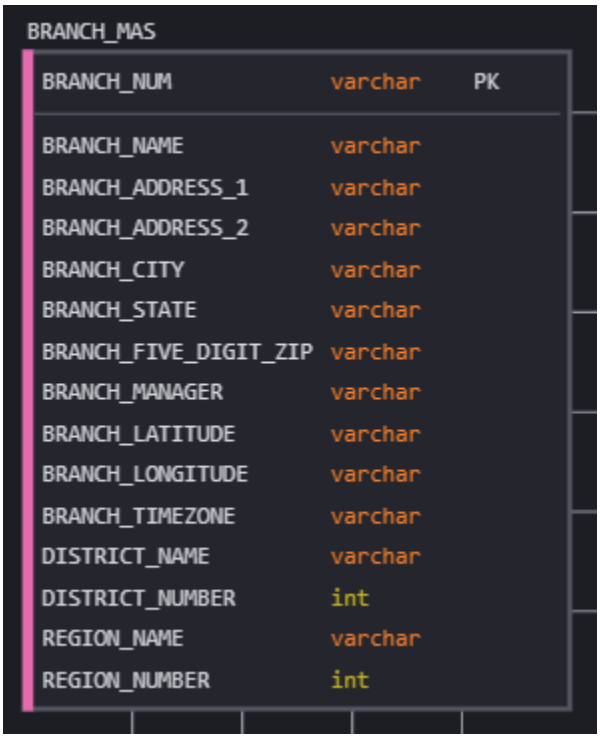
ABC Procurement Optimization Modular Design

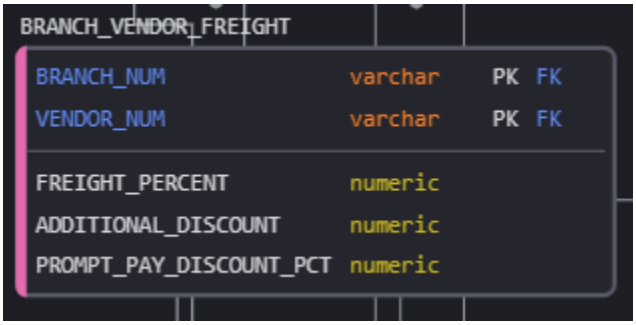
Branches.py

Information on branches, branch to vendor mapping, branches relation to the programs and hence the region/custom location etc. will be prepared in this module.

- get_branch_master()
    1. return a table with all the branches in the ABC operating group, along with district and region
    2. Branch number can be the pk



- get_branch_vendor_freight()
    1. return a table with the freight rate between all branches and vendors supplying them
    2. also add the Additional Discount and Prompt Pay Percent

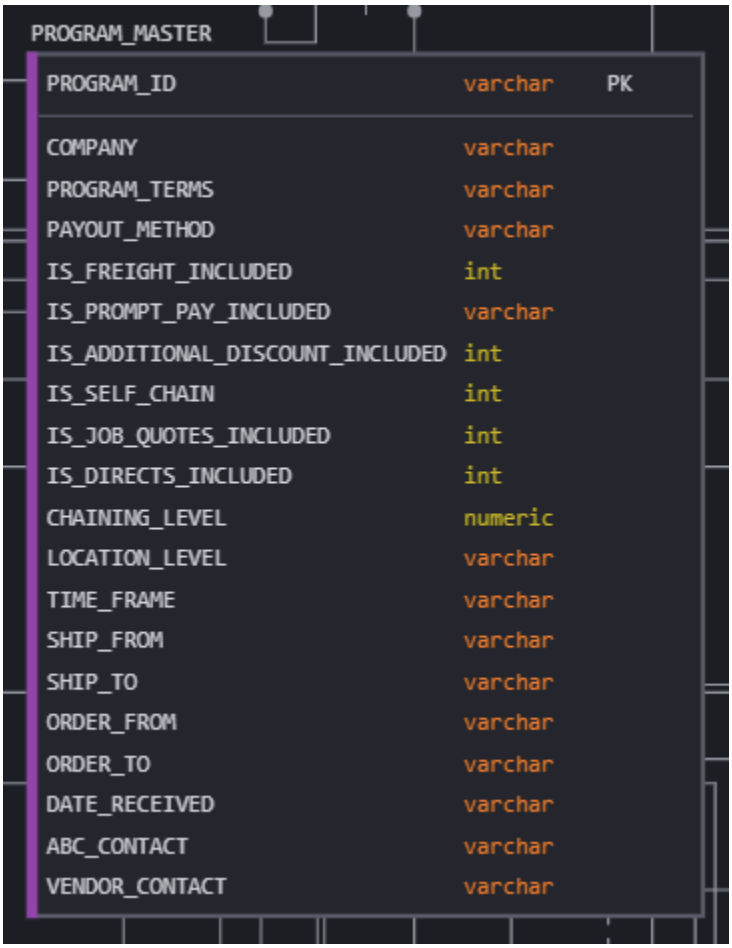


BuildPrograms.py

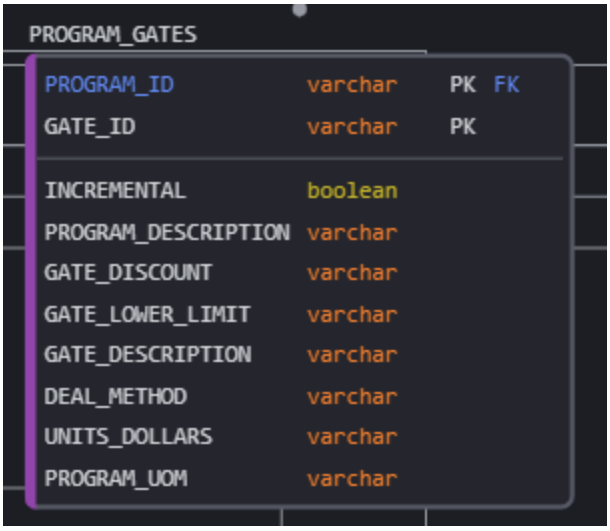
This module will be utilized to prepare the data on all programs.

- build_program_master()

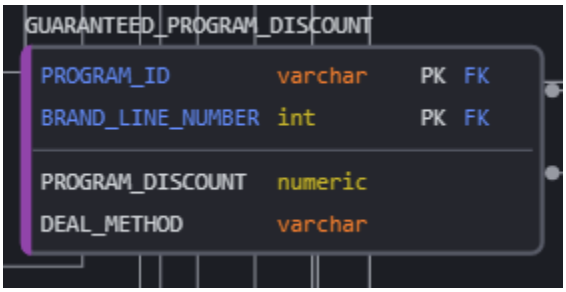
1. prepare a table with all the programs offered in the current year with the details like discount methods, chaining level, etc.
2. Program Terms – Guaranteed (Non-Gated) or Non-Guaranteed (Gated)
3. Deal Method - % Discount or Cost Discount per UOM
4. Combine tables – Table All and Rebate Program Log from data lake
5. Is Freight, Prompt Pay, Additional Discounts Included and Is Self Chain
6. Is Job Quotes and Directs Included



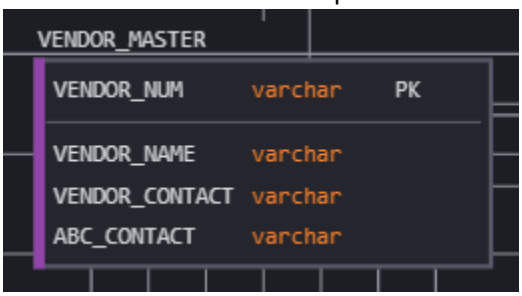
- build_program_gates()
    1. prepare a table for the gated programs with the gate ids created like 0,1,2, etc.
    2. Gate Discount – the % discount (guaranteed discount or deal value) or $ discount for the program-gate depending on the Programs Deal Method (5% or $3/SQ)
    3. Gate lower limit
    4. Deal Method – Discount in Percent or Dollars
    5. Program UOM – the UOM in which the gates are represented
    6. Units Dollars – Whether program gates are in units or dollars



