Reward Points Data Move

Contents

[Store Data import 2](#_Toc367457499)

[SSIS Project 2](#_Toc367457500)

[Order to run separate packages 2](#_Toc367457501)

[Exchange Rate package Notes 3](#_Toc367457502)

[Control Flow 3](#_Toc367457503)

[Data Flow 3](#_Toc367457504)

[Cancelled Orders Package Notes 3](#_Toc367457505)

[Control Flow 3](#_Toc367457506)

[Data Flow 3](#_Toc367457507)

[Case statement explanation 4](#_Toc367457508)

[Store Item Package Notes 4](#_Toc367457509)

[Control Flow 5](#_Toc367457510)

[Data Flow 5](#_Toc367457511)

[Store Order Package Notes 6](#_Toc367457512)

[Control Flow 6](#_Toc367457513)

[Data Flow 6](#_Toc367457514)

[Case statement explanation 8](#_Toc367457515)

[Data conversions and scripts 8](#_Toc367457516)

[Store Order Item Notes 8](#_Toc367457517)

[Control Flow 8](#_Toc367457518)

[Data Flow 10](#_Toc367457519)

[Derived Column 10](#_Toc367457520)

[Manual Data Clean Up 10](#_Toc367457521)

[Transaction Table corrections. 10](#_Toc367457522)

[Lost transactions rows that have now been put back 12](#_Toc367457523)

[Changes to the old store to make it better -- New Store Process 12](#_Toc367457524)

[Rewards Points Data Import Part One – One time runs 13](#_Toc367457525)

[SSIS Project 13](#_Toc367457526)

[Do it all package notes 13](#_Toc367457527)

[User Eligibility section 13](#_Toc367457528)

[Project reward section 13](#_Toc367457529)

[Control Flow 13](#_Toc367457530)

[Data Flow 14](#_Toc367457531)

[Data Conversion and Script 14](#_Toc367457532)

[Monthly Budget Section 14](#_Toc367457533)

[Control Flow 14](#_Toc367457534)

[Data Flow 15](#_Toc367457535)

[Data Script 15](#_Toc367457536)

[Daily Budget Section 16](#_Toc367457537)

[Control Flow 16](#_Toc367457538)

[Data Flow 16](#_Toc367457539)

[Data Script 17](#_Toc367457540)

[Transactions Section 17](#_Toc367457541)

[Control Flow 17](#_Toc367457542)

[Data Flow 18](#_Toc367457543)

[Data Conversion Scripts 18](#_Toc367457544)

[Transactions Breakdown Section 19](#_Toc367457545)

[Control Flow 19](#_Toc367457546)

[Data Flow 20](#_Toc367457547)

# Rewards Points Data Import Part One – One time runs

## Manual Tables

RewardsProgram is a table that must be filled out manually. Here is the script, make sure you get current data:

**insert into [EleriumClean].[dbo].[RewardsProgram]**

**([ID]**

**,[AddonPercentage]**

**,[BonusPoints]**

**,[EcpmBudgetShareModifier]**

**,[EcpmImpressionsModifier]**

**,[EcpmRevenueModifier]**

**,[EstimatedPremiumBudgetDefault]**

**,[EstimatedPremiumBudgetTrendModifier]**

**,[Fairness]**

**,[LibraryPercentage]**

**,[LibraryThreshold]**

**,[PointExchangeRate]**

**,[PopularityThreshold])**

**Values**

**(1**

**,0.88**

**,36044.7207 -- get the latest value of bonus pool amount**

**,1.00**

**,3.00**

**,0.001**

**,10000.00**

**,1.50**

**,0.80**

**,0.11**

**,3**

**,20.00**

**,100)**

## SSIS Project

### Do it all package notes

This package includes all the one time imports and the daily run imports. It starts off with the daily run User Eligibility section (that is also a separate package).

#### User Eligibility section

This is a daily run package and will be gone over in detail in a later section.

#### Project reward section

Primary Key IDs are not preserved from this table.

##### Control Flow

There is one before script, to truncate the table and set the Identity seed to zero:

**truncate table [dbo].[RewardProjectReward];**

**DBCC CHECKIDENT ('[dbo].[RewardProjectReward]', RESEED, 1);**

##### Data Flow

Data is taken from the Postgres database from the rewards\_project\_awards Table with this query:

**SELECT**

**id,**

**batch\_id,**

**project\_id,**

**is\_library,**

**awarded\_points,**

**popularity,**

**library\_popularity,**

**popularity\_score**

**FROM**

**rewards\_project\_awards**

**order by id**

It is done with the order by id so that the relative order stay the same even though this does not keep the same IDs there are no FK to this ID so it is not needed to keep them the same.

###### Data Conversion and Script

The column is\_library is converted from a Boolean to a bit.

#### Monthly Budget Section

The Postgres Primary Keys are preserved for this table! Because there are no breaks in the IDs and with an order by we are able to just let the MS database use its own increment so long as we set the seed to 1.

##### Control Flow

There are two Execute SQL Tasks that run during the control flow a before and an after script.

The before script drops a Foreign Key FK\_RewardDailyBudget\_MonthlyBudget, truncates the table and resets the identity seed to 1.

**ALTER TABLE [dbo].[RewardDailyBudget] DROP CONSTRAINT [FK\_RewardDailyBudget\_MonthlyBudget];**

**TRUNCATE TABLE [dbo].[RewardMonthlyBudget];**

**DBCC CHECKIDENT ('[dbo].[RewardMonthlyBudget];RESEED, 1);**

The after script adds the Foreign Key back.

**ALTER TABLE [dbo].[RewardDailyBudget] WITH CHECK ADD CONSTRAINT [FK\_RewardDailyBudget\_MonthlyBudget] FOREIGN KEY([MonthlyBudgetID])**

**REFERENCES [dbo].[RewardMonthlyBudget] ([ID]);**

##### Data Flow

Data is taken from the Postgres database from the rewards\_months Table with this query:

**SELECT**

**rewards\_months.id,**

**rewards\_months.month,**

**rewards\_months.year,**

**rewards\_months.post\_date,**

**rewards\_months.change\_date,**

**rewards\_months.actual\_budget,**

**rewards\_months.estimated\_budget,**

**rewards\_months.points\_per\_dollar,**

**rewards\_months.ecpm\_budget,**

**rewards\_months.ecpm\_downloads,**

**rewards\_months.ecpm\_impressions\_modifier,**

**rewards\_months.ecpm\_revenue\_modifier,**

**rewards\_months.ecpm\_budget\_share\_modifier**

**FROM**

**public.rewards\_months**

**order by id**

At the time of writing this there are no breaks in the primary key for the months, if the data does develop a missing number in the primary key then the MS table will have to have the Identity increment turned off and the ID will have to be inserted from the incoming data. Then the identity increment turned back on and reseeded.

###### Data Script

The year and month column form the postgres database are converted into a date column in the new MS database Table RewardMonth.

Code:

**DateTime dt = new DateTime(Row.year, Row.month, 1);**

**Row.MonthlyBudgetDate = dt;**

#### Daily Budget Section

This Table will preserver the original Primary Key IDs from the Postgres database. The new Table in the MS database has to have the auto Identity increment turned off to let the ID from the Postgres database get inserted (turning identity insert on does not work in SSIS package because of the separate connections made for the execute task and data flow). Basically the table has to be copied dropped and re-created, luckily when turning it off the table has just been truncated, but when it turned back on it has data.

##### Control Flow

There are two Execute SQL Tasks for this section and before and after script.

The before script drops two Foreign Keys, truncates the table and then turns off the auto increment of the Primary Key.

**ALTER TABLE dbo.RewardTransaction DROP CONSTRAINT FK\_RewardTransaction\_DailyBudgetID**

**ALTER TABLE dbo.RewardProjectReward DROP CONSTRAINT FK\_ProjectReward\_DailyBudgetID**

**truncate table [dbo].[RewardDailyBudget];**

The after script turns the auto increment of the Primary Key back on, adds back the two dropped Foreign Keys and reset the increment seed.

**declare @id int**

**begin**

**select @id =(select max(id) from [dbo].[RewardDailyBudget]);**

**DBCC CHECKIDENT ('[dbo].[RewardDailyBudget]', RESEED, @id);**

**End**

##### Data Flow

Data is taken from the Postgres database from the rewards\_point\_batches Table with this query:

**SELECT**

**id,**

**"date",**

**month\_id,**

**post\_date,**

**change\_date,**

**addon\_percentage,**

**library\_percentage,**

**bonus\_percentage,**

**budget\_for\_batch,**

**points\_for\_batch,**

**fairness,**

**library\_threshold,**

**popularity\_threshold**

**FROM**

**rewards\_point\_batches**

**order by id**

###### Data Script

The column has a conversion issue that is taken care of with the following C# code in the Script component:

**Row.NewPopularityThreshold = String.IsNullOrEmpty(Row.popularitythreshold) ? 0 : Convert.ToInt32(Row.popularitythreshold);**

#### Transactions Section

The Primary Key from the Postgres database via a temporary holding column called SourceID. The reason for this is the same from the Store tables, there is now a one to one relationship schema where there was a one to many.

##### Control Flow

There are two Execute SQL Tasks for this section and before and an after script

The before script drops a Foreign Key, truncates the table the resets the seed number to 1

**ALTER TABLE [dbo].[RewardTransactionBreakdown] DROP CONSTRAINT [FK\_RewardTransactionBreakdown\_RewardTransaction];**

**truncate table [dbo].[RewardTransaction];**

**DBCC CHECKIDENT ('[dbo].[RewardTransaction]', RESEED, 1);**

The after script adds back the Foreign key that was dropped.

**ALTER TABLE [dbo].[RewardTransactionBreakdown] WITH NOCHECK ADD CONSTRAINT [FK\_RewardTransactionBreakdown\_RewardTransaction] FOREIGN KEY([RewardTransactionID])**

**REFERENCES [dbo].[RewardTransaction] ([ID]);**

##### Data Flow

Data is taken from the Postgres database from the rewards\_point\_batches Table with this query:

**SELECT**

**rewards\_log\_entries.id,**

**rewards\_log\_entries.user\_id,**

**rewards\_log\_entries.related\_id,**

**rewards\_log\_entries.related\_type\_id,**

**rewards\_log\_entries.post\_date,**

**rewards\_log\_entries.point\_change,**

**rewards\_log\_entries.type,**

**CASE**

**WHEN rewards\_log\_entries.type = 'g' THEN (Select rewards\_user\_awards.batch\_id FROM public.rewards\_user\_awards WHERE rewards\_log\_entries.related\_id = rewards\_user\_awards.id)**

**ELSE null**

**END AS "dailyBatchID"**

**FROM**

**public.rewards\_log\_entries**

**where type <> 'b' and type <> 'c'**

Notice here the “where clause” that excludes “type” ‘b’ and ‘c’, this will not bring over the log entries for orders and cancellations of orders. Remember there is a schema change that means these will now have a one to one relationship to its corresponding data and not a one to many. Also that the data will be inserted back into the new table once the store order data is imported.

###### Data Conversion Scripts

There are two script component, one to convert the column ‘type’ from a char to a tinyint and the other is to take the related\_id column and put it in the in the new schemas proper column.

C# code:

**var typeID = Row.relatedtypeid;**

**switch (typeID)**

**{**

**case 166:**

**{**

**// store order**

**Row.OrderID = Row.relatedid;**

**break;**

**}**

**case 4:**

**{**

**// user transfer**

**Row.TransferUserID = Row.relatedid;**

**break;**

**}**

**case 230:**

**{**

**// daily reward**

**Row.DailyBudgetID = Row.dailyBatchID;**

**break;**

**}**

**case 226:**

**{**

**// bonus point award not recording the id where this cam from it is always null**

**break;**

**}**

**}**

The Postgres database schema had the related\_id and related\_id\_type columns where the type told you what the id related to via a cryptic code that was not in a table in the database. The new schema has three FK ID columns. These are OrderID for the FK to the Orders ID, TransferUserID for the FK to the Users ID this is the ID of the user that the points where given to or came from (if the points are negative then it indicates who they were sent to and if they are positive it indicates who they came from), and a DailyBudgetID that is for the FK to the DailyBudget(old daily\_batch) ID this is used when it is a daily point generation row. All of these can be null but only one cannot be null (there is a constraint on the Transactions table that makes it so only one of them cannot be null).

#### Transactions Breakdown Section

There cannot be a keeping of the Primary Key from the Postgres database because this new Table is a new schema for four different table from Postgres.

The purpose of this new table is to have the breakdown of the amount of point given to a user from the generation of each daily budget award so they can see how much cam e form each project that contributed to their total award for that day. This use to be a confusing four table deep query as seen by the query to get the data for this table.

##### Control Flow

There are two Execute SQL Tasks for this section a before and after script.

The before script truncates the table and then re sets the increment seed to 1.

**truncate table [dbo].[RewardTransactionBreakdown];**

**DBCC CHECKIDENT ('[dbo].[RewardTransactionBreakdown]', RESEED, 1);**

The after script remaps the original postgres (reward\_log\_entry) Transaction table Primary Keys to their new MS Table Primary Keys.

