Reward Points Operating Processes

[Monthly Ran Stored Procedures 5](#_Toc369528600)

[Generate Monthly Budget 5](#_Toc369528601)

[Purpose of the Stored Procedure 5](#_Toc369528602)

[Variables 5](#_Toc369528603)

[Passed In 5](#_Toc369528604)

[Output 5](#_Toc369528605)

[Other Stored Procedures Called 5](#_Toc369528606)

[General Outline and Algorithms and Formulas Used 5](#_Toc369528607)

[Unique Download Count for a Month 6](#_Toc369528608)

[Purpose of the Stored Procedure 6](#_Toc369528609)

[Variables 6](#_Toc369528610)

[Passed In 6](#_Toc369528611)

[Output 6](#_Toc369528612)

[General Outline and Algorithms and Formulas Used 7](#_Toc369528613)

[Expired Unused Points Over 3 Years Old 7](#_Toc369528614)

[Purpose of the Stored Procedure 7](#_Toc369528615)

[Variables 7](#_Toc369528616)

[Output 7](#_Toc369528617)

[General Outline and Algorithms and Formulas Used 7](#_Toc369528618)

[Daily Ran Stored Procedures/SSIS Package 8](#_Toc369528619)

[Order for the Store Procedures and SSIS package to run 8](#_Toc369528620)

[Generate Daily Budget 8](#_Toc369528621)

[Purpose of the Stored Procedure 8](#_Toc369528622)

[Variables 8](#_Toc369528623)

[Other Stored Procedures Called 8](#_Toc369528624)

[General Outline and Algorithms and Formulas Used 8](#_Toc369528625)

[SSIS Package to get User and Project Data from Postgres 9](#_Toc369528626)

[Control Flow 9](#_Toc369528627)

[Data Flow 10](#_Toc369528628)

[Black List 10](#_Toc369528629)

[Data Scripts 10](#_Toc369528630)

[Project 10](#_Toc369528631)

[Project Dependency 11](#_Toc369528632)

[Source Reward Split 11](#_Toc369528633)

[Data Conversion 13](#_Toc369528634)

[User 13](#_Toc369528635)

[Data Scripts 13](#_Toc369528636)

[User Eligibility 14](#_Toc369528637)

[Purpose of the Stored Procedure 14](#_Toc369528638)

[General Outline and Algorithms and Formulas Used 14](#_Toc369528639)

[Library Eligibility 14](#_Toc369528640)

[Purpose of the Stored Procedure 14](#_Toc369528641)

[Variables 15](#_Toc369528642)

[Output 15](#_Toc369528643)

[Other Stored Procedures Called 15](#_Toc369528644)

[General Outline and Algorithms and Formulas Used 15](#_Toc369528645)

[Award Daily Points to Users 15](#_Toc369528646)

[Purpose of the Stored Procedure 16](#_Toc369528647)

[Variables 16](#_Toc369528648)

[Output 16](#_Toc369528649)

[@errorMessage varchar(max)—returns success or actual SQL error 16](#_Toc369528650)

[Other Stored Procedures Called 16](#_Toc369528651)

[General Outline and Algorithms and Formulas Used 16](#_Toc369528652)

[Schema 18](#_Toc369528653)

[Main Tables 18](#_Toc369528654)

[Daily Budget Table 18](#_Toc369528655)

[Table Definition 18](#_Toc369528656)

[Schema 18](#_Toc369528657)

[Columns 18](#_Toc369528658)

[Foreign Keys 18](#_Toc369528659)

[Constraints 18](#_Toc369528660)

[Special note 19](#_Toc369528661)

[Monthly Budget Table 19](#_Toc369528662)

[Table Definition 19](#_Toc369528663)

[Schema 19](#_Toc369528664)

[Columns 19](#_Toc369528665)

[Note for consideration 19](#_Toc369528666)

[Constraints 20](#_Toc369528667)

[Reward Split Table 20](#_Toc369528668)

[Table Definition 20](#_Toc369528669)

[Special Note—Change to ZERO PERCENT SPLITS!!!! 20](#_Toc369528670)

[Schema 20](#_Toc369528671)

[Columns 20](#_Toc369528672)

[Foreign Keys 21](#_Toc369528673)

[Constraints 21](#_Toc369528674)

[Project Blacklist Table 21](#_Toc369528675)

[Table Definition 21](#_Toc369528676)

[Schema 21](#_Toc369528677)

[Columns 21](#_Toc369528678)

[Transactions Table 21](#_Toc369528679)

[Table Definition 21](#_Toc369528680)

[Schema 21](#_Toc369528681)

[Columns 21](#_Toc369528682)

[Foreign Keys 22](#_Toc369528683)

[Constraints 22](#_Toc369528684)

[Indexes 22](#_Toc369528685)

[Transactions Breakdown Table 22](#_Toc369528686)

[Table Definition 22](#_Toc369528687)

[Schema 22](#_Toc369528688)

[Columns 22](#_Toc369528689)

[Indexes 22](#_Toc369528690)

[Rewards Program Table 22](#_Toc369528691)

[Table Definition 22](#_Toc369528692)

[Schema 23](#_Toc369528693)

[Columns 23](#_Toc369528694)

[Constraints 23](#_Toc369528695)

[Special note 23](#_Toc369528696)

[Support Tables 23](#_Toc369528697)

[Legacy Project Table 23](#_Toc369528698)

[Table Definition 23](#_Toc369528699)

[Schema 23](#_Toc369528700)

[Columns 23](#_Toc369528701)

[Foreign Keys 24](#_Toc369528702)

[Legacy User Table 24](#_Toc369528703)

[Table Definition 24](#_Toc369528704)

[Schema 24](#_Toc369528705)

[Columns 24](#_Toc369528706)

[Foreign Keys 24](#_Toc369528707)