- build_guaranteed_program_discount()
    1. return a table with the program discount (guaranteed discount or deal value) of each program at each applicable brand line level
    2. Deal Method – Discount in Percent or Dollars

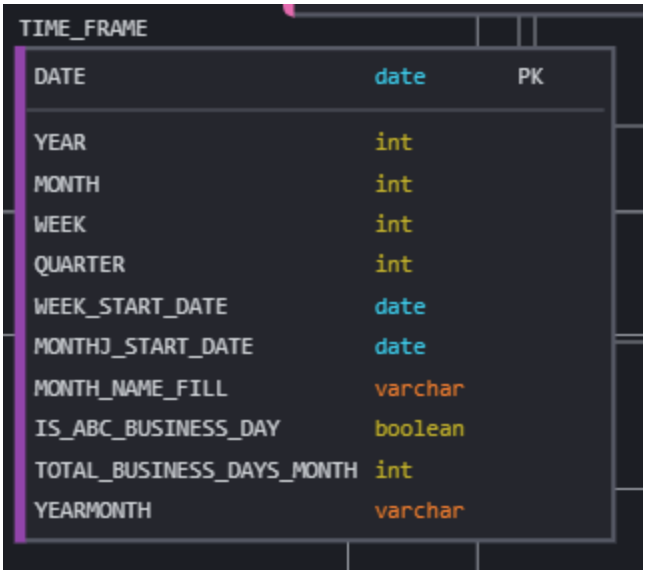


- get_vendor_master()
    1. return a table with all the vendor details like vendor number, name, etc.
    2. Vendor Number can be the pk

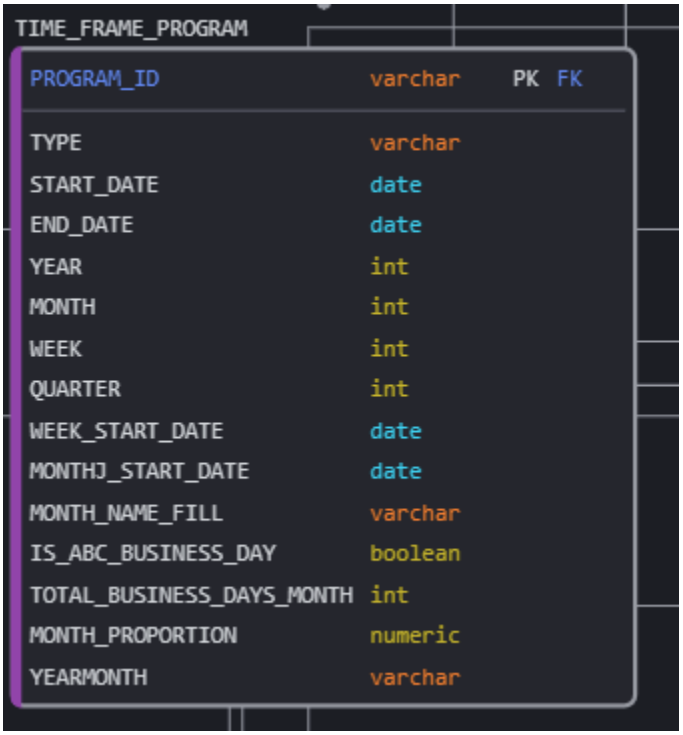


- build_program_vendors()
    1. return a table with all the programs offered by the vendor number

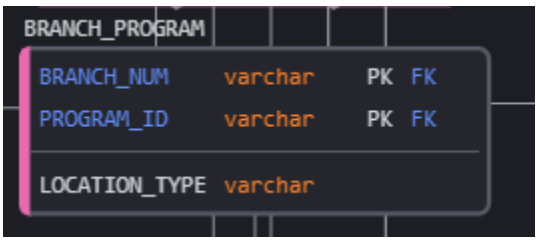
- build_time_frame()
    1. return a table with all the days, months in the calendar year



- build_program_time_frame ()
    1. return a table with all the programs and the months in which they are applicable
    2. Month Proportion – For ST programs calculate the proportion the programs are applicable in a month



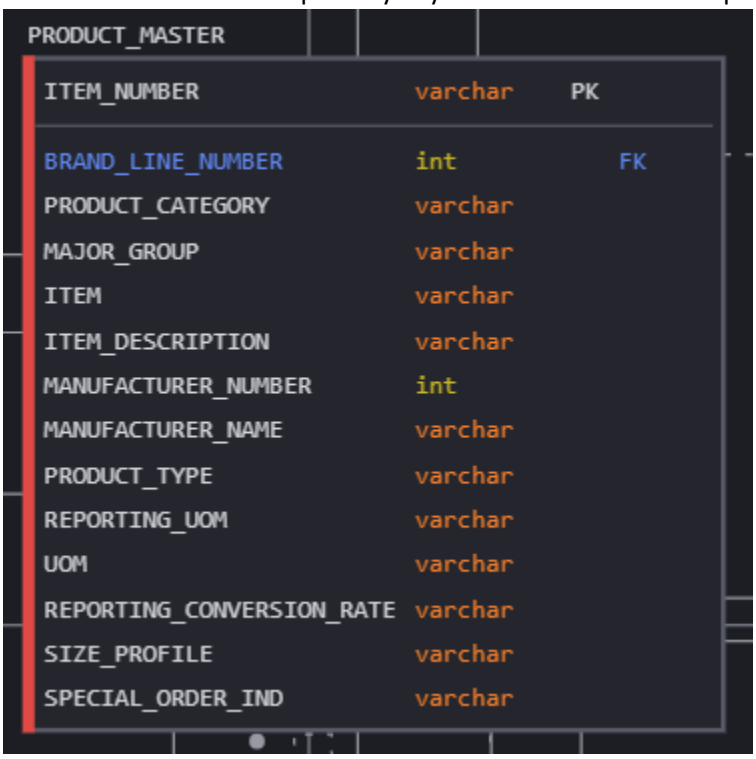
- build_program_branches()
    1. prepare a table with all the branches and programs applicable to them



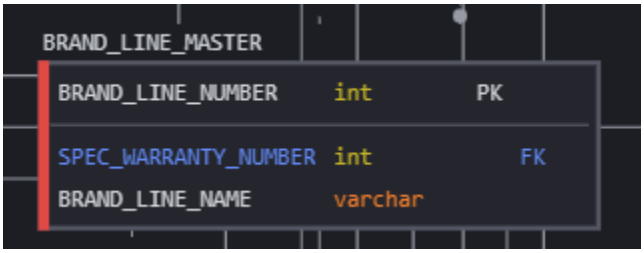
Products.py

The products related information like product groups, hierarchy like brand line, etc. will be formed in this module.

