Challenge Dec-2024
Pricing Policy

A solution with DT5GL/SQL/Python
by Jack Jansonius – 23 December 2024

Problem Statement (from the website):

Challenge Dec-2024

Pricing Policy

A company sells Standard and Premium products manufactured in different countries. It uses a mixed bracket volume and cost policy for orders with multiple units:

Cost Accounting

- **1-99 units:** $10 per unit (Standard), $15 per unit (Premium)
- **100-499 units:** $9 per unit (Standard), $14 per unit (Premium)
- **500+ units:** $8 per unit (Standard), $13 per unit (Premium)

For products that originated not in the US or Canada, there is a handling fee of $50 per order. If the order includes hazardous material, the total price should be increased by 30%.

Here is an example of the order:

- 200 Premium products originated in Germany with hazardous materials
- 50 Standard products originated in the US without hazardous materials
- 200 Premium products originated in the Canada without hazardous materials
- 150 Standard products originated in China with hazardous materials.

Tables in the database:

SQLite:

Table: client_order

order_id	description
1	Order in the challenge
2	2x Order in the challenge
3	Order with 1 item
4	Order with some remaining test cases

Table: order_item

order_id	item_id	product_type	number_of_units	origin_country	hazardous
1	1	Premium	200	Germany	Yes
1	2	Standard	50	US	No
1	3	Premium	200	Canada	No
1	4	Standard	150	China	Yes
2	1	Premium	200	Germany	Yes
2	2	Standard	50	US	No
2	3	Premium	200	Canada	No
2	4	Standard	150	China	Yes
2	5	Premium	200	Germany	Yes
2	6	Standard	50	US	No
2	7	Premium	200	Canada	No
2	8	Standard	150	China	Yes
3	1	Standard	500	France	No
4	1	Premium	99	Canada	Yes
4	2	Premium	500	US	Yes
4	3	Premium	99	France	No
4	4	Premium	500	Germany	No

PostgreSQL: same

Implementation of the decision model in DT5GL/SQL/Python (Solution 1):

```
SQLite_database: "Database/PricingPolicy.sqlite3"
# PostgreSQL_database: "pricingpolicy"

Table 0:
If:                                          | 0| 1| 2| 3|
'Next Order'                                 | Y| N| -| -|
'Next Item in Order'                         | -| -| Y| N|
Then:
order is selected                            | X|  |  |  |
order is finished                            |  | X|  |  |
item is selected                             |  |  | X|  |
item is finished                             |  |  |  | X|
# .......

Proposition: 'Next Order'
Obtain_instance_from_database_view: order

Proposition: 'Next Item in Order'
Obtain_instance_from_database_view: orderItem


rTable 1:
If:                                  | 0| 1| 2|
orderItem.product_type = "Standard"  | Y| Y| Y|
orderItem.number_of_units < 100      | Y| N| N|
orderItem.number_of_units < 500      | -| Y| N|
Then:
price_per_unit = 10                  | X|  |  |
price_per_unit = 9                   |  | X|  |
price_per_unit = 8                   |  |  | X|
# .......

rTable 2:
If:                                  | 0| 1| 2|
orderItem.product_type = "Premium"   | Y| Y| Y|
orderItem.number_of_units < 100      | Y| N| N|
orderItem.number_of_units < 500      | -| Y| N|
Then:
price_per_unit = 15                  | X|  |  |
price_per_unit = 14                  |  | X|  |
price_per_unit = 13                  |  |  | X|
# .......


Attribute: handling_fee      Type: Integer
Equals: 0 if orderItem.origin_country in ["US", "Canada"] else 50

Attribute: surcharge         Type: Real
Equals: 1.3 if orderItem.hazardous == "Yes" else 1.0

Attribute: total_item_price  Type: Real
Equals: (orderItem.number_of_units * price_per_unit + handling_fee) * surcharge

Attribute: print_order_item  Type: Text
Equals: str(orderItem.number_of_units) + " x " + str(price_per_unit)  \
      + " " + " " + str(handling_fee) + " handling fee "[1]            \
      + ("+ 30% hazardous " if orderItem.hazardous == "Yes" else "")  \
      + "= $" + str(total_item_price)
```

[1] The text '0 handling fee' can be suppressed by:
+ ("+ " + str(handling_fee) + " handling fee " if handling_fee > 0 else "")

```
Database_view: order
With_attributes: id, description
Query:
 SELECT order_id, description
 FROM client_order
 LIMIT 1 OFFSET %s
With_arguments: order.auto_index

Database_view: orderItem
With_attributes: ordernr, itemnr, product_type, number_of_units, origin_country,
hazardous
Query:
SELECT *
 FROM order_item
 WHERE order_id = %s
 LIMIT 1 OFFSET %s
With_arguments: order.id, orderItem.auto_index

Attribute: orderItem.origin_country Type: Text
Attribute: orderItem.hazardous      Type: Text


GOALATTRIBUTE: order
Repeat_until: finished

Case: finished
Print: "All orders processed"

Case: selected
Print: "Order: %s - %s" order.id  order.description
>>: total_order_price = 0


GOALATTRIBUTE: item
Repeat_until: finished

Case: finished
Print: "Total order price: $%s"  total_order_price
Print: "-------------------------------------------------------"

Case: selected
Print: "%s" print_order_item
>>: total_order_price = total_order_price + total_item_price
```

