

**Bank of Canada Collateral and Pledging Report**

Background Information

The purpose of this report is to provide the Bank of Canada with data on collateral pledging

which is necessary for evaluating linkages between counterparties and activities, as well as

determining the magnitude of transactions across various asset classes.

This return is to be completed at the business day frequency. Business days are defined to

include all weekdays except federal and provincial statutory holidays. The attached template

identifies a selection of rows and columns which are only required to be updated as of month

end. Values are required to be reported in these cells for every business day; however, previous

values can be carried forward un/l month end is reached.

The template for this return has been provided in **“objectives”** section.

[Definition for asset types]

**Level 1 Asset:** Government Bond

**Level 2 Asset:** Non-financial corporate bond/equity, with credit rating between A and AAA.

**Level 3 Asset:** Financial corporate bond/equity, or non-financial corporate bond/equity with credit rating under A.

Objectives

Fill in the report template as requested, based on Bank of Canada's rules.

Table

Description automatically generated

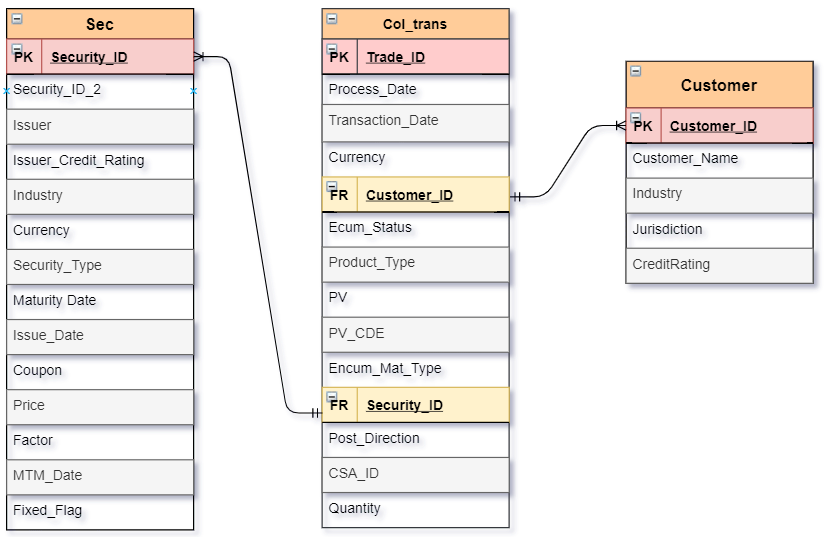
Inputs

Collateral transactions as of Jul 2020. (“col\_trans.csv")

Security instrument details. (“sec.csv")

Counterparty details. ("customer.csv")

ER – Diagram



Analysis of this case and potential challenges

Given the three tables (Col\_trans, Sec, Customer), we will know that the main table is the col\_trans table and expand it by appending two more columns (counterparty type and how many assets they have according to asset class). There are three main steps we need to consider:

* Comparing to the report template, we notice that we need first to distinguish three counterparty types (Domestic, other Domestic, and Foreign) and three-level asset (level 1, level 2, and level 3).
* We need to expand the col\_trans by appending each counterparty’s type and the corresponding asset class according to the securityID.
* We need to sum up each level’s asset and group by the counterparty type, direction and collateral type, since we need to follow the format of the template header.

We may encounter two challenges in this case:

1. Each security has two securityID (security\_id and security\_id\_2), however, there is just one security\_id column in col\_trans, in which we need to make sure that we compare the security\_id in col\_trans to both securityIDs.
2. Comparing to the report template, each counterparty corresponds with two directions (collateral received or pledged), each direction corresponds with two collateral types (initial margin and variation margin). It seems like we may need to perform that the counterparty cartesian join the direction column and collateral type column.

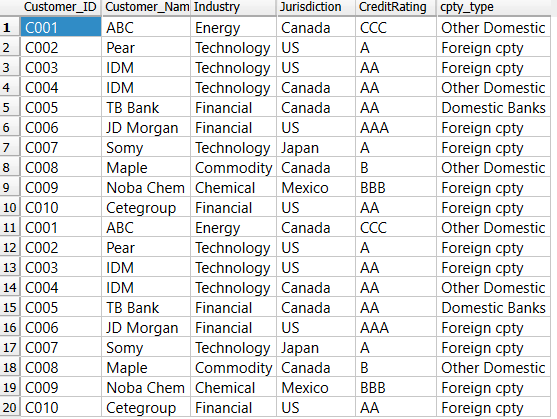
Step 1: Get Counterparty Type

*We need to perform a CASE statement in order to get each customer’s counterparty type*