**update rgb**

**set rgb.RewardTransactionID = rg.id**

**from [dbo].[RewardTransactionBreakdown] rgb**

**inner join [dbo].[RewardTransaction] rg on rgb.RewardTransactionID = rg.sourceid;**

##### Data Flow

Data for this new MS Table is acquired via this postgres SQL query:

**SELECT**

**ras.id,**

**ras.awarded\_points,**

**rpa.project\_id,**

**rle.id as record\_id**

**FROM**

**public.rewards\_award\_segments ras**

**join public.rewards\_user\_awards rua on ras.user\_award\_id = rua.id**

**join public.rewards\_project\_awards rpa on ras.project\_award\_id = rpa.id**

**join public.rewards\_log\_entries rle on rua.id = rle.related\_id and rle.related\_type\_id = 230**

When the column related\_type\_id is equal to 230 this means that this log entry is for the awarding of daily points and that the related\_id points to the Primary Key of the reward\_user\_award Table that is being removed from the schema. The reward\_award\_segment ID is not needed here and can possible be removed.

There are no data conversion or scripts.

# Store Data import

## Manual Tables

### Fix Monthly current row!!!!!!!!

Make sure current month has correct data.

Delete fake months!

Id 5 and below.

StoreCategory must be filled in manually with this data:

**ID Name DisplayOrder**

**1 Paypal 4**

**2 Amazon.com Gift Cards 1**

**3 Amazon.co.uk Gift Cards 2**

**4 Amazon.de Gift Cards 3**

**5 Curse Employees 0**

**6 Other 5**

StoreCurrency must be filled in with this data:

**ID Code Name Symbol**

**1 USD U.S. Dollare $**

**2 EUR Euro €**

**3 GBP Pound Sterling £**

### Store Tables to be filled in manually

StoreCurrency

ID Code Name Symbol

1 USD U.S. Dollar $

2 EUR Euro €

3 GBP Pound Sterling £

StoreCategory

ID Name DisplayOrder

1 Paypal 4

2 Amazon.com Gift Cards 1

3 Amazon.co.uk Gift Cards 2

4 Amazon.de Gift Cards 3

5 Curse Employees 0

6 Other 5

## SSIS Project

### Order to run separate packages

Do the Store after Rewards Points SSIS (else store data gets truncated!).

1. ExchangeRate
2. CancelledOrders
3. Storeitems
4. StoreOrder
5. StoreItemCode
6. Run the manual fixes in the helper scripts SQL file
7. Manually update StoreItem’s AvatarID, StoreCategoryID, RequiresItemCode,DisplayOrder, rerun FKs for AvatarID and StoreCategoryID

### Exchange Rate package Notes

There is no ID column preservation in this package.

#### Control Flow

The Control Flow has a before and after Execute SQL Task. The before truncates the table and sets the seed to 1. The after adds the USD exchange rate to the table, a row for each month that is currently in there. Here is the SQL statement:

**INSERT INTO [dbo].[StoreExchangeRate] (StoreCurrencyID, DateCreated, DateModified, [Date], Rate)**

**SELECT 1, DateCreated, DateModified, [Date],20**

**FROM [dbo].[StoreExchangeRate] Where StoreCurrencyID = 2**

#### Data Flow

Data is taken from the Postgres database direct from table store\_exchangerate Table data access mode. There are two Scripts to convert data. Currency script takes the ‘currency’ column from postgres and changes it form a string to an integer.

1. EUR changes to 2
2. GBP changes to 3
3. Default is 0

Rate Conversion script takes the ‘rate’ column from postgres is a conversion to the new way of representing the rate, **Row.NewRate = 20/Row.rate**.

The MS SQL table that the data is going into is **[Elerium]. [dbo].[StoreExchangeRate].**

### Cancelled Orders Package Notes

There is no ID column in this table.

#### Control Flow

The Control Flow has a before and after Execute SQL Task. The before just truncates the table. The after fixes two entries (at this point they have their postgres OrderID’s they will be changed later).

**update [dbo].[SourceCancelledOrders] set PointChange = 1.0 where OrderID = 1008;**

**update [dbo].[SourceCancelledOrders] set PointChange = 1593.0 where OrderID = 5066 and ItemID = 14;**

These are two discrepancies that have to be manually fixed because of either bad original data or edge case that is easier manually fixed rather than correct in query (time wise faster to fixe edge case than have query deal with it and possible break something else, these are one time move data queries).

#### Data Flow

Data is taken from the Postgres database via a complex query of the rewards\_log\_entries table. This query handles all the data manipulations needed to move to the new schema ( as noted before there are only two rows that need a manual fix after this query is ran and inserted in to the MS SQL database).