[Project Dependency Table 24](#_Toc369528708)

[Table Definition 24](#_Toc369528709)

[Schema 24](#_Toc369528710)

[Columns 24](#_Toc369528711)

[Split Source Table 25](#_Toc369528712)

[Table Definition 25](#_Toc369528713)

[Schema 25](#_Toc369528714)

[Columns 25](#_Toc369528715)

[Foreign Keys 25](#_Toc369528716)

[Reward Monthly Ecpm Downloads Table 25](#_Toc369528717)

[Table Definition 25](#_Toc369528718)

[Columns 25](#_Toc369528719)

[Keys 26](#_Toc369528720)

[Constraints 26](#_Toc369528721)

[Triggers 26](#_Toc369528722)

[Indexes 26](#_Toc369528723)

[Reward Project Popularity Table 26](#_Toc369528724)

[Table Definition 26](#_Toc369528725)

[Columns 26](#_Toc369528726)

[Keys 26](#_Toc369528727)

[Constraints 26](#_Toc369528728)

[Triggers 26](#_Toc369528729)

[Indexes 26](#_Toc369528730)

# Monthly Ran Stored Procedures

These three procedures do not need to be ran in any order. Monthly budget does need to run on the first of the month before the daily budget procedure runs.

## Generate Monthly Budget

RP\_GenerateMonthlyBudget is the name of this Stored Procedure.

### Purpose of the Stored Procedure

This stored Procedure will create a row in RewardMonthlyBudget. It calculates the ECPM budget and Estimated Premium Budget. It gets the ECPM downloads via the Stored Procedure RP\_GetUniqueDownloadsForMonth. The other is historical values are gotten from the Table RewardsProgram.

### Variables

#### Passed In

None

#### Output

None, but will use normal return code for success or failure.

Failure code are:

1. 900 – Main Insert transaction failed
2. 901 – did not get ECPM downloads

### Other Stored Procedures Called

1. RP\_GetUniqueDownloadsForMonth

### General Outline and Algorithms and Formulas Used

The first day of the current month is needed in the procedure, it is calculated via the following script:

**CONVERT(date, DATEADD(dd, -(DAY(GETUTCDATE()) -1), GETUTCDATE()), 101);**

That will get the first day of the current month no matter what day of the month it is ran.

It checks to see if the row for the current month already exist before it runs the calculations and inserts the current month. It uses this SQL query to see if it exist:

**SELECT MonthlyBudgetDate FROM [dbo].[RewardMonthlyBudget] WHERE MonthlyBudgetDate = @begingOfMonth**

If it does exist the Stored Procedure returns success without doing any further calculations or processing.

Calculating the Estimated Monthly Budget:

The estimated premium budget trend modifier is selected from the RewardsProgram Table.

Then the past two actual monthly budgets are gotten from this table. If either of these values are null then a default of 1.0 is used. If both past actual budgets are not default (1.0) then the estimated budget is calculated by this formula:

**SELECT @estimatedPremiumBudget = @twoMonthsAgoActualBudget \* (POWER ((@oneMonthAgoActualBudget / @twoMonthsAgoActualBudget), @estimatedPremiumBudgetTrendModifier));**

If only one of the past two months is greater than the default (1.0) then that value is used for the estimated for the month being generated. If both are the default then the default estimated budget is gotten from the RewardsProgram and use for the estimated budget.

Calculating the ECPM Budget:

The first day of the previous month is passed to the Stored Procedure RP\_GetUniqueDownloadsForMonth that returns the number of unique downloads for that month (this is an overall unique downloads, one IP downloads five projects only counts as one unique overall download).

The formula to calculate ECPM Budget is as follows:

**SELECT @ecpmBudget = ROUND((@ecpmDownloads \* @ecpmImpressionsModifier \* @ecpmRevenueModifier \* @ecpmBudgetShareModifier), 2, 1);**

The Modifiers are gotten from the RewardsProgram Table.

## Unique Download Count for a Month

RP\_GetUniqueDownloadsForMonth is the name of this Stored Procedure.

This is now an SSIS package (EcpmDownloads.dtsx )that runs the same query as the procedure but stores the result into a table on SQL02 Elerium.RewardMonthlyEcpmDownloads.

### Purpose of the Stored Procedure

This procedure does a count of the overall unique downloads for the passed in month. It will only count once if a unique IP downloads several files.

### Variables

#### Passed In

@firstDayOfMonthToCalculate datetime

Pass the first day of the month that you want the total for the unique downloads for.

#### Output

@uniqueDownloadsForMonth int OUTPUT

This is the count of unique downloads for the month that was passed in the in param.

### General Outline and Algorithms and Formulas Used

**SELECT @uniqueDownloadsForMonth =**

**(SELECT COUNT(DISTINCT IpAddress)**

**From [DownloadStatistics].[dbo].Record**

**where Date >= @startDate and Date < @endDate)**

The startDate is the first day of the month and the endDate is the last day of the month.

Startdate is a variable that is passed in and the endDate is then derived in the procedure. The procedure can easily be changed to get the unique counts for any time frame, you would just have to add the endDate to the input params. Then change any place that call this procedure.

## Expired Unused Points Over 3 Years Old

RP\_ExpirePoints3YearsOld is the name of this Stored Procedure.

### Purpose of the Stored Procedure

This will calculate any unused points that are over three years old and then expire them.

### Variables

#### Output

@errorMessage varchar(max)—returns success or actual SQL error

### General Outline and Algorithms and Formulas Used

Need to get two variables then use them to calculate how many points a User has that are over three years old.

The two variables are All Time Points and past Three Year Points.

These two sums are from the Transactions Table that hold all points given and used by a User. There is no need to sum the positive and negative sums separately like was done in the Postgres/Python formula.