Graphical user interface, text, application

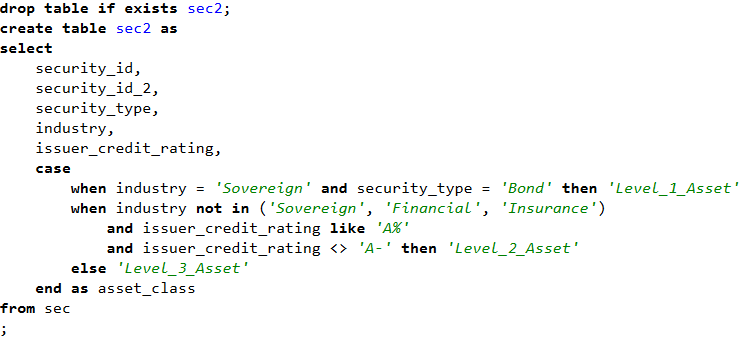
Description automatically generated

After the query, the cust2 table looks like the following:

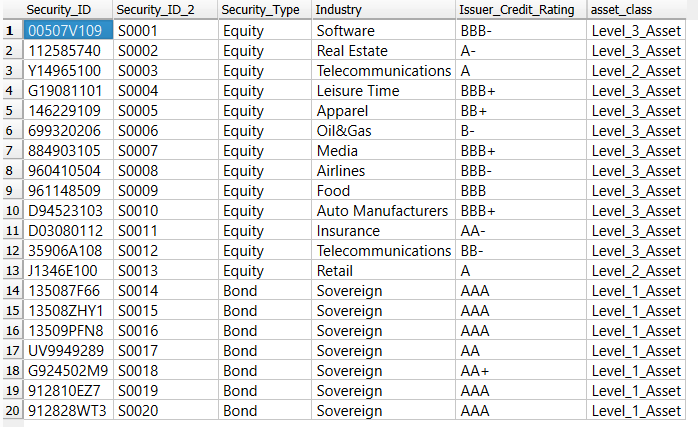


Step 2: Separate Level 1, Level 2, Level 3 Asset depending on their industry, security type and credit rating of issuers

We need to use a CASE statement to separate each asset class, from government bond(level 1 asset) to Financial corporate bond/equity, or non-financial corporate bond/equity with credit rating under A(level 3 asset).

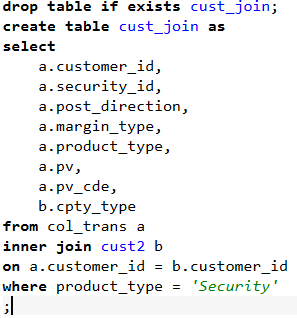


After the query, the sec2 table looks like (creating a new column specifying asset type):

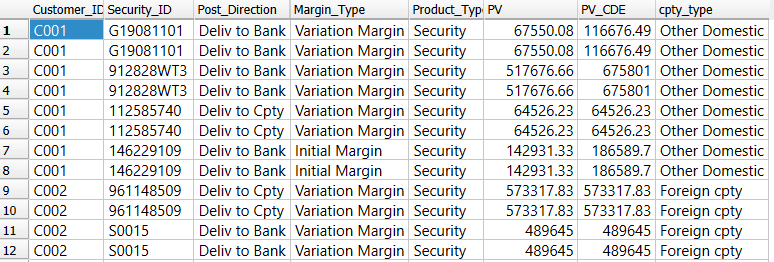


Step 3: Join the counterparty type to col\_trans table

We need to use an **INNER JOIN statement** to avoid not fetching the blank or NULL value in both Col\_trans and Cust2 table, as well as a **where** statement to filter out the **security type** product

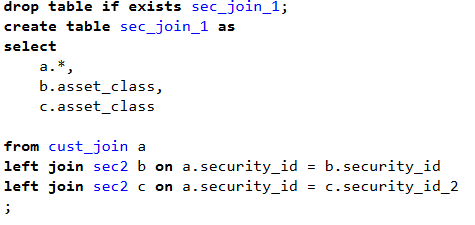


After the query, the cust\_join table looks like:

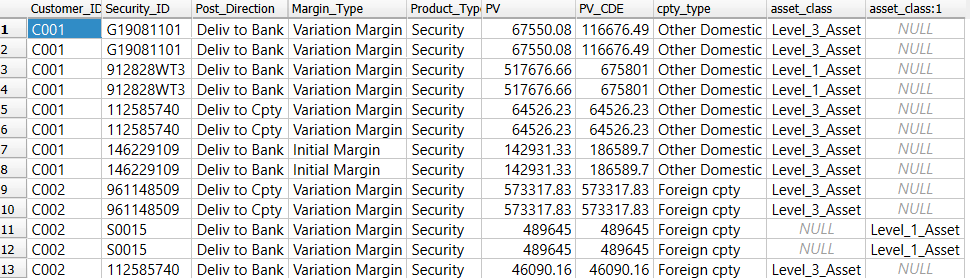


Step 4: Join the asset class (level 1, level 2, level 3) in **sec2** to **cust\_join** table

Method 1(not the best way): Notice that one product can have two security ID (security\_id, security\_id\_2), so we **LEFT JOIN** twice with sec2 in order to match all security IDs, however, there would be two columns columns specifying each product’s **security type**, since we left join twice , and the second column (asset\_class: 1) is produced because of the match in **security\_id\_2**

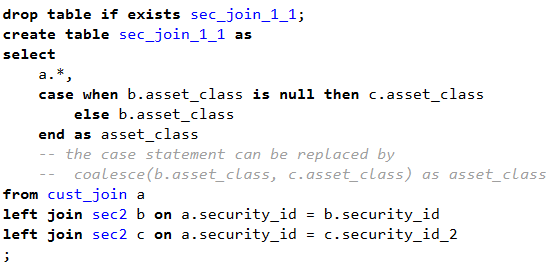


Notice that after the query, there are two columns specifying each product’s **security type**, since we left join twice , and the second column (asset\_class: 1) is produced because of the match in **security\_id\_2**.

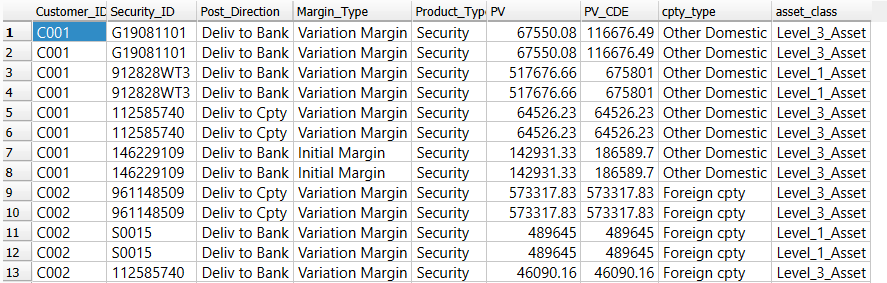


Thus, we want to combine the two columns using a **CASE statement**, that is, when **b.asset\_class** is NULL, we know that we need set the corresponding value to **c.asset\_class.** Similar argument fits the case that **c.asset\_class** is NULL.

The other opportunity to replace the case statement is to use a **coalesce statement** due to the **coalesce functionality.**



After the query, the **sec\_join\_1\_1** table looks more normally:

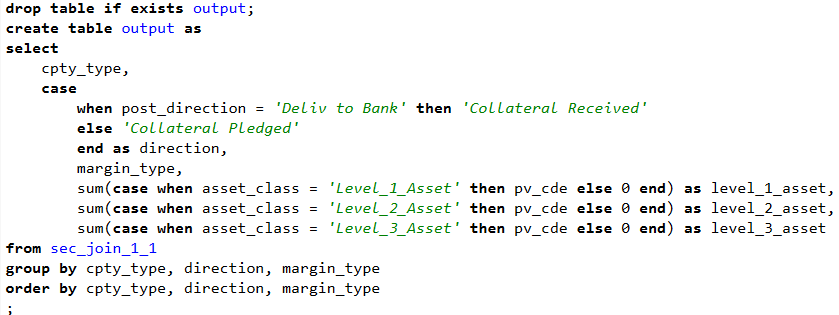


*Later we will introduce two more elegant queries to simplify this step, instead of using two join statement.*

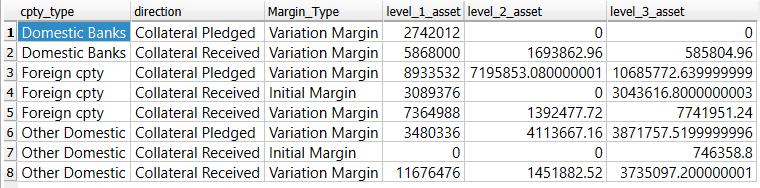
Step 5: Try to approach the report template by summing up each level’s asset and group by Counterparty type, post\_direction, margin\_type

Notice that the report template specifies the first three columns’ name, and each record has its each level’s asset specified. We can argue that we need to group by the **counterparty type(cpty\_type), post\_direction, margin\_type,** and then sum up each its each level’s asset.