**select**

**a.user\_id,**

**a.related\_id as order\_id,**

**b.item\_id,**

**a.post\_date,**

**c.currency,**

**CASE c.currency WHEN 'USD' THEN b.quantity\*c.currency\_cost\*20**

**ELSE round(**

**b.quantity\*c.currency\_cost\*20\* (**

**(a.point\_change/(select sum(currency\_cost \* quantity)**

**from rewards\_log\_entries as a1**

**full outer join public.store\_orderitem as b1 on a1.related\_id = b1.order\_id**

**left join store\_item as c1 on b1.item\_id = c1.id**

**where type = 'c' and b1.item\_id is not null and order\_id = a.related\_id))/20**

**)**

**)**

**END as pointChange**

**from rewards\_log\_entries as a**

**full outer join public.store\_orderitem as b on a.related\_id = b.order\_id**

**left join store\_item as c on b.item\_id = c.id**

**where type = 'c' and b.item\_id is not null**

##### Case statement explanation

Without the case statement the query does not know how to handle the other type of currencies. Also it was not always stored in the database what the exchange rate for those currencies was for those currencies for that cancelled transaction. Therefore it has to be calculated with an inner select statement that accounts for the fact that reward\_log\_entries can have multiple store\_orderitems that have to be changed to multiple reward\_log\_entries because we are going to a one item per order (vary quantity) system.

### Store Item Package Notes

This Table will preserver the original Primary Key IDs from the Postgres database. The new Table in the MS database has to have the auto Identity increment turned off to let the ID from the Postgres database get inserted (turning identity insert on does not work in SSIS package because of the separate connections made for the execute task and data flow). Basically the table has to be copied dropped and re-created, luckily when turning it off the table has just been truncated, but when it turned back on it has data.

#### Control Flow

The Control Flow has a before and after Execute SQL Task.

The before task drops two Foreign Keys from other tables

1. FK\_StoreItemCode\_StoreItem
2. FK\_StoreOrder\_StoreItem

Truncates the table, then does the turns off identity for the primary key.

The after task turns identity for the primary key on and sets the seed to the next number via this script:

**declare @id int**

**begin**

**select @id =(select max(id) from [dbo].[StoreItem]);**

**DBCC CHECKIDENT ('[dbo].[StoreItem]', RESEED, @id);**

**End**

It also adds back the two foreign keys that where dropped.

##### Special note

The Foreign Key FK\_StoreItem\_StoreCategory must be added ‘WITH NOCHECK’ because that data is added manually after the SSIS data import.

#### Data Flow

Data is taken from the Postgres database via a query of the store\_item table. The only reason a query is used here is because there is some test data the does not need to come over (some of it is very rude).

**SELECT**

**store\_item.id,**

**store\_item.name,**

**store\_item.status,**

**store\_item.post\_date,**

**store\_item.change\_date,**

**store\_item.legal\_info,**

**store\_item.require\_email,**

**store\_item.instantly\_deliverable,**

**store\_item.display\_order,**

**store\_item.currency\_cost,**

**store\_item.employee\_only,**

**store\_item.currency**

**FROM**

**public.store\_item**

**where id > 7 and id <> 16**

There are several data conversions and scripts before the data is inserted into the new MS database.

1. IsEmployeeOnly data Conversion and script – bool true false into a bit (0,1)
2. IsAutomaticallyFulfilled
3. RequiresEmailAddress
4. IsAvailable data Conversion and script – change ‘d’ to 0 and ‘n’ to 1
5. Name data conversion truncates the length of the name to 128
6. Display order data conversion to a tinybit
7. Currency data conversion – USD to 1, EUR to 2, GBP to 3
8. Default values for RequiresItemCodes and StoreCategoryID

### Store Order Package Notes

There is no ID column to ID Column preservation in this package, but the Postgres ID is stored in a temp SourceID column and used later to update other tables that have a Foreign Key to the StoreOrder ID.

#### Control Flow

The Control Flow has a before and after Execute SQL Task.

The before task drops two foreign keys

1. FK\_RewardTransaction\_StoreOrder
2. FK\_StoreItemCode\_StoreOrder

Then truncates the table and reset the identity seed to 1.

The after task adds the two foreign keys back that the before task dropped.

#### Data Flow

Data is taken from the Postgres database via a complex query of the store\_order table. This query handles all the data manipulations needed to move to the new schema and fills in missing data.

This query combines Postgres Tables store\_order and store\_orderItem (this is where the one to many part resides).