**SELECT @allTimePoints = (SELECT SUM(PointChange)**

**FROM [RewardPoints].[dbo].[RewardTransaction]**

**WHERE UserID = @cursorUserID);**

**SELECT @pastThreeYearPoints = (SELECT SUM(rt3Y.PointChange)**

**FROM [RewardPoints].[dbo].[RewardTransaction] rt3Y**

**WHERE rt3Y.UserID = @cursorUserID and rt3Y.DateCreated > @date3YearsAgo)**

Then to get how many points need to be expired:

**SELECT @possiblePointsToExpire = @allTimePoints - @pastThreeYearPoints;**

**If this is a positive number then that amount should be expired.**

# Daily Ran Stored Procedures/SSIS Package

## Order for the Store Procedures and SSIS package to run

The Daily Budget does not need the SSIS Package nor does the SSIS Package need the Daily Budget, but these two need to run before the next three.

1. Generate Daily Budget
2. SSIS Package Users and Project
3. User Eligibility
4. Library Eligibility
5. Award Daily Points to Users

These are ran by code, Cobalt.Elerium.Jobs.RewardPointsCalculationJob

**THIS HAS TO RUN AFTER the job DLS\_UpdateProjectPopularity!!! It is on the DownLoadStatistics DB Server!**

## Generate Daily Budget

RP\_GenerateDailyBudget is the name of this Stored Procedure.

### Purpose of the Stored Procedure

This will calculate the amount of points that can be distributed out to the Users for the day it is run.

### Variables

The normal return integer with return error codes:

1. 901 -- Main Insert transaction failed

### Other Stored Procedures Called

1. RP\_GenerateMonthlyBudget only if it has not been run for the month.

### General Outline and Algorithms and Formulas Used

First thing done are two checks.

First check is to see if the procedure has run for the day already. It does this by querying on this:

**IF (SELECT DailyBudgetDate FROM [dbo].[RewardDailyBudget] WHERE DailyBudgetDate = @todaysDate) IS NOT NULL**

If that is not null then it returns normal with success.

The second check is to see if the monthly budget is available and if it is not then it calls the stored procedure to generate the monthly budget. It checks this with this query:

**IF (SELECT MonthlyBudgetDate FROM [dbo].[RewardMonthlyBudget] WHERE MonthlyBudgetDate = @begingOfMonth) IS NULL**

Next it gets all the needed variables from the RewardsProgram Table and the MonthlyBudgetID.

Now it determine the amount of points to be distributed for this day’s budget. It does this by first getting the total amount of points available for the month ( (RewardMonthlyBudget.PremiumBudget(or Estimated) plus RewardMonthlyBudget.EcpmBudget) time the point exchange rate). Then a sum of how many points have been already given out in daily rewards by summing the RewardDailtBudget.DailyPoints where the MonthlyBudgetID is this month’s ID (if null that means zero). Then minus the points used from total monthly available.

**SELECT @pointsRemainingForMonth = @monthlyPointBudget - @pointsUsedThisMonth;**

@monthlyPointBudget is either the estimated budget or the premium budget plus the ECPM budget times the point exchange rate.

After that the number of days left in the month is gotten by:

**SELECT @daysLeftInMonth = DATEDIFF(day, GETUTCDATE(), DATEADD(month, 1, DATEADD(day, 1-day(GETUTCDATE()), GETUTCDATE())));**

Then points left is divided by days left in the month to get how many points will be given out for the day:

**SELECT @dailyPoints = @pointsRemainingForMonth/@daysLeftInMonth;**

## SSIS Package to get User and Project Data from Postgres

UserEligibility.dtsx is the name of the SSIS package.

### Control Flow

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

The before scripts truncates all related tables and reseeds the RewardSplitSource Table:

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

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

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

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

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

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

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

The Table RewardSplit is truncated here because its new set of on filtered data is going into its source Table RewardSplitSource. At any time one of these two Tables should be empty.

The after script runs a query that corrects a Coalesce that was put in to fix a reading of a nullable column issue.

**Update [dbo].[RewardLegacyProject] set DefaultFileID = null where DefaultFileID = -99;**

### Data Flow

Here are the five tables that data is put into form the Postgres database.

#### Project

This gets a list of all Projects and its current Owner’s ID, Default File ID, Status, Stage and name. The Postgres Project ID is preserved in the new Tables ID column that is a Primary Key but does not have an Identity Specification.

The query to get the list of Projects form Postgres is:

**select a.project\_id,**

**a.manager\_id,**

**b.name,**

**coalesce(( SELECT f.id**

**FROM files\_file f**

**join repositories\_repository as repo on f.repository\_id = repo.id**

**WHERE repo.project\_id = a.project\_id**

**AND f.is\_alternate = false**

**AND f.status NOT IN ('v', 'd', 'c', 'a', 'u', 'w') and repo.name = 'mainline'**

**ORDER BY f.type desc, f.post\_date desc LIMIT 1),-99) as defaultFileId,**

**b.status,**

**b.stage**

**from repositories\_repository as a, projects\_project as b**

**where a.project\_id = b.id and a.name = 'mainline'**

The where clause for name = ‘mainline’ is to filter out clones of the same project. The Coalesce is there because SSIS was having an issue with the Default File ID being NULL so it is set to -99 then later changed to null.

#### Project Dependency

This Table holds the project relationships for what projects are used as libraries. The query that pulls the data from Postgres does very little filtering so the Stored Procedure does the bulk of the filtering. It has to be this way because some data is needed then it is discarded after it is used in one part of the algorithm (see Library Eligibility).

The query is:

**select distinct f.id as File\_ID, repo2.project\_id as DependencyProjectID**

**FROM files\_relationship as rel**

**join files\_file as f on f.id = rel.file\_id**

**join repositories\_repository as repo on f.repository\_id = repo.id**

**join repositories\_repository as repo2 on rel.repository\_id = repo2.id**

**join projects\_project as p2 on repo2.project\_id = p2.id**

**WHERE**

**rel.type IN ('e', 't')**

**order by DependencyProjectID,File\_ID**

The Where clause type in (‘e’,’t’) is it must be an embedded project or a tool.