Also, in reality, it’s a good manner that we sort the whole table.



After the query, the **output** table looks like:



**Problem 1: We have 12 records in template, why we only have 8 records in the table?**

Notice that in template table, each **counterparty type** corresponds with 2 **direction**, and each direction corresponds with two **collateral type**, so actually we will have 3 \* 2 \* 2 = 12 rows. The reason why we just have 8 columns is because we didn’t include those records with all levels asset equals to 0.

(i.e.:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Domestic Banks | Collateral Pledged | Initial Margin | 0 | 0 | 0 |

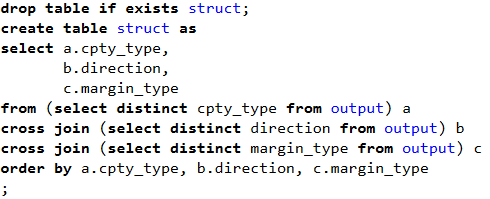
)

**Solution:**

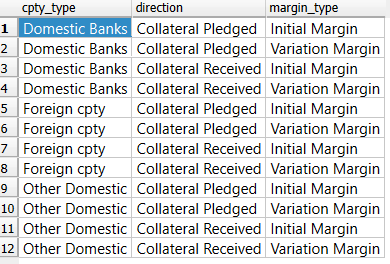
\*\* We can use a **CROSS JOIN statement** to produce the header first. \*\*

Step 6: Initialize the first three columns as the header of our report

The goal can be achieved via **crossing join** the **output** table twice, one for the direction column and the other for the margin type column.

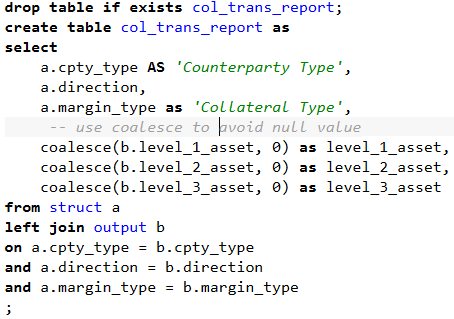


After the query, the header looks like the same as the template report:

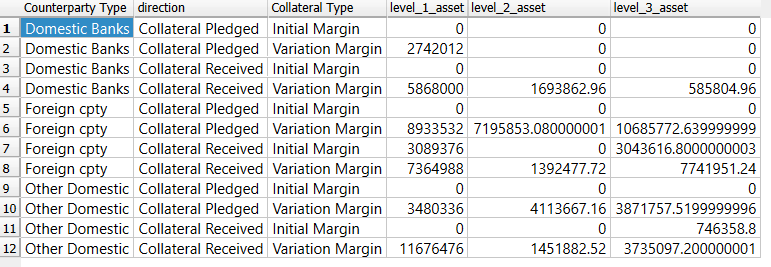


Step 7 (final step): Join the three columns (level 1, level 2, level 3 assets) of output table to our initialized header

So far, we have set the header for the whole table, we just need to join the three columns representing counterparty’s each level asset to our header table (table **struct**), as well as changing the column name to fit the template report.



After the query, the final **collateral transaction report** looks like:



**Problem 2:**

In **step 4**, we use two join statements in order to match all products with corresponding security\_id or security\_id \_2 in **sec2** table, is there a way that we can simplify this process?

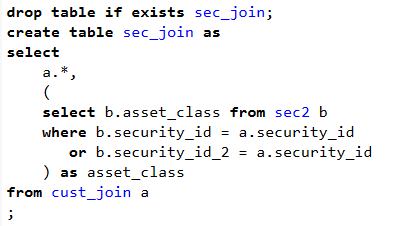
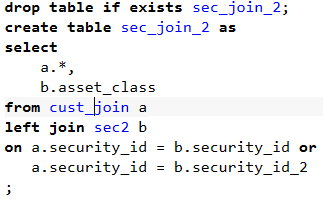
**Solution:**

The problem is about how we can simplify the process that we find all matches in column **security\_id** and **security\_id\_2** in table **sec2**.

We have two better ideas:

* Left join once and use a **WHERE** and **OR** clause (1)
* Use a **SUBQUERY** to fetch corresponding asset\_class (2)

To illustrate (1): To illustrate (2):



The table resulting from both methods are exactly the same:

