Function: **products_sold_23_march_qualifier**
    - Description: Applies a special discount if a product is sold on the 23rd of March.
    - Functionality: Checks if the sold date matches the 23rd of March and applies the corresponding discount.
4. **Quantity of Products Sold**:
    - Function: **more_than_5_qualifier**
    - Description: Determines if the quantity sold exceeds 5 units for the same product.
    - Functionality: Evaluates the quantity sold and applies the discount based on predefined quantity ranges.

Database Interaction

- **Database Connection**:
    - Utilizes Oracle JDBC driver for database connectivity.
    - Inserts processed data into the **orders** table, including details such as order date, expiry date, product category, quantity, unit price, channel, payment method, discount, and total price.

Logging Mechanisms

- **Logging Engine Rule Interactions**:
    - Function: **log_event**
    - Description: Writes information about engine rule interactions into the **logs.txt** file.
    - Structure: Logs timestamp, log level, and message for each rule interaction event.

Database Table Structure

- **Orders Table**:
    - Columns: ORDER_DATE, EXPIRY_DATE, DAYS_TO_EXPIRY, PRODUCT_CATEGORY, PRODUCT_NAME, QUANTITY, UNIT_PRICE, CHANNEL, PAYMENT_METHOD, DISCOUNT, TOTAL_PRICE.

Usage

1. **Data Input**:
    - Ensure that Scala and necessary dependencies are installed.
    - Place the order data CSV file (TRX1000.csv) in the src/main/resources directory.

```
timestamp,product_name,expiry_date,quantity,unit_price,channel,payment_method
2023-04-18T18:18:40Z,Wine - White Pinot Grigio,2023-06-10,6,122.47,Store,Visa
2023-04-09T20:17:37Z,Pepper - Red Thai,2023-05-22,5,23.18,Store,Cash
2023-05-09T13:28:27Z,Wine - Chardonnay Mondavi,2023-08-04,7,189.74,Store,Cash
2023-04-09T02:56:43Z,External Supplier,2023-05-23,14,9.92,App,Visa
2023-03-27T17:55:57Z,Beans - Black Bean Preserved,2023-05-28,13,145.85,Store,Cash
2023-04-27T04:41:13Z,Beef - Inside Round,2023-07-18,9,247.62,Store,Visa
2023-05-02T11:40:53Z,Bread - Calabrese Baguette,2023-05-28,1,111.21,Store,Visa
2023-03-17T16:27:46Z,Zucchini - Mini Green,2023-05-05,15,146.75,Store,Visa
2023-03-22T16:45:21Z,Eggs - Extra Large,2023-04-08,16,48.05,App,Visa
2023-05-11T16:44:16Z,Pork - Liver,2023-07-20,17,191.13,Store,Visa
2023-04-04T14:52:06Z,Sesame Seed,2023-06-08,4,62.69,Store,Cash
2023-04-14T11:44:13Z,Tuna - Yellowfin,2023-05-23,9,183.3,Store,Cash
```

    - Update database connection details in the Database.scala file. -> Put the ODBC8 file
    - Run the Main.scala file to execute the application.
    - Check the logs for information about the execution process.
    - Verify that the orders are successfully inserted into the database.
    - Find the exported CSV file (orders_with_discounts.csv) with the results.

2. **Configuration**:
    - Update the database connection details (URL, username, password) in the application.
3. **Execution**:
    - Run the application to process order transactions, calculate discounts, and insert data into the database.
4. **Verification**:
    - Check the **orders** table in the database for inserted records with calculated discounts and total prices.
    - Review the **logs.txt** file for logged engine rule interactions and any error messages.

```
Timestamp: 2024-05-06T18:25:36.329766500Z   LogLevel: info  Message: Openning Writer
Timestamp: 2024-05-06T18:25:36.498047100Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.499047600Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.499047600Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.499047600Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.500049900Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.500049900Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.500049900Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.500049900Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.501046400Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.501553300Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.501553300Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.501553300Z   LogLevel: info  Message: Order Discount Counted
Timestamp: 2024-05-06T18:25:36.501553300Z   LogLevel: info  Message: Order Discount Counted
```

Dependencies

- Java JDBC for database connection.
- Scala standard library for file operations and data processing.

Future Improvements

- Enhance error handling and logging.
- Implement unit tests for critical components.
- Allow customization of discount rules and database connection details through configuration files.
- Optimize database interaction for better performance.