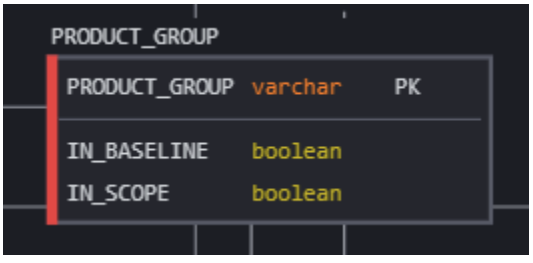
- build_product_master()
    1. return table with each item number along with other item hierarchy details like Spec Warranty, Product Category, etc.
    2. Item Number will be the primary key. Will be used to create product groups



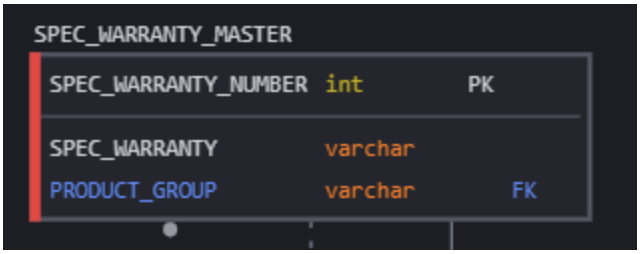
- get_brand_line_master()
    1. return table with each brand line number as primary key



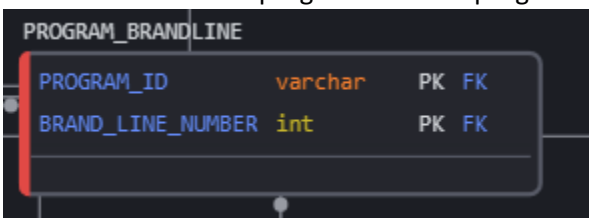
- build_product_group()
    1. prepare the product groups which has the fungible/replaceable products in it



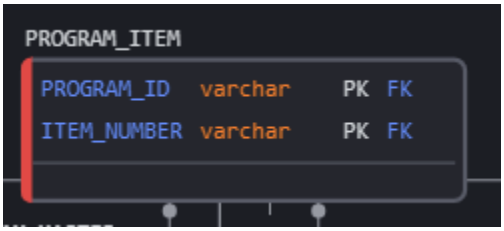
- get_spec_warranty_master()
    1. return table with each spec warranty number as primary key and the related product group as attribute



- get_program_brandline()
    1. return table with the program id of the programs applicable to each brand line



    2. return table with the program id of the programs which are applicable at item number level

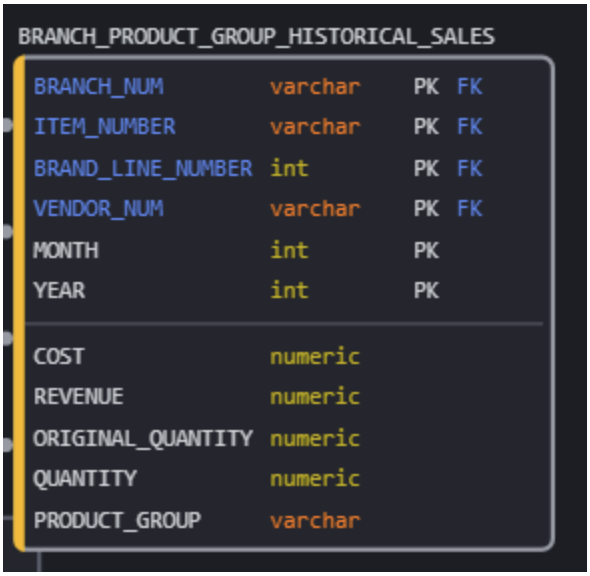


- build_product_group_program()
    1. return the programs applicable to each product group
    2. there will be multiple programs applicable to a product group since the product group contains multiple products offered by different vendors

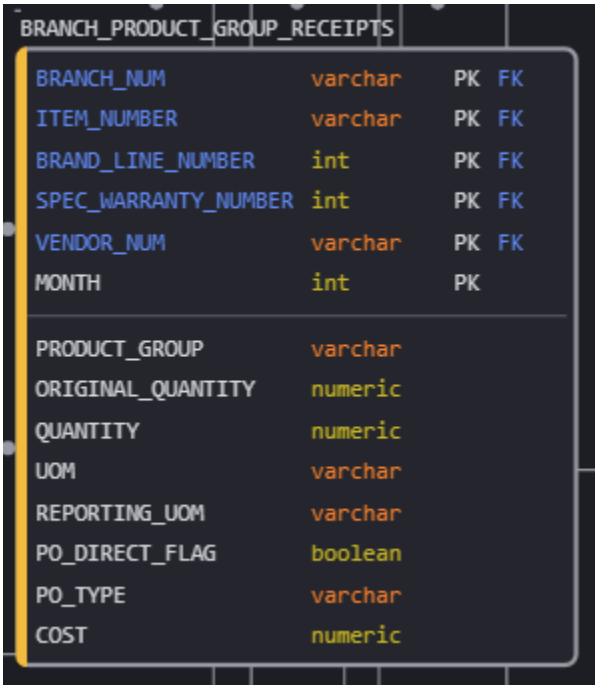
BranchDemandForecast.py

Forecast the monthly quantity of all products to be sold at each branch. Compute job quotes, directs proportions etc.

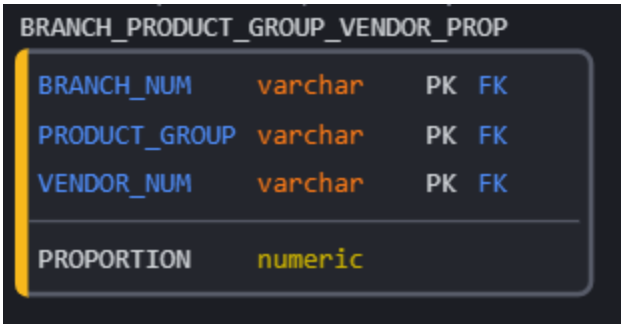
- get_product_group_branch_historical_sales()
    1. return a table with the historical receipts for each product purchased from each applicable vendor at any branch
    2. calculate the quantity = reporting uom quantity = original quantity/reporting conversion rate



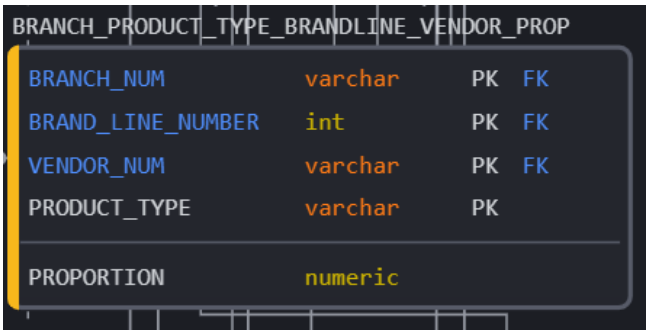
- get_product_group_branch_historical_receipts()
    1. return a table with the historical receipts for each product purchased from each applicable vendor at any branch
    2. calculate the quantity = reporting uom quantity = original quantity/reporting conversion rate
    3. PO Type to get Job Quotes Quantity
    4. PO Direct Flag to get Direct Shipments Quantity



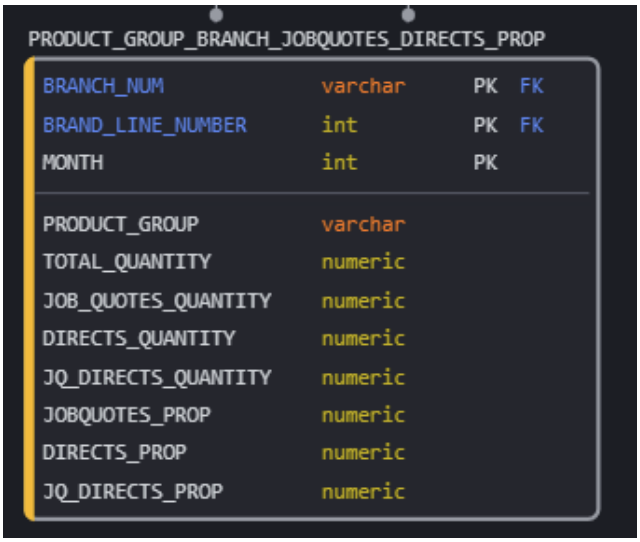
- build_branch_product_group_vendor_prop()
    1. since a product group contains products offered by multiple vendors, we need to calculate the historical proportion of products purchased from a vendor from the total purchases for the product group from all vendors
    2. use table branch_product_group_receipts
    3. this will be used for setting discretionary vs non- discretionary demand