#### Source Reward Split

This Table holds the unfiltered User splits from the Postgres database. There is a schema change from the Postgres database to the new database. The old way was to only have an entry if it was not the Owner getting 100% and there was a condition that would put a zero% entry into the table. Also this table did not know if the Project or User was eligible it is just the raw data that the individual Owners had set as how they wanted their splits to be done if all User they have given a split to are eligible. Note when we do Phase two we will still have to have this raw data Table and the current Table of this is how the split ends up after eligibility is ran. The zero% in the old schema was created when Ownership changed hands. This caused an interesting side effect that was not deterministic in how it acted, if the current new owner went inactive it then looked for zero% splits to give the inactive owners points too. Sometimes it tried to split other inactive Users splits to both zero% and current Owner. These Zero% splits could be cause by moderators changing owners by mistake then changing it back. The concept was not well planned out and not properly executed so the concept is being removed in Phase one, if there is a need to introduce this back in Phase Two then it can be fully conceptualized and planed out then.

The query to get the data from the Postgres database is:

**SELECT**

**rs.project\_id,**

**user\_id,**

**percentage**

**FROM**

**rewards\_projectrewardsplit rs**

**left join projects\_project p on rs.project\_id = p.id**

**left join repositories\_repository r on rs.project\_id = r.project\_id**

**left join users\_user u on r.manager\_id = u.id**

**where rs.percentage > 0.00 and p.status = 'n' and p.stage not in ('x','d')**

**and u.is\_in\_rewards\_program is true and u.is\_banned = false**

**union**

**select distinct r.project\_id, manager\_id, 1.00 as percentage**

**from repositories\_repository r**

**left join projects\_project p on r.project\_id = p.id**

**left join users\_user u on r.manager\_id = u.id**

**where r.name = 'mainline' and p.status = 'n' and p.stage not in ('x','d')**

**and not exists(select null from rewards\_projectrewardsplit rs where rs.project\_id = r.project\_id and percentage > 0.00)**

**and u.is\_in\_rewards\_program is true and u.is\_banned = false**

**order by project\_id**

The first one gets all the projects that have a split entered in the Postgres split table that is not a zero% split. The second part of the UNION gets all projects that do not have an entry in the split Table (other than zero% splits) and get the current owner and sets their split as 100%.

Projects that do not have a status that is ‘n’ or have a stage of ‘x’ or ‘d’ are not pulled over.

##### Data Conversion

The column percentage from Postgres has a data conversion on it is the SSIS package to numeric otherwise it has issues when making the move to the new database.

#### User

This Table hold the Users and if they are banned, in the rewards program and their last login date time.

The query for this is:

**select a.id, a.is\_banned, a.is\_in\_rewards\_program, max(b.last\_date)**

**from users\_user as a**

**left join users\_ipaddressinfo as b on a.id = b.user\_id**

**group by a.id**

##### Data Scripts

Two Data conversions and scripts are used to change two Postgres Boolean values into bits. The two columns are is\_banned and is\_in\_rewards\_program.

Example C# script used to convert the Boolean to a bit after it has been converted to as string is:

**switch (Row.Copyofisbanned)**

**{**

**case "True":**

**Row.NewIsBanned = true;**

**break;**

**case "False":**

**Row.NewIsBanned = false;**

**break;**

**default:**

**Row.NewIsBanned = false;**

**break;**

**}**

## User Eligibility

RP\_UserEligiblity is the name of this Stored Procedure.

### Purpose of the Stored Procedure

Process the RewardSplitSource Table into an all-inclusive list of users AND owners AND their defined percentages of the reward split.

Remember the RewardSplit Table only has in it Projects and Users that are eligible to get points for the day in the DailyBudgetDate.

### General Outline and Algorithms and Formulas Used

The Stored Procedure is well documented.

1.) Remove any duplicate records in source table that are duplicate for projectID AND userID

2.) Remove all project records in source table WHERE the project was blacklisted

a.) Projects can only have a status of N AND Stage cannot be X or D

b.) remove all projects where the owner is not in the rewards program

3.) Process all users in source table who have 100% allocated AND are determined to be eligible.

a.) Remove all records FROM source table WHERE project ID exist FROM inserts FROM step 3

4.) Process all users in source table who have 100% allocated AND are determined NOT to be eligible.

a.) Remove all records FROM source table WHERE project ID exist FROM inserts FROM step 4

5.) Process all users in source table who have between 1% AND 99% allocated AND are determined NOT to be eligible

a.) Remove all records FROM source table WHERE user criteria match what was used in step 5

6.) Process all users in source table who have between 1% AND 99% allocated AND are determined to be eligible

a.) Remove all records FROM source table WHERE user criteria match what was used in step 6

7.) Process all users in source table who have 0% allocated

a.) Remove all records FROM source table

## Library Eligibility

RP\_LibraryEligibility is the name of this Stored Procedure.

This must have a Link to the DownloadStatistics database that will be on a different database server.

### Purpose of the Stored Procedure

This will calculate the library popularity scores of all projects that are determined to be libraries by this procedure.

### Variables

#### Output

None, but will use normal return code for success or failure.

Failure code are:

1. 930 -- Main Insert transaction failed

### Other Stored Procedures Called

1. RP\_GenerateDailyBudget only if it has not been run for the day.

### General Outline and Algorithms and Formulas Used

The query that brings the data from Postgres gets all previous default files so filter it down to just the current Default Files. Then remove Black Listed Projects.