Implementation of the decision model in DT5GL/SQL/Python (Solution 2):

```
# PostgreSQL_database: "pricingpolicy"
SQLite_database: "Database/PricingPolicy.sqlite3"

Table 0:
If:                                          | 0| 1| 2| 3|
'Next Order'                                 | Y| N| -| -|
'Next Item in Order'                         | -| -| Y| N|
Then:
order is selected                            | X|  |  |  |
order is finished                            |  | X|  |  |
item is selected                             |  |  | X|  |
item is finished                             |  |  |  | X|
# .......

Proposition: 'Next Order'
Obtain_instance_from_database_view: order

Proposition: 'Next Item in Order'
Obtain_instance_from_database_view: orderItem

rTable 1:
If:                                  | 0| 1| 2| 3| 4| 5|
orderItem.product_type = "Standard"  | Y| Y| Y| N| N| N|
orderItem.product_type = "Premium"   | -| -| -| Y| Y| Y|
orderItem.number_of_units < 100      | Y| N| N| Y| N| N|
orderItem.number_of_units < 500      | -| Y| N| -| Y| N|
Then:
price_per_unit = 10                  | X|  |  |  |  |  |
price_per_unit = 9                   |  | X|  |  |  |  |
price_per_unit = 8                   |  |  | X|  |  |  |
price_per_unit = 15                  |  |  |  | X|  |  |
price_per_unit = 14                  |  |  |  |  | X|  |
price_per_unit = 13                  |  |  |  |  |  | X|
# .......

rTable 2:
If:                                                  | 0| 1|
True = orderItem.origin_country in ["US", "Canada"]  | Y| N|
Then:
handling_fee = 0                                     | X|  |
handling_fee = 50                                    |  | X|
# .......

rTable 3:
If:                                                  | 0| 1|
orderItem.hazardous = "Yes"                          | Y| N|
Then:
surcharge = 1.3                                      | X|  |
surcharge = 1                                        |  | X|
# .......

Initial_instructions:
>>: True = 1             # define Boolean
End_Instructions

Attribute: total_item_price  Type: Real
Equals: (orderItem.number_of_units * price_per_unit + handling_fee) * surcharge

Attribute: print_order_item  Type: Text[2]
Equals: str(orderItem.number_of_units) + " x " + str(price_per_unit)  \
      + " + " + str(handling_fee) + " handling fee "                  \
      + ("+ 30% hazardous " if orderItem.hazardous == "Yes" else "")  \
      + "= $" + str(total_item_price)
```

[2] Rest of the code as in solution 1.

Testrun both solutions

```
Order: 1 - Order in the challenge
200 x 14 + 50 handling fee + 30% hazardous = $3705.0
50 x 10 + 0 handling fee = $500.0
200 x 14 + 0 handling fee = $2800.0
150 x 9 + 50 handling fee + 30% hazardous = $1820.0
Total order price: $8825
-------------------------------------------------------
Order: 2 - 2x Order in the challenge
200 x 14 + 50 handling fee + 30% hazardous = $3705.0
50 x 10 + 0 handling fee = $500.0
200 x 14 + 0 handling fee = $2800.0
150 x 9 + 50 handling fee + 30% hazardous = $1820.0
200 x 14 + 50 handling fee + 30% hazardous = $3705.0
50 x 10 + 0 handling fee = $500.0
200 x 14 + 0 handling fee = $2800.0
150 x 9 + 50 handling fee + 30% hazardous = $1820.0
Total order price: $17650
-------------------------------------------------------
Order: 3 - Order with 1 item
500 x 8 + 50 handling fee = $4050.0
Total order price: $4050
-------------------------------------------------------
Order: 4 - Order with some remaining test cases
99 x 15 + 0 handling fee + 30% hazardous = $1930.5
500 x 13 + 0 handling fee + 30% hazardous = $8450.0
99 x 15 + 50 handling fee = $1535.0
500 x 13 + 50 handling fee = $6550.0
Total order price: $18465
-------------------------------------------------------
All orders processed
Time elapsed: 0:00:07.970265[3]
```

[3] Varies depending on database used.

SQL code to create the databases.

```
CREATE TABLE client_order (order_id    int     PRIMARY KEY,
                           description varchar);

INSERT INTO client_order
(
    description,
    order_id
)
VALUES
('Order in the challenge ', 1),
('2x Order in the challenge', 2),
('Order with 1 item', 3),
('Order with some remaining test cases', 4);


CREATE TABLE order_item (order_id           int     REFERENCES client_order
(order_id),
                         item_id            int,
                         product_type       varchar,
                         number_of_units    int,
                         origin_country     varchar,
                         hazardous_material varchar);

INSERT INTO order_item
(
    hazardous_material,
    origin_country,
    number_of_units,
    product_type,
    item_id,
    order_id
)
VALUES
('Yes', 'Germany', 200, 'Premium', 1, 1),
('No', 'US', 50, 'Standard', 2, 1),
('No', 'Canada', 200, 'Premium', 3, 1),
('Yes', 'China', 150, 'Standard', 4, 1),
('Yes', 'Germany', 200, 'Premium', 1, 2),
('No', 'US', 50, 'Standard', 2, 2),
('No', 'Canada', 200, 'Premium', 3, 2),
('Yes', 'China', 150, 'Standard', 4, 2),
('Yes', 'Germany', 200, 'Premium', 5, 2),
('No', 'US', 50, 'Standard', 6, 2),
('No', 'Canada', 200, 'Premium', 7, 2),
('Yes', 'China', 150, 'Standard', 8, 2),
('No', 'France', 500, 'Standard', 1, 3),
('Yes', 'Canada', 99, 'Premium', 1, 4),
('Yes', 'US', 500, 'Premium', 2, 4),
('No', 'France', 99, 'Premium', 3, 4),
('No', 'Germany', 500, 'Premium', 4, 4);[4]
```

[4] SQL commands created with SQLite; also works on PostgreSQL. No idea why SQLite lists the attributes in reverse order in the INSERT command, but there will be a reason for that.