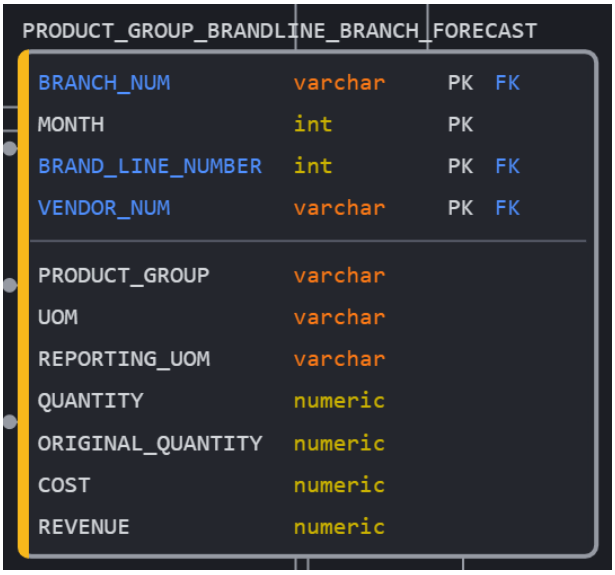
- build_branch_product_type_brandline_vendor_prop()
    1. these proportions let us convert the forecasts at product type level to brand line – vendor level
    2. brand line – vendor level is important since we can have many vendors for a brand line and we need the forecast for each brand line purchased from the specific vendor to apply the vendor's program discounts
    3. uses table branch_product_group_receipts first and then uses the branch_product_group_historical_sales to get any proportions which were not in receipts data



- build_jobquotes_directs_historical_prop()
    - return a table with the historical proportions of total quantity which was job quotes and directs for a product group f at branch b and time t
    - jobquotes_prop
        - for a product group f at branch b and time t sum the total quantity purchased with PO_TYPE='J' and divide by the total quantity to get jobquotes_prop
    - directs_prop
        - for a product group f at branch b and time t sum the total quantity purchased with WAREHOUSE = 'D' to get the directs quantity and divide by the total quantity to get directs_prop
    - jq_directs_prop
        - for a product group f at branch b and time t sum the total quantity purchased with PO_TYPE='J' and WAREHOUSE = 'D' and divide by the total quantity to get jq_directs_prop

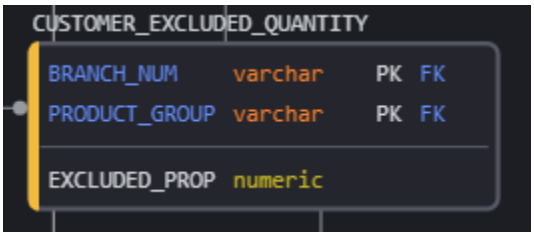


- build_product_group_brandline_branch_forecast()
  - read Enterprise forecast data to get the forecast data (Quantity and Revenue) for the project
    - Forecast for all products except Special Orders is at Item level. The item_number column in the data is basically the item dimension from dim_product and not the item_number iteself
    - Special Orders products Forecast is at Product Type level. Also, Special Orders products forecast only has Revenue and Quantity is null. This is because Special Orders can not be forecasted in quantity since the quantity and unit cost do have any relation for the Special Orders products.
  - Read product master table to convert the Non Special Orders forecast from item level to product type level
  - Use table branch_product_type_brandline_vendor_prop to convert the forecast at product type to brandline-vendor level
  - Try average proportions at regional then overall company level to fill any Null proportions
  - Give equal proportions to brandline-vendor for any product type where we did not have any proportions at all
  - Now use the branch_product_group_historical_sales table to convert the revenue to costs using proportions. Calculate the cost/revenue proportion for each brand line and use it in the forecast table to get costs from revenue
  - This table will contain finally the revenue, cost, original quantity, and quantity (original quantity/reporting conversion rate) at branch- brandline-vendor-month level.

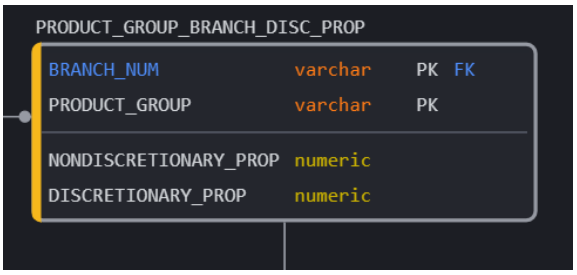


- get_customer_excluded_prop()

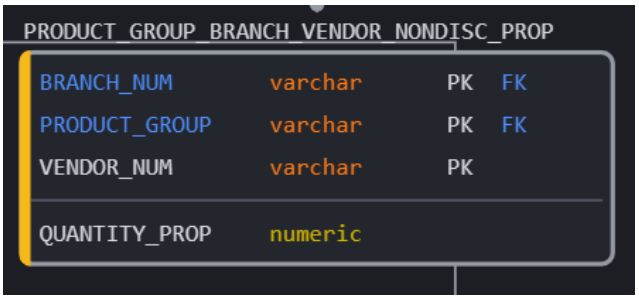
1. return a table with the historical proportion of a product which was sold to excluded customers at a branch



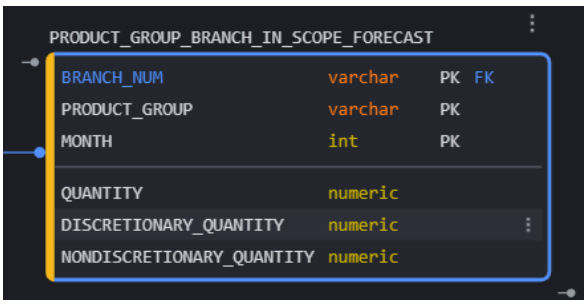
- **get_product_group_branch_disc_prop()**
  - return a table with discretionary_prop and nondiscretionary_prop for each product group at each branch



- **get_product_group_branch_vendor_nondisc_prop()**
  - return a table with nondiscretionary_prop for each product group at each branch and for each vendor
  - this proportion is the proportion of total demand (and not just nondiscretionary_demand), so the total quantity_prop at product_group-branch-vendor level will not be equal to 1

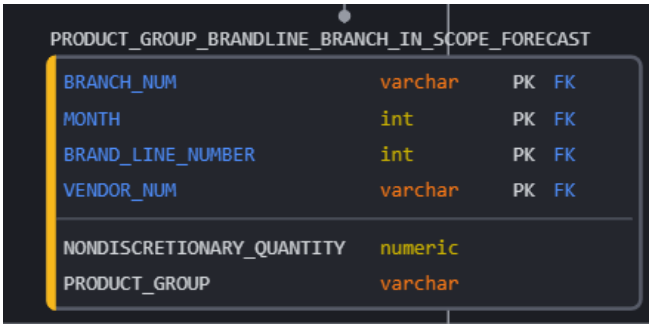


- **get_product_group_branch_in_scope_forecast()**
  - use the customer_excluded_quantity table to get the excluded_prop for each product at each branch
  - return a table with the in-scope forecast of each product at each branch after applying the excluded_prop on product_group_brandline_branch_forecast table
    = total_forecast * (1-excluded_prop)
  - using discretionary and non-disc proportions from table product_group_branch_disc_prop, add the discretionary_quantity and nondiscretionary_quantity

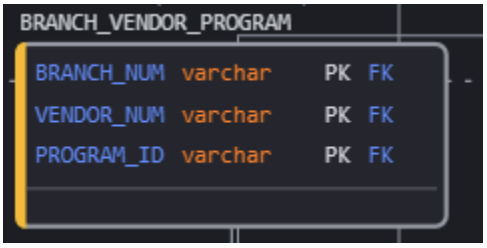


- **get_product_group_brandline_branch_in_scope_forecast()**

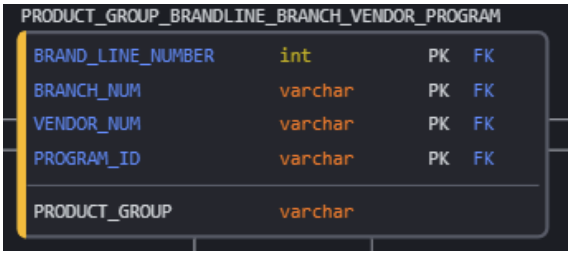
- using the table product_group_brandline_branch_forecast **and the table** product_group_branch_vendor_nondisc_prop return the nondiscretionary_quantity of brand lines – vendors – branch -month



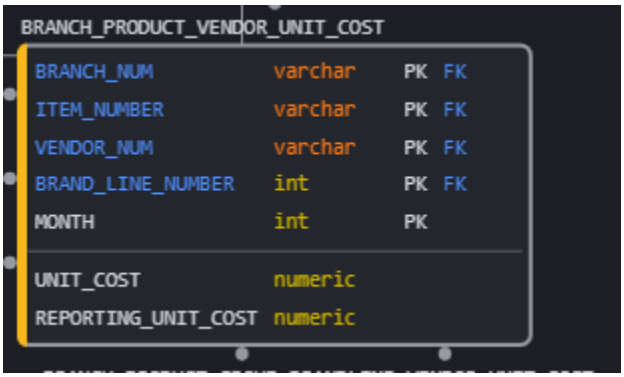
- **build_branch_vendor_program()**
  1. merge tables branch_program and program_vendor on program_id to create table branch_vendor_program
  2. all the vendors and the programs applicable to branches will be in this table



- **build_product_group_brandline_branch_vendor_program()**
  1. merge tables branch_vendor_program and product_group_program on program_id to get all the product groups and brand lines mapping to branch and the vendor programs

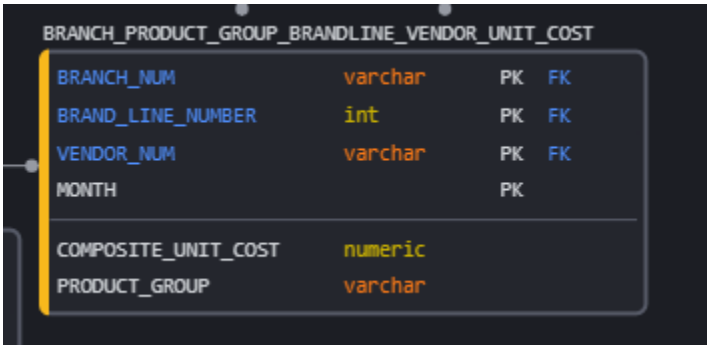


- **build_branch_product_vendor_unit_cost()**
  - return a table with the replacement unit cost at uom and reporting uom level for each branch item vendor level



- **build_branch_product_group_brandline_vendor_unit_cost()**
  1. merge tables branch_product_vendor_unit_cost and product_group on item number and brand line number to add the product groups
  2. compute composite unit cost for each brand line within a product group at each applicable branch from the supplying vendors

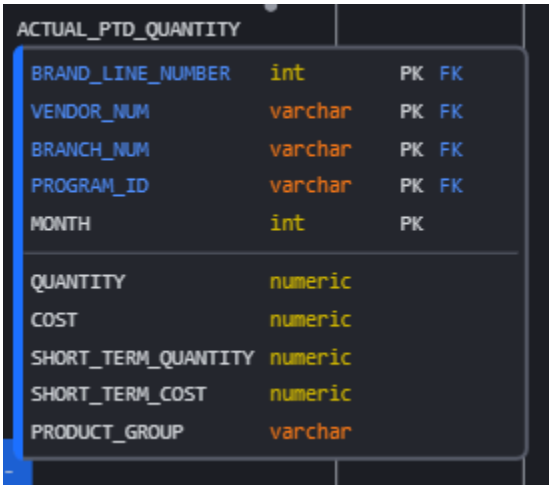
- composite_unit_cost for two products in a Brandline in a product group = (product1 sales weightage * product1 unit cost + product2 sales weightage * product2 unit cost)
- if all items in a brand line do not have any sales/receipts then give equal weightage,
- o.w. if some items do not have receipts then give 0 weightage to those items.



IntegrateData.py

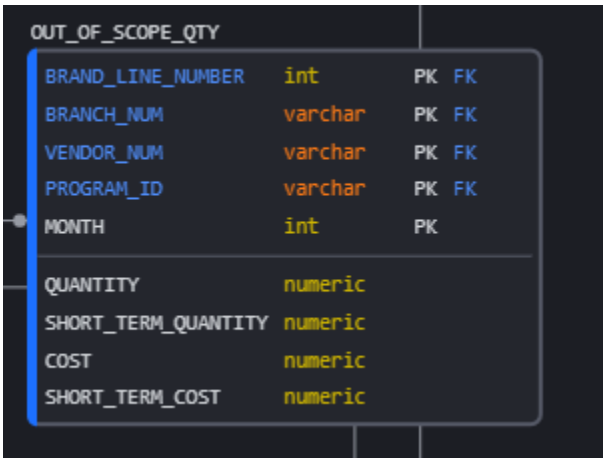
Module will utilize the inputs prepared and will integrate the inputs to prepare inputs for procurement optimization model.

- **compute_actuals_ptd_qty()**
  1. use the branch_product_group_receipts table and return a table with the actual quantity purchased to date of product group f and brand line i from vendor v at branch b for program p at month m
  2. Use Month Proportion from program time frame to calculate the quantity applicable to a program in a month and then add the quantity to get total quantity for the program

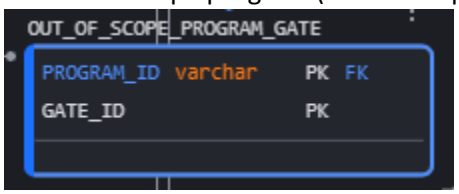


- **compute_out_of_scope_program_qty()**
  1. use the branch_product_group_receipts table and return a table with the actual quantity purchased to date of out of scope product group f and brand line i from vendor v at branch b for in scope programs p
  2. these out of scope products are non fungible products
  3. use Month Proportion from program time frame to calculate the quantity applicable to a program in a month and then add the quantity to get total quantity for the program
  4. for each distinct program applicable to all in-scope products at a branch and any month sum the forecast quantity of all the products in this program which are out-of-scope and map this program to its vendor
     (for baseline - for any month this quantity will simply be the quantity purchased from a vendor of all the products which come under a program and are not-in-scope. All the annual programs with same set of

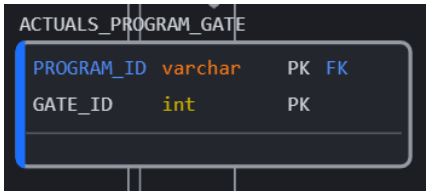
products applicable at a branch will have the same quantity except for the short-term programs which will only show quantity for months in which they are active)



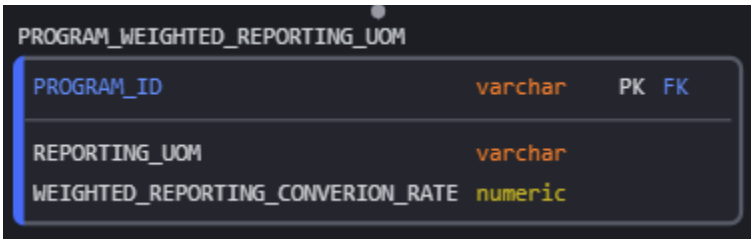
- build_os_actuals_program_gate()
    1. use branch_product_group_receipts, actuals_ptd_qty, out_of_scope_qty table and return a table with the gate hit for each
        - out of scope program (additional programs applicable to out of scope products)



        - actuals ptd programs (st programs applicable in past months)

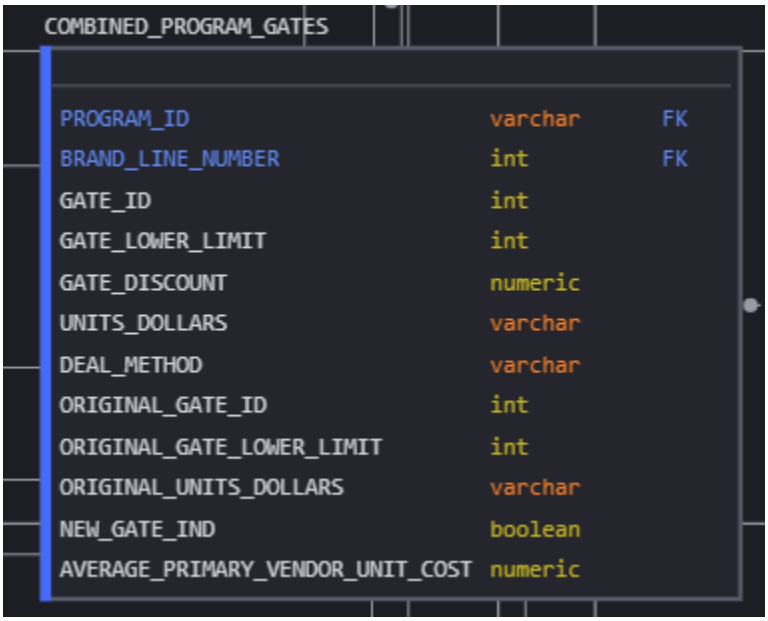


- build_program_weighted_reporting_uom()
    1. compute weighted_reporting_conversion_rate for each program where the gate limits are not specified in the reporting uom of applicable products
        - weighted_reporting_conversion_rate for x products in a program = (product1 receipts weightage * product1 reporting conversion rate + product2 receipts weightage * product2 reporting conversion rate + …)



- build_combined_program_gates()
    1. This table contains the new gates created for the programs which chain among themselves as well as the out of scope programs (with discounts mapped to the gate hit).

2. The programs which chain together are grouped by primary
3. vendor and then gates are created by getting gate limits of all programs and sorting themin ascending order
4. AVERAGE_PRIMARY_VENDOR_UNIT_COST - The average composite unit cost of each brandline (of chained programs) from the primary vendor. Programs with dollar based limits are converted to units using this cost.



- **compute_unit_cost_discount()**
  1. calculate the program discount in $ that vendor provides for a unit of product and return a table for each decision variable –> (product group – brand line - branch_num - vendor_num - program_Id - gate_id – month)
  2. Algorithm for $unit\_cost\_discount_{fivbpgt}$ calculations

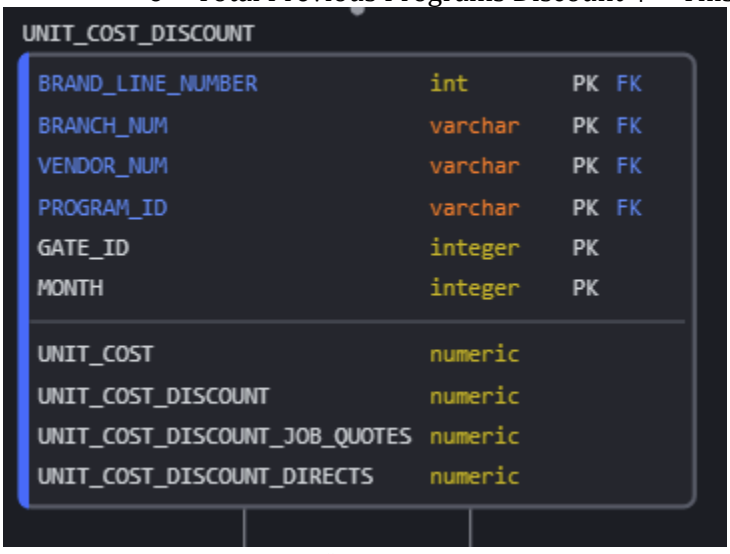
  ➢ For the product group f , brand line b, from vendor v at branch b and time t
  - Get list of all distinct programs applicable from unit_cost_discounts table and call it $program\_list_{fivbt}$
  - Get chaining level of all programs from $program\_list_{fivbt}$
  - Get dense rank on chaining levels to match programs at same chain level to one rank
    - Rank 1 - Lowest Chaining Level Number Programs (e.g. Chain Level = 0,0,0)
    - Rank 2 - Second Lowest Chaining Level Programs (Chain Level = 2,2)
    - Rank N - Highest Chaining Level Programs (Chain Level = 5,5,5)
    - Calculate Program Discount or Unit Cost Discount on each dense rank
    - Programs at any one rank will get the discounts independently of other program at same level meaning they do not chain from itself and hence they won't impact each other's discount.
    - Programs do not chain if -
      - cost based programs
      - chaining Level is 'e' however the program gets discount (calculate discount separately)
      - or JobQuotes Not included and PO Type 'J' * (exclude these)
      - or DirectShip not included and PO Shipped to Direct Warehouse* (exclude these)

  * Program which do not provide rebates for Job quote POs and Direct shipments will be handled by Job Quote Proportion and Direct Proportions used in the optimization function. For non-job quote forecast and non-directs forecast, all the applicable programs will provide the discount,

whereas for job quote and directs forecast, the programs with JobQuotes not included and DirectShip not included respectively will not chain and not provide any discount as well. We need to think about the cases where a job quote can be direct as well.

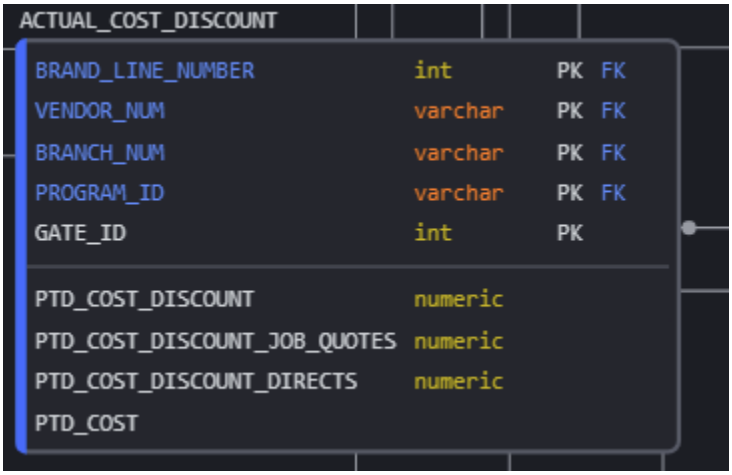
- For Dense Rank n = 1 to N:
  - If Dense Rank n = 1:
    - Total Previous Programs Discount = 0
  - If Chaining Level == 'e': # No chaining for e chains, hence no previous programs deduction and accrual:
    - Previous Programs Discount Deductions = 0
  - Else:
    - Previous Programs Discount Deductions = Total Previous Programs Discount
  - Set cost based programs chaining discounts deduction to 0 because they don't chain
  - Cost =$unit\_cost_{fivb}$ (correct UOM especially for cost discount programs or will need merchandising conversion rate)
  - For programs p in Dense Rank = n:
    - $freight\_cost_{fivbpt}$ = Freight Cost = Cost * Is Freight Included * Freight Percent
    - $prompt\_pay\_discount_{fivbpt}$ = Prompt Pay Discount = (Cost - Freight Cost) * Is Prompt Pay Included * Prompt Pay Percent
    - $additional\_discount_{fvbpt}$ = Additional Discount = (Cost - Freight Cost - Prompt Pay Discount) * Additional Discount Percent
    - $self\_chain\_discount_{fivbpt}$ = Self Chain Discount = (Cost - Freight Cost - Prompt Pay Discount - Additional Discount - Previous Programs Discount Deductions) * Self Chain Percent * Guaranteed Discount Percent