This is the query:

**select**

**store\_order.id,**

**store\_order.user\_id,**

**store\_order.status,**

**store\_order.post\_date,**

**store\_order.change\_date,**

**store\_orderitem.quantity,**

**store\_orderitem.item\_id,**

**store\_item.currency,**

**store\_item.currency\_cost,**

**CASE WHEN store\_item.currency\_cost > 0**

**and store\_item.currency <> 'USD'**

**and (store\_itemlogentry.usd\_cost = 0.0**

**or store\_itemlogentry.usd\_cost is null)**

**THEN store\_order.total\_cost /**

**(select sum(si.currency\_cost \* soi.quantity)**

**from public.store\_order so**

**left join public.store\_orderitem soi on so.id = soi.order\_id**

**left join store\_item si on soi.item\_id = si.id**

**where so.id = store\_order.id) / 20**

**WHEN store\_item.currency = 'USD' THEN 1.0**

**WHEN store\_item.currency\_cost > 0**

**and store\_itemlogentry.usd\_cost > 0**

**and store\_itemlogentry.usd\_cost is not null**

**THEN store\_itemlogentry.usd\_cost/store\_item.currency\_cost**

**ELSE 0.0**

**END as exchangerate**

**from public.store\_order**

**left join public.store\_orderitem on store\_order.id = store\_orderitem.order\_id**

**left join store\_item on store\_orderitem.item\_id = store\_item.id**

**left join public.store\_itemlogentry on store\_order.id = store\_itemlogentry.order\_id**

**and store\_orderitem.item\_id = store\_itemlogentry.item\_id**

**order by store\_order.id**

There are 8 orders that will have to be manually updated because of either bad original data or edge case calculations. However they cannot be fully fixed here yet. For some of these it is an issue of the exchange rate is a decimal precision of more than four decimal places.

These scripts can be run now.

update dbo.StoreOrder set PricePerItem = 1.0 where SourceID = 1007

update dbo.StoreOrder set PricePerItem = 1.0 where SourceID = 1008

update dbo.StoreOrder set ExchangeRate = 1.593 where SourceID = 5066 and PricePerItem = 50.00

Order 5066 is the only mixed currency order and the query does not handle that, since it the only one no need to make the query account for it this query is a one-time move data use case. The other two just have missing data in postgres.

##### Case statement explanation

The case statement is account for the different currencies to calculate the exchange rate when that data is missing. Have to take the total amount in points the order cost and sum up the currency cost for each item and quantity that is why there are inner select statements.

##### Data conversions and scripts

1. Exchange Rate from string to money
2. Currency ID USD to 1, GBP to 2, EUR to 3
3. Status ‘f’ to 2, ‘p’ to 1, ‘c’ to 3
4. Currency cost if null or string ‘0.00’ to (money)0.00 issue

### Store Order Item Notes

The ID value is not carried over from the Postgres database.

#### Control Flow

The Control Flow has a before and after Execute SQL Task.

In the before script one Foreign Key is dropped, FK\_StoreItemCode\_StoreOrder.

Then table is truncated and the Identity seed is reset to 1.

In the after script there are some very important scripts that handle updating the old Postgres IDs to the new MS Store Order IDs, insert orders and cancelations into the Transaction Table. Also the Foreign Key that was dropped is added back.

Update the StoreItemCode Table with the new MS StoreOrder IDs using the SourceID.

**UPDATE StoreItemCode**

**SET StoreItemCode.StoreOrderID = SO.id**

**FROM [dbo].[StoreItemCode] StoreItemCode**

**INNER JOIN [dbo].[StoreOrder] SO ON StoreItemCode.StoreOrderID = SO.SourceID**

**WHERE StoreItemCode.StoreItemID = SO.StoreItemID**

Add the Foreign key back.

**ALTER TABLE [dbo].[StoreItemCode] WITH CHECK ADD CONSTRAINT [FK\_StoreItemCode\_StoreOrder] FOREIGN KEY([StoreOrderID])**

**REFERENCES [dbo].[StoreOrder] ([ID])**

**GO**

**ALTER TABLE [dbo].[StoreItemCode] CHECK CONSTRAINT [FK\_StoreItemCode\_StoreOrder]**

**GO**

Take all the orders and insert an individual row for each order into the Transactions Table, remember that the new StoreOrders Table is a union of a one to many table relationship. This is why we have to have new order IDs because the Postgres ones are now not unique.

**INSERT INTO [dbo].[RewardTransaction] (UserID, OrderID, DateCreated, PointChange, [Type])**

**SELECT UserID, ID, DateCreated, (Quantity \* PricePerItem \* ExchangeRate \* 20\*(-1)) AS PointChange, 5 FROM [dbo].[StoreOrder]**

Update the SourceCancelledOrders Table (only used to store data until moved to Transactions no new data will be put here) with the New MS StoreOrder IDs using the SourceID.