Now the rules followed to determine if a project is a Library:

1. The type of relationship must be either “tool” or “embedded”, type IN ('e', 't') that is filtered in the Postgres query when the data is pulled into the Source table
2. The project must have at least as many other projects using than what the LibraryThreshold is set to in RewardsProgram for it to be a Library (equal to or greater than threshold). At this point Libraries of Libraries contribute to this count, but when it comes to counting the popularity scores from Libraries of Libraries you do not count those popularity scores.
3. The Library Popularity Score is calculated by getting the popularity score of each project that uses the Library divided by how many Libraries it uses. Example, Library called “Math” is used by three projects and each of those projects use ten libraries each. Project “One” is a library of a Library so we do not take its popularity score, project “Two” uses ten libraries including “Math” and has a popularity score of 100.3, so “Math” gets 10 added to its total Library Popularity score from Project “Two”, Project “Three” uses five libraries including “Math” and has a popularity score of 43.2, so “Math” gets 8 points added to its total Library Popularity score for a total of 18. Notice here that the division is cast to a rounded down Integer. This is how it was done in the Postgres/Python formula. This will change to getting decimal numbers after a few runs on the new system.

Yes the Library Popularity score is initial stored in the RewardSplit Table where there can be multiple entries for a project when this data is queried use a distinct on project and do not get the users column and you will get just one row for each project.

## Award Daily Points to Users

RP\_AwardDailyPointsToUsers is the name of this Stored Procedure.

### Purpose of the Stored Procedure

This will divide up the daily points we have and award them to the users, it will also record how many points each project got and for the User what projects contributed how many point to the total it got for that day.

### Variables

#### Output

### @errorMessage varchar(max)—returns success or actual SQL error

### Other Stored Procedures Called

1. RP\_GenerateDailyBudget – only if it has not run yet
2. RP\_UserEligiblity – only if it has not run yet
3. RP\_LibraryEligibility – only if it has not run yet

### General Outline and Algorithms and Formulas Used

Get DailyPoints, addonPoints, LibraryPoints, PopularityThreshold, and fairness from the RewardsDailyBudget.

Then get the Total of the Adjusted Popularity Scores for Addons and for Libraries.

**SELECT @totalOfAddonAdjPopScores = (SELECT sum(POWER(p.PopularityScore, @fairness))**

**FROM (select distinct ProjectID, LibraryPopularity from [RewardPoints].[dbo].[RewardSplit] where Islibrary = 0) AS rs**

**left join [DownloadStatistics].[dbo].[Project] AS p ON rs.ProjectID = p.ID**

**WHERE p.PopularityScore >= @popularityThreshold )**

**SELECT @totalOfLibsAdjPopScores = (SELECT sum(POWER(rs.LibraryPopularity + p.PopularityScore, @fairness))**

**FROM (select distinct ProjectID, LibraryPopularity from [RewardPoints].[dbo].[RewardSplit] where Islibrary = 1) AS rs**

**left join [DownloadStatistics].[dbo].[Project] AS p ON rs.ProjectID = p.ID**

**WHERE rs.LibraryPopularity + p.PopularityScore >= @popularityThreshold )**

The formula for the adjusted popularity score is:

(Library popularity score plus popularity score) to the power of fairness, library popularity will be zero for addons.

To calculate the amount of points that a project has to be split to its Users this is the formula:

((Library or Addon Points) \* Project’s Adjusted Popularity Score) / Library or Addon Total of Adjusted Popularity Score. Floored to the nearest 10,000th place.

The reason it is floored to the ten thousand place is that if it rounded up then there would be more points given out in total that allotted to Libraries and Addons. In the end we calculate how many points that were supposed to be given out compared to how many were given out then take the difference and add it to the bonus pool.

All the above gets stored into the Table RewardProjectReward.

The next Tables that gets data inserted are the RewardTransaction and RewardTransactionBreakdown. First calculate how many points each Projects gives to each User from the RewardSplit Table and RewardProjectReward Table, this is part of the RewardTransactionBreakdown Table information (for now store it in a temp Table because do not have the Foreign Key to RewardTransaction):

**INSERT INTO #tempTransBreak(projectID, userID, points)**

**Select rpr.ProjectID,**

**rs.UserID,**

**rpr.RewardedPoints \* rs.Percentage as Points**

**FROM [RewardPoints].[dbo].[RewardProjectReward] rpr**

**left join [RewardPoints].[dbo].[RewardSplit] rs ON rpr.ProjectID = rs.ProjectID**

**WHERE rpr.DailyBudgetID = @dailyBudgetID**

**order by rs.UserID**

Now the Temp Table with the above data can be looped trough to sum each Users total points earn for the day and inserted into RewardsTransaction then that ID can be gotten along with the above Temp Table data can be inserted in to RewardTransactionBreakdown.

Once all User have been looped through the sum of points given out to user for that day can be summed then that minus from the total amount of points to be given out that day to get how many points will be going to the bonus pool. This number should be equal to or greater than the total number of point to be given out for the day times the Bonus Percentage for the day. Add it to RewardsProgram.BonusPoints and log the amount added to the RewardDailyBudget. ActualBonusPointsAdded row for the day.

**SELECT @totalPointsGiven = (select sum(PointChange) FROM [dbo].[RewardTransaction] where DailyBudgetID = @dailyBudgetId)**

**SELECT @bonusPoints = @dailyPoints - @totalPointsGiven**

**UPDATE [dbo].[RewardsProgram] SET BonusPoints = (SELECT BonusPoints + @bonusPoints FROM [dbo].[RewardsProgram]);**

**UPDATE [dbo].[RewardDailyBudget] SET ActualBonusPointsAdded = @bonusPoints WHERE ID = @dailyBudgetId;**

# Schema

## Main Tables

### Daily Budget Table

#### Table Definition

This Table hold information about how many points were distributed for the given day. The points to U.S. dollar conversion is set for a month so that is why it is not stored in this table. All the calculated values are done by the stored procedure RP\_GenerateDailyBudget.