    For Quantity Based programs: (non incremental and non-cost based)
    - If $program\_group_{fivbt}[program == p \ and \ program\_discount\_type == percent]$:
      - $program\_discount_{fivbpgt}$ = Program Discount[gate = g] = (Cost - Freight Cost - Prompt Pay Discount - Additional Discount - Previous Programs Discount Deductions - Self Chain Discount) * Guaranteed Discount Percent
    - Else $program\_group_{fivbt}[program == p \ and \ program\_discount\_type == cost]$:
      - $program\_discount_{fivbpgt}$ = Guaranteed Cost Discount - Freight Cost - Prompt Pay Discount - Additional Discount
    - $unit\_cost\_discount_{fivbpgt} = program\_discount_{fivbpgt}$
    - This level programs discount += Program Discount
  - If Chaining Level != 'e':
    - Total Previous Programs Discount += This level programs discount

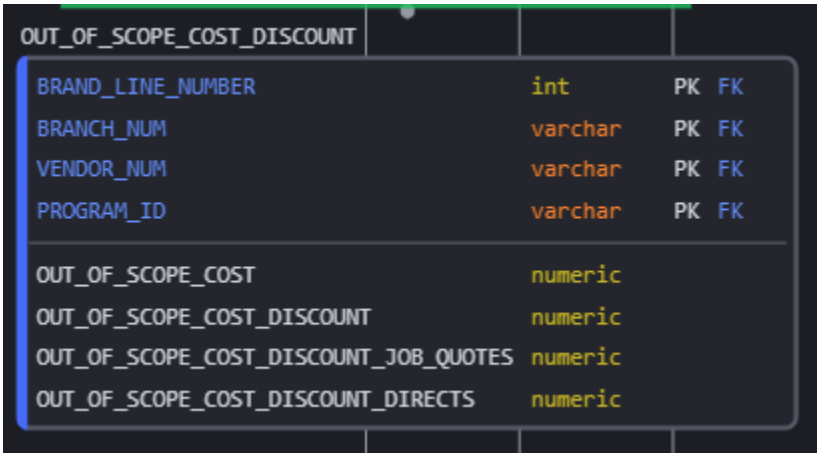


- compute_actuals_cost_discount()

1. follow the unit_cost_discount calculations above except only for the programs which are currently active (for short term) or annual programs. This is because the actuals or part of actuals will only qualify for programs which are applicable annually or currently and, whatever discount they would have gotten on earlier months for short-term programs they have already received it. Do not apply to any short-term programs applicable in future
2. calculate the program discount in $ that vendor provides for total receipts and return a table for each decision variable –> (product group – brand line - branch_num - vendor_num - program_Id - gate_id)



- **compute_out_of_scope_products_cost_discount()**
    1. follow the unit_cost_discount calculations above for the out of scope or non-fungible products
    2. calculate the program discount in $ that vendor provides for total receipts and return a table for each decision variable –> (product group – brand line - branch_num - vendor_num - program_Id - gate_id)



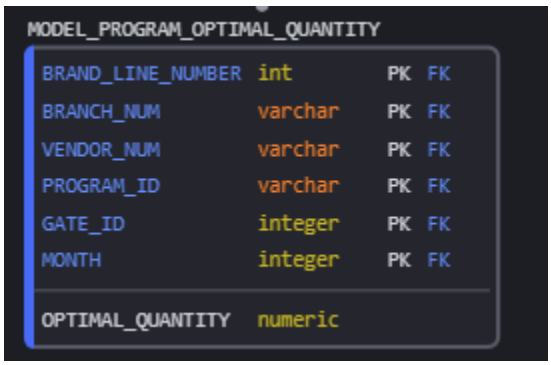
OptimizeABCProcurement.py

Module prepares the procurement optimization model and solves it and then returns the raw outputs.

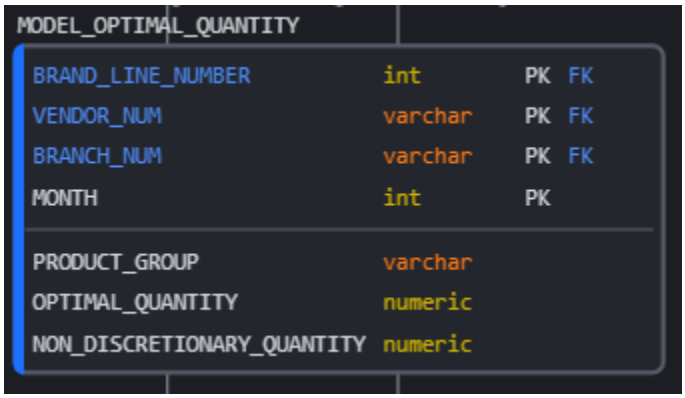
- **build_decision_var_details()**
    3. use table unit_cost_discount to get each decision variable –> (product group - brand line -branch_num - vendor_num - program_Id - gate_id – month)
- **define_and_solve()**
    - _add_decision_variables
    - _add_objective
    - _add_supply_demand_constraint
    - _add_equal_quantity_constraint
    - _add_one_program_gate_constraint
    - _add_BigM_constraint
    - _add_program_gates_lower_limit_constraint

- _add_program_gates_upper_limit_constraint
- _solve
- _collect_outputs

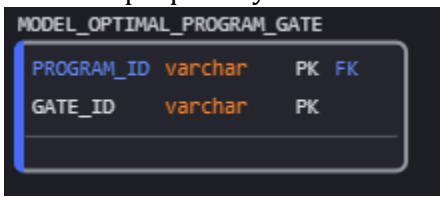
- generate_program_outputs()
  - write the following 3 tables when model is run optimally
    - return the optimal recommended quantity of product group f and brand line i to be purchased from vendor v at branch b under program p at gate g and time t



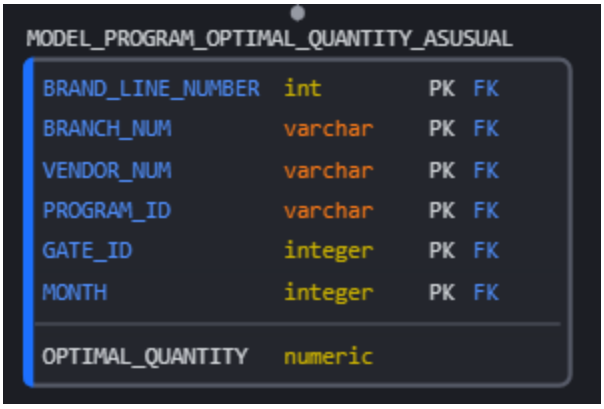
    - return the optimal recommended quantity of product group f and brand line i to be purchased from vendor v at branch b and time t

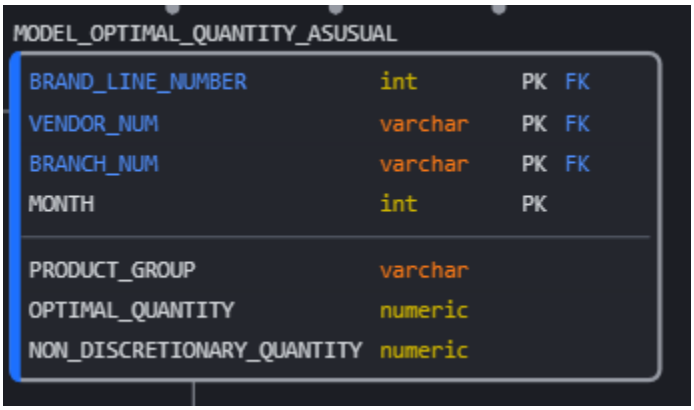


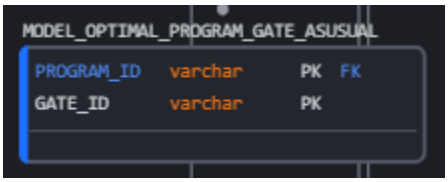
    - return the gate hit for each program based on the recommended quantity and the actuals, and out of scope quantity