**UPDATE SCO**

**SET SCO.OrderID = SO.ID**

**FROM [dbo].[SourceCancelledOrders] SCO**

**INNER JOIN [dbo].[StoreOrder] SO ON SCO.OrderID = SO.SourceID**

**WHERE SCO.ItemID = SO.StoreItemID**

Take the data in SourceCancelledOders and insert it into Transactions.

**INSERT INTO [dbo].[RewardTransaction] (UserID, OrderID, TransferUserID, DateCreated, PointChange, [Type], DailyBudgetID)**

**SELECT UserID, OrderID, NULL, PostDate, PointChange, 6, NULL FROM [dbo].[SourceCancelledOrders]**

There is some manual data clean up to do! This will be detailed after the Data Flow portion.

#### Data Flow

Data is taken from the Postgres database from the store\_itemcode Table.

#### Derived Column

The current Date Time is gotten and used for the two new MS columns CreatedDate and ModifiedDate.

### Manual Data Clean Up

Important note, it is not reliable that the same new MS will be consistently mapped. So use these script as a tool to correct the data do not just run them.

#### Transaction Table corrections.

Original source order id 337 translates into three orders and the point change for each needs manual correction. The new point changes are -1443.0, -8658.0, -1154.0 (there is a rounding issue with the calculated exchange rate).

Original source order id 702 translates into two orders and the point change for each needs manual correction. The new point changes are -866.0 and -1444.0 (there is a rounding issue with the calculated exchange rate).

Original Source order id 2515 point changes need to be -1980.0.

Original source order id 2596 translates into two orders and the point change for each needs to be manually corrected. The points need to be updated to 1200.0 and 4200.0 (add 200 to each).

Original source order id 5280 translates into two orders but only one needs updating. The none -10000 point change need to be changed to 3251.0 (old value should be -3101.0)

Original source order id 1008 and 1007 store order was corrected previously but double check again that the point change for both are -1.0.

Original source order id 5066 translates into two orders and the one that row for store order that was wrong was corrected earlier but double check that the transaction row is correct, a -10000.0 point change and a -1593.0.

Here are the scripts that were used to correct these issues. The first one is used to get the new order IDs.

**SELECT \***

**FROM [RewardPoints].[dbo].[StoreOrder]**

**where sourceid in (337,5066,702,1007,1008,2515,2596,5280)**

**order by sourceid**

Remember these IDs are from one test run of the SSIS package, might not be the same IDs!!!

**--337**

**select \* from dbo.RewardTransaction where OrderID in (341,342,343)**

**update dbo.RewardTransaction set PointChange = -1443.0 where [Type] = 5 and OrderID = 343**

**update dbo.RewardTransaction set PointChange = -8658.0 where [Type] = 5 and OrderID = 341**

**update dbo.RewardTransaction set PointChange = -1154.0 where [Type] = 5 and OrderID = 342**

**-- 702**

**select \* from dbo.RewardTransaction where OrderID in (750,751)**

**update dbo.RewardTransaction set PointChange = -866.0 where [Type] = 5 and OrderID = 750**

**update dbo.RewardTransaction set PointChange = -1444.0 where [Type] = 5 and OrderID = 751**

**--1007**

**select \* from dbo.RewardTransaction where OrderID in (1088)**

**update dbo.RewardTransaction set PointChange = -1.0 where [Type] = 5 and OrderID = 1088**

**update dbo.StoreOrder set PricePerItem = 1.0 where SourceID = 1007**

**--1008**

**select \* from dbo.RewardTransaction where OrderID in (1089)**

**update dbo.RewardTransaction set PointChange = -1.0 where [Type] = 5 and OrderID = 1089**

**update dbo.StoreOrder set PricePerItem = 1.0 where SourceID = 1008**

**--2515**

**select \* from dbo.RewardTransaction where OrderID in (2707)**

**update dbo.RewardTransaction set PointChange = -1980.00 where [Type] = 5 and OrderID = 2707**

**-- 2596**

**select \* from dbo.RewardTransaction where OrderID in (2795,2796)**

**update dbo.RewardTransaction set PointChange = -1200.0 where [Type] = 5 and OrderID = 2796**

**update dbo.RewardTransaction set PointChange = -4200.0 where [Type] = 5 and OrderID = 2795**

**--5066**

**select \* from dbo.RewardTransaction where OrderID in (5566,5567)**

**update dbo.RewardTransaction set PointChange = -1593.0 where [Type] = 5 and OrderID = 5566**

**update dbo.StoreOrder set ExchangeRate = 1.593 where SourceID = 5066 and PricePerItem = 50.00**

**--5280**

**select \* from dbo.RewardTransaction where OrderID in (5797,5798)**

**update dbo.RewardTransaction set PointChange = -3251.0 where [Type] = 5 and OrderID = 5798**

#### Lost transactions rows that have now been put back

Because the Transactions for orders were recreated three lost transactions of three orders were recreated.