#### Schema

No Triggers. Does have the normal PK Index.

##### Columns

1. ID – Primary Key, Auto Incremented, numbers imported from the Postgres database
2. MothlyBudgetId – this is the ID of the RewardMonthlyBudget ID this days budget is getting it points from
3. ActualBonusPointsAdded – this is the actual amount of points put in the bonus pool this day, the rounding left overs are added to the bonus pool.
4. AddonPercentage – This is the percent of the daily budget points that will go to Addons, historical copied from RewardsProgram Table
5. BonusPercetage – This is the percent of the daily budget points that will go to the Bonus pool, calculated by 1 – (Addon% + Library%)
6. DailyBudgetDate – This is the date that the budget infor
7. DailyPoints – This is the amount of points allocated for the day
8. DailyPermiumBudget – This is the amount of U.S. dollar allocated for the day
9. DateCreated – this is the date/time the entry was inserted
10. DateModified – this is the date/time the entry was changed/updated
11. Fariness – this is the fairness use in the calculation of daily Rewards, historical copied from RewardsProgram Table
12. LibraryPercentage – This is the percent of the daily budget points that will go to Libraries, historical copied from RewardsProgram Table
13. LibraryThreshold – This is how many Addons must use you as a Library before are considered to be a Library (Greater than or Equal to this number) , historical copied from RewardsProgram Table
14. PopularityThreshold – This is the minimum popularity score a Project must have for it to receive points, historical copied from RewardsProgram Table

##### Foreign Keys

1. FK\_RewardDailyBudget\_MonthlyBudget – This points to the ID in the RewardsMonthlyBudget Table

##### Constraints

1. CK\_DailyBudget\_AddonPercentage – Addon percent cannot be greater than 1.0
2. CK\_DailyBudget\_BonusPercentage – Bonus percent cannot be greater than 1.0
3. CK\_DailyBudget\_Fairness – Fairness cannot be greater than 1.0
4. CK\_DailyBudget\_LibraryPercentage – Library percent cannot be greater than 1.0

###### Special note

Looking at this I see there can be improvement in these constraints. Instead of the individual percent check, I would rather have one that say the total for Addon, Library, and bonus has to be equal to 1.0.

### Monthly Budget Table

#### Table Definition

This table holds the monthly budget information. How much money in U.S. dollar will be distributed in points for the given month and how many points make a U.S. dollar. It also stores the data that is used to do the calculation from the RewardsProgram Table (except for one piece see note).

#### Schema

No Triggers or Foreign Keys. Does have the normal PK Index.

##### Columns

1. ID – Primary Key, Auto Incremented, same number as in Postgres
2. DateCreated – this is the date/time the entry was inserted
3. DateModified – this is the date/time the entry was changed/updated
4. EcpmBudget – This is calculated by the stored procedure RP\_GenerateMonthlyBudget
5. EcpmBudgetShareModifier – used to calculate ECPM Budget, gotten from RewardsProgram Table and stored here for historical record
6. EcpmDownloads – this is the total unique downloads for the previous month, it is gotten with the stored procedure RP\_GetUniqueDownloadsForMonth. This is not a total of unique downloads of each project but a count of overall unique downloads. If from one IP address five different projects are downloaded then for this count it only counts as one download.
7. EcpmImpressionsModifier – used to calculate ECPM Budget, gotten from RewardsProgram Table and stored here for historical record
8. EcpmRevenueModifier – used to calculate ECPM Budget, gotten from RewardsProgram Table and stored here for historical record
9. EstimatedPremiumBudget – This is calculated by the stored procedure RP\_GenerateMonthlyBudget
10. MonthlyBudgetDate – This is the first day of the month that this budget is for
11. PointExchangeRate – This is how many points to a U.S. Dollar
12. PremiumBudget – this is the amount that accounting comes up with for the monthly budget for points, this is in U.S. Dollar

###### Note for consideration

There is one other column from the rewardsProgram Table that is used in the monthly budget calculation that should be considered to be stored historically in this Table, EstimatedPremiumBudgetTrendModifier. In the Postgres database it was not stored but I think it should be now. We can fill past data with either the current value, zero, or figure out its value by doing a little math. Doesn’t look like it was used from the beginning just when an estimated budget was used.

##### Constraints

1. DF\_MonthlyBudget\_EcpmBudget -- Default value is 0.0
2. DF\_MonthlyBudget\_EcpmBudgetShareModifier -- Default value is 0.0
3. DF\_MonthlyBudget\_EcpmDownloads -- Default value is 0.0
4. DF\_MonthlyBudget\_EcpmImpressionsModifier -- Default value is 0.0
5. DF\_MonthlyBudget\_EcpmRevenueModifier -- Default value is 0.0
6. DF\_MonthlyBudget\_EstimatedPremiumBudget -- Default value is 0.0
7. DF\_MonthlyBudget\_PointExchangeRate -- Default value is 0.0

### Reward Split Table

This Table is truncated daily by a SSIS Package, then a Stored Procedure filters the data in Split Source table into this table!

#### Table Definition

This Table ONLY holds Projects and Users that are eligible to receive reward points! It also holds what percentage each user on the project gets (no Zero percent splits see special note). The LibraryPopularity and IsLibrary coulmns are calculated by the Stored Procedure RP\_LibraryEligibility, they hold respectively the calculated Library Popularity (see stored procedure definition on how this is done) and the bit/Boolean for if it is a library or not.