  - write the following 3 tables when model is run business-as-usual







Outputs.py

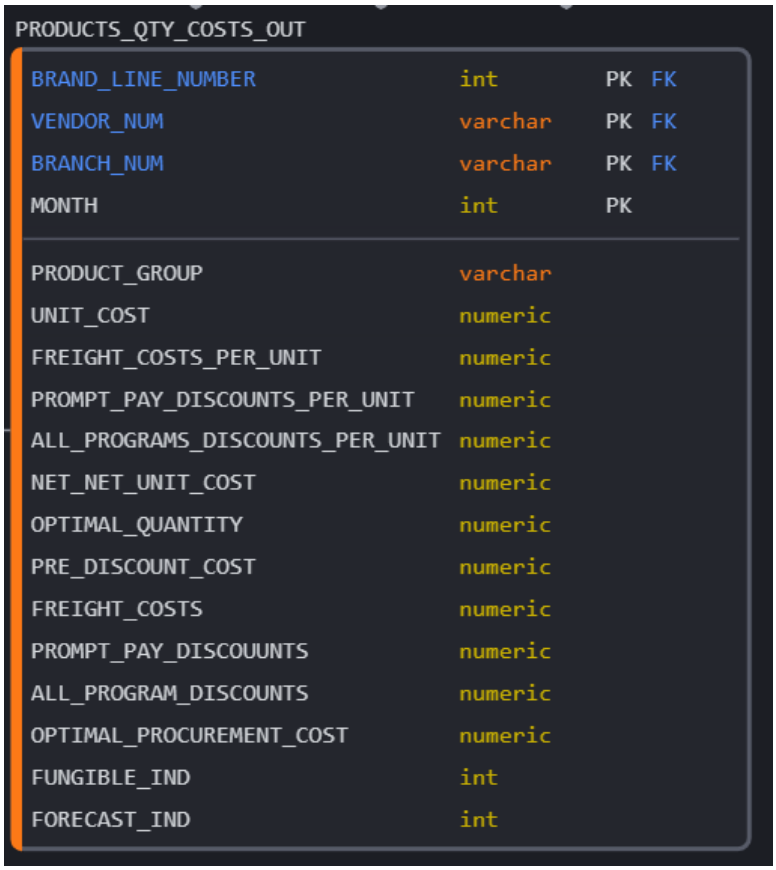
Module will prepare the output tables from the raw outputs of the procurement optimization model.

- write_model_outputs**(run_type = optimal)**
- **Using the model outputs and other tables for optimal run, write the following 3 tables:**
  - ➢ Now, we will combine the actuals, optimal (fungible) and non-fungible products in the output tables.
    - fungible_ind = 1 when the product is fungible and vice-versa.
    - forecast_ind = 1 when the optimal quantity for product is for a future month or actuals ptd and vice-versa.
    - Add customer excluded quantity back to optimal quantity for the fungible feasible products
  - ➢ Created 3 new output tables

1. PRODUCTS_QTY_COSTS_OUT

- This table contains the final optimal quantity, costs, optimal procurement costs (net net cost) at each vendor - brand line - branch - month level.
- The fungible_ind = 1 when the product is fungible and vice-versa.
- The forecast_ind = 1 when the optimal quantity for product is for a future month or actuals ptd and vice-versa. The actuals can be obtained by setting forecast_ind = 0.

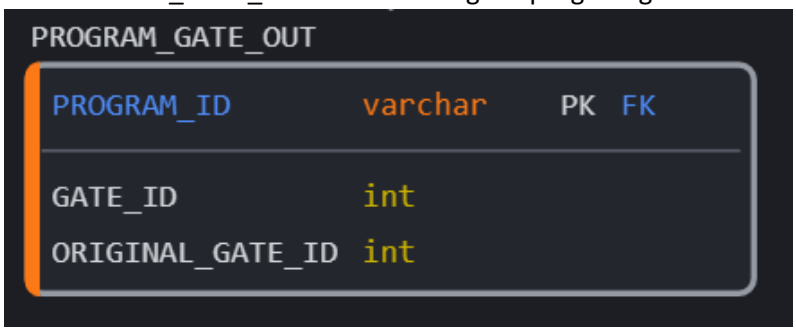
- The columns freight_costs_per_unit and freight_costs which gives us the freight costs per unit of quantity purchased and the total freights costs on the total quantity purchased/ to be purchased
- The columns prompt_pay_discounts_per_unit and prompt_pay_discounts gives us the rompt pay discounts per unit of quantity purchased and the total prompt pay discounts on the total quantity purchased/ to be purchased
- The columns all_programs_discoiunts_per_unit and all_programs_discoun gives us the discounts from all programs applicable to that brand line-vbendor-branch-month per unit of quantity purchased and the total programs discounts on the total quantity purchased/ to be purchased



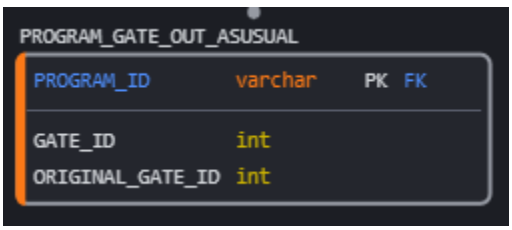
2. PROGRAMS_QTY_COSTS_OUT

- This table basically gives us the **program - gate** hit when we buy **brand line - vendor - branch - month .** It then also gives the quantity and pre_discount_costs applicable to the **brand line - vendor - branch - month - program.** Mostly all the programs applicable to a **brand line - vendor - branch - month** will have same quantity and costs except for the ST programs which has the quantity and costs multiplied by the month_proportion of the program**.**The fungible_ind = 1 when the product is fungible and vice-versa.
- The forecast_ind = 1 when the optimal quantity for product is for a future month or actuals ptd and vice-versa.
- ORIGINAL_GATE_ID - Gate id of original program gates.
- The column – unit_cost_discount is the discount from the program

3. PROGRAM_GATE_OUT
   - This table has the gate hit for all the programs - actuals, fungible, non -fungible. ORIGINAL_GATE_ID - Gate id of original program gates.



- write_model_outputs**(run_type = business-as-usual)**
- **Using the model outputs and other tables for business-as-usual run, write the following 3 tables:**



Config.py

- Application configuration settings and global variables

SharedGlobal.py

- Application database and data lake connector
- Run Control Table

Baseline Run

- In order to accurately measure the business benefit the Procurement Optimization application provides, we need to be able to run the historical actuals (referred to as the Baseline) as compared to the optimal recommendations.
- The measurement should run the historical actuals through the entire solution's model to arrive at the Product Group Actuals, which then stands in place for the forecast in the true, future looking Optimization application. This results in a output value of lost cost savings, in dollars, between the Baseline and the Optimal, for the given time period.
- Process
    - Two config parameters are used to run baseline
        1. basleline – parameter to indicate whether the run is for baseline or not
        2. baseline_year – the year for which the baseline mapping needs to be done
    - All the input data and output data will be stored in new schema – baseline
    - The objective function in the baseline run will run two times.
        1. First run is for optimal scenario which uses the receipts data as forecast at fungible product group level and assigns the demand to different brand lines in that fungible product groups
        2. Second run is for baseline/actuals scenario where the receipts data is used as it is at the brand line level i.e., without shifting demand from one brand line to another in a product group. The receipts data will be pinned using a constraint to fix the values at brand line level so that we can use the production code without any significant changes to
    - The Output tables written both for optimal and baseline/ actuals can be used to provide an apples to apples comparison and a high confidence value of lost cost savings.
        - Optimal tables – products_qty_costs_out
        - Baseline table - products_qty_costs_out_as_usual