The Postgres order IDs are 923,924,925.

Order 923 is for item 13 and cost 660 points for user id 42836 username smcn.

Order 924 is for item 9 and cost 400 points for user id 25845 username colonpowell.

Order 925 is for item 9 and cost 400 points for user id 5967 username Sunn.

It has been decided that we will just use the bonus pool points to correct this oversight. Use the web site to give each user the points back.

#### Three user with negative point balances.

Three Users where found to have negative balances because of the expire points process.

User 124096 with -5127.1171

User 1538261 with -1048.3381

User 40594 with -53.0475

Bonus points will be given to give them a zero balance.

#### Users with test data points

Not sure yet what we will be doing with these points, most should of expired long time ago and the new Stored Procedure that expires points will do that. The one case of CAMERON having a canceled order that there is not actual order and getting back 1000 points should not be corrected because this is a correction of his point total to his actual point total.

ID 86755

ckknight -- CAMERON!!!!!

In Postgres there is a record in the log\_entries of a cancel order number four for 1000 points but there is no such order or other record of it being canceled.

Since there is no order of cancellation of it in the new system a transaction of a positive 1000 points was not generated.

Because of the low number for the order number this is a test data that was not properly cleaned up and it just so happen to give him and extra 1000 points. we are doing nothing in the new system because those 1000 points are from an data clean up error he made and did not earn those points so they will disappear for his total.

He will go from 3050 total points to 2050.

CAMERON!!!!!!!!!!!!!!!!!!!!!!!!!!!

id 1804736

tonpu -- James Wu  [jwu@curse.com](mailto:jwu@curse.com)

Looks to me like he was given some bonus points then made some test orders and canceled them and CAMERON did not clean up the test data properly again.

He might still work for curse.

Interestingly these points are all from 2009 and have not been expired

(I have a feeling that when i run my point expire I am going to find a lot of points that the current one have not expired)

This is a 200 point difference (PG has More)

also all his points came from bonus pool (real points or test points????)

and there more canceled orders than spends

572631;1804736;1003;166;"2010-07-12 18:37:06.33429";-200.0000;"b"

279474;1804736;7;166;"2009-07-02 16:30:00.846268";200.0000;"c"

279473;1804736;8;166;"2009-07-02 16:29:56.43828";200.0000;"c"

279472;1804736;10;166;"2009-07-02 16:29:52.424138";200.0000;"c"

279442;1804736;23;166;"2009-05-12 17:23:58.560187";-200.0000;"b"

279441;1804736;1;226;"2009-05-12 00:14:40.768263";200.0000;"n"

279440;1804736;22;166;"2009-05-11 23:37:51.339907";-200.0000;"b"

279439;1804736;1;226;"2009-05-11 21:56:52.88826";200.0000;"n"

CAMERON!!!!!!!!!

1918371

Magus090 - old curse employee

More test data not clean up

279457;1918371;25;166;"2009-06-09 18:19:33.961029";-400.0000;"b"

279456;1918371;24;166;"2009-06-09 17:55:24.273936";-351.0000;"b"

This is a positive points to the user he was given 1000 bonus points (these might have been made up bonus points)

This is a 751 point difference (MS has the more)

CAMERON!!!!!!!!

2141038

James Wu again

more test data, all point are from a bonus point given that might be fake points still recorded.

this one is a 200 point difference (PG has more)

# Changes to the old store to make it better -- New Store Process

In the old store you could purchase multiple types of items in one order and there was an issue with someone going and putting in their cart X number of these then by the time they got to check out part of their order would not be available anymore because someone else bought the last of one of the items they had in their cart.

It has been decided to simplify the order process and eliminate the cart all together. Now you place an individual order for each item you want with the amount of that item you want. If the customer wants multiple Items they will have same number of orders, but if wants multiple of one item (quantity more than one of item X) they only need to place one order.

As noted earlier the schema has changed to support this new one item per order concept.

In the Postgres database there was a store\_itemlogentry Table that went with the store\_order Table. The itemlogentry held what item was ordered how many, point cost and usd\_cost per item, and the store\_order held userID and total cost. The new StoreOrder table has the exchange rate, Quantity and price per Item so total cost can be calculated. Both have created date, modified date and status.