##### Special Note—Change to ZERO PERCENT SPLITS!!!!

The Postgres database and Python code had a condition for Zero percent split entries. In that schema a zero percent was entered when a manager/owner of a project made someone else the manager. That is the only way a zero percent entry happened. Side effect of this was that if the manager went in active (not logged in for 60 days) the zero point user got the points. The issue is that this was a side effect that was not well planned out and was not consistent in all situation and variations. Also moderator can change managers then realize they made a mistake and change it back, now you have someone with a zero percent split that should not have one. In addition it is unclear if the Users are even aware of this behavior in the reward points program. The current page will display zero percent splits that are not in the table and for users that where never managers, so it would be impossible for a user to even tell if that person would get the mangers split if the manager went inactive. If we decide to implement something like this it will have to be better thought out and planned. As of phase one Zero percent splits are ignored.

#### Schema

No Triggers. Does have the normal PK Index.

##### Columns

1. ID – Primary Key, Auto Incremented
2. ProjectID -- this is the ID of the Project form the Postgres database
3. UserId -- this is the Identity of the User. This is the same User ID from Postgres.
4. Percentage – this is the percentage this user gets for this project
5. LibraryPopularity – this is calculated by the Stored Procedure RP\_LibraryEligibility (see it for calculation)
6. Islibrary – this is a bit/Boolean that is the project is a library it is set to 1
7. DailyBudgetDate – this is a date only column (no time) that hold the date that it was populated by the Stored Procedure RP\_LibraryEligibility. It is checked by the stored procedure RP\_AwardDailyPointsToUsers to see if the data is the current day’s data.

##### Foreign Keys

1. FK\_RewardSplit\_User -- This points to the User Table ID column for the column UserID

##### Constraints

1. DF\_RewardSplit\_Islibrary – Default value is 0 (zero)
2. DF\_RewardSplit\_LibraryPopularity – Default value is 0.0

### Project Blacklist Table

This Table is truncated and data imported daily by a SSIS Package!

#### Table Definition

This Table hold a list of projects that are currently on the black list, meaning they do not eligible for reward points.

#### Schema

No Constraints, Triggers, or Foreign Keys. Does have the normal PK Index.

##### Columns

1. ID – Primary Key, Auto Incremented
2. ProjectID – The Id of the Project that is on the Black List

### Transactions Table

#### Table Definition

This Table holds all User transactions of point changes.

#### Schema

No Triggers.

##### Columns

1. ID – Primary Key, Auto Incremented
2. DailyBudgetID – Nullable, if points form a daily reward then this is the Id of the DailyBudget ID the points came from
3. OrderID – Nullable, if points are because of an order the this is the Order ID
4. TransferUserID – Nullable, this is either the user the points are being sent to (if negative) or the User the points came from (if positive)
5. UserID – this is the Identity of the User. This is the same User ID from Postgres.
6. DateCreated – this is the date/time the entry was inserted
7. PointChange – this is the amount of points this entry changed the users total by
8. Type – this is a numeric code for the type of point change the codes are: PointsGenerated=1;TransferredToUser=2;TransferredFromUser=3;BonusPointsAwarded=4;OrderPlaced=5;OrderCancelled=6;PointsExpired=7;OrderFulfilled=8;OrderRefunded=9;
9. SourceID – This is the Postgres ID, it is here only for the import and then will be removed. It is used to get the Foreign Key FK\_RewardTransactionBreakdown\_RewardTransaction corrected to the new Table ID.

##### Foreign Keys

1. FK\_RewardTransaction\_DailyBudget -- This points to the RewardDailyBudget Table ID column for the column DailyBudgetID
2. FK\_RewardTransaction\_StoreOrder -- This points to the StoreOrder Table ID column for the column StoreOrderID
3. FK\_RewardTransaction\_User -- This points to the User Table ID column for the column UserID
4. FK\_RewardTransaction\_UserByTransferUser -- This points to the User Table ID column for the column UserID

##### Constraints

1. CK\_RewardTransaction\_ID\_FK\_OnlyOneOrNone – only one of the following column can be NOT NULL the other two have to be null or all three can be null. StoreOrderID, DailyBudgetID,TransferUserID
2. CK\_RewardTransaction\_Type -- ([Type]=(9) OR [Type]=(8) OR [Type]=(7) OR [Type]=(6) OR [Type]=(5) OR [Type]=(4) OR [Type]=(3) OR [Type]=(2) OR [Type]=(1))

##### Indexes

1. IX\_RewardTransaction\_Type – index on the Type Column
2. IX\_RewardTransaction\_UserID – index on the UserID column

### Transactions Breakdown Table

#### Table Definition

This Table holds break down of the RewardTransaction Table entries that are the Daily Reward for a User, It breaks down the Users total points by the projects the points came from.

#### Schema

No Constraints or Triggers.

##### Columns

1. ID – Primary Key, Auto Incremented
2. ProjectID – this is the ID of the Project form the Postgres database
3. RewardTransactionID – This is the ID from the RewardTransaction Table
4. Points – This is the amount of points the particular project contributed to the user for their total amount of point for the given transaction

##### Indexes

1. Normal PK index
2. IX\_RewardTransactionBreakdown\_RewardTransactionID – index on column RewardTransactionID

### Rewards Program Table

#### Table Definition

This Table is to hold information on how the Program Rewards are calculated. There will only be one row in this Table. It also hold the Bonus Points total. The reason there is only one row with data on how the rewards are calculated is because every time the data is used it is stored in the table that used it so we have a historical value what is was at the time it was used by the table that used it. Also additions to the Bonus Pool are tracked by the RewardDailyBudget Table.

#### Schema

No Triggers or Foreign Keys. Does have the normal PK Index.

##### Columns

1. ID – Primary Key, one row it is 1
2. BonusPoints – This is the total of Bonus Points available to distribute
3. Fairness – this is the fairness use in the calculation of daily Rewards
4. LibraryThreshold – This is how many Addons must use you as a Library before are considered to be a Library (Greater than or Equal to this number)
5. AddonPercentage – This is the percent of the daily budget points that will go to Addons
6. LibraryPercentage – This is the percent of the daily budget points that will go to Libraries
7. PopularityThreshold – This is the minimum popularity score a Project must have for it to receive points
8. EcpmImpressionsModifier – This is used in the calculation of monthly ECPM Budget
9. EcpmRevenueModifier -- This is used in the calculation of monthly ECPM Budget
10. EcpmBudgetShareModifier -- This is used in the calculation of monthly ECPM Budget
11. EstimatedPremiumBudgetDefault – this is used if no Estimated Premium budget can be calculated
12. PointExchangeRate – This is how many points make one U.S. Dollar
13. EstimatedPremiumBudgetTrendModifier -- This is use in the calculation of the monthly Estimated Premium Budget

##### Constraints

1. CK\_RewardsProgram\_AddonPercentage – Addon percent cannot be greater than 1.0
2. CK\_RewardsProgram\_Fairness – Fairness cannot be greater than 1.0
3. CK\_RewardsProgram\_LibraryPercentage – Library percent cannot be greater than 1.0

###### Special note

Looking at this I see there can be improvement in these constraints. Instead of the individual percent check, I would rather have one that say the total for Addon + Library has to be equal or less than to 1.0.

## Support Tables

### Legacy Project Table

This Table is truncated and data imported daily by a SSIS Package!

#### Table Definition

This Table is to hold information on Projects that is used in determining Library counts, Reward Splits, and Project Eligibility. This is a Phase One only Table that will be eliminated in a later Phase, the eventual goal is to Project Table (or some other version of two combined schemas).

#### Schema

No Constraints or Triggers. Does have the normal PK Index.

##### Columns

1. ID – Primary Key, not from Postgres, Auto Incremented
2. OwnerID – This is the User ID of the Owner/Manager of the Project
3. DefaultFileID – this is the File ID of the current Release of for the Project (zip file)
4. Name – this is the name of the Project (here for ease of look up)
5. Stage – this is the stage, it is used for determining Project eligibility must not be ‘x’ or ‘d’
6. Status – this is the status, it is used for determining Project eligibility must Foreign Keys be an ‘n’

##### Foreign Keys

1. FK\_RewardLegacyProject\_User -- This points to the User Table ID column for the column UserID

### Legacy User Table

This Table is truncated and data imported daily by a SSIS Package!

#### Table Definition

This Table is to hold information on Users that is needed to determine if they are eligible to be awarded points each day. This is a Phase One only Table that will be eliminated in a later Phase, the eventual goal is just to use the Table User.

#### Schema

No Constraints or Triggers. Does have the normal PK Index.

##### Columns

1. UserID - this is the Identity of the User. This is the same User ID from Postgres.
2. IsBanned – this is a bit/Boolean if the user is banned then equal 1.
3. IsBlockedFromUsingPoints – this is a bit/Boolean that indicates the User may not use the store but still is getting points, used prevent someone from spending point that we might have to take away
4. IsInRewardsProgram- this is a bit/Boolean if the user is in the Rewards Program the equal 1.
5. LastLogin – this is the recorded last login for the user.

##### Foreign Keys

1. FK\_RewardLegacyUser\_User – This points to the User Table ID column for the column UserID

### Project Dependency Table

This Table is truncated and data imported daily by a SSIS Package!

#### Table Definition

This table is to hold the unfiltered Library use data. It is used to calculate how many Projects use a Library and if a Project qualifies as a Library.

#### Schema

No Foreign Keys, Constraints, or Triggers. Does have the Normal PK Index.

##### Columns

1. DefaultFileID – Current file(zip of the project files) that is the latest release
2. DependencyProjectID – this is the Project Id that the corresponding Default File is using as a Library or Tool.

The query that bring the data over from Postgres gets even the previous Default Files so part of the cleanup of this table is done in the Stored Procedure RP\_LibraryEligibilty right at the beginning it removes the non-current Default Files.

It does it with this delete command:

**DELETE rpd FROM [dbo].[RewardProjectDependency] rpd**

**WHERE NOT EXISTS (**

**SELECT DefaultFileID**

**FROM [dbo].[RewardLegacyProject] rlp**

**WHERE rlp.DefaultFileID = rpd.DefaultFileID**

**)**

The Table RewardLegacyProject has the current default file for each project.

### Split Source Table

This Table is truncated and data imported daily by a SSIS Package!

#### Table Definition

This Table is to hold the unprocessed/raw percentages that a User gets from a Project. The old Postgres data model was to only have an entry if there was something other than a %100 to the Project Manager/Owner. There was also no filtering for User eligibility neither was the row deleted when a percentage was changed from a non-zero percentage to a no percentage.

The only filtering that is done is the query to get the data from Postgres is to remove zero percentage rows.

Managers are added to the table and given a split of %100.

Once the data is imported to this holding table then a Stored Procedure is ran that filters the data and inserts it in the Table RewardSplit. The Stored Procedure that does this filtering is RP\_UserEligibility.

#### Schema

No Constraints or Triggers. Does have the Normal PK Index.

##### Columns

1. ID, this is the Primary Key and is Auto Incremented. It is not a copy of a Postgres existing ID.
2. ProjectID, this is the Identity of the Project. This is the same Project ID from Postgres.
3. UserID, this is the Identity of the User. This is the same User ID from Postgres.
4. Percentage, this is the percentage amount that the User gets for the particular Project.

##### Foreign Keys

1. FK\_RewardSplitSource\_User – This points to the User Table ID column for the column UserID

### Reward Monthly Ecpm Downloads Table

#### Table Definition

This holds the number unique download over a month period. This is not by File or Project, it is just by how many different IP Addresses downloaded any files from us during that month.

#### Columns

1. ID – Primary Key, Auto Incremented
2. UniqueCount – The count of how many IP Addresses have downloaded a File this time frame
3. BeginDate – The start of the time frame
4. EndDate – The end of the time frame

#### Keys

1. Normal PK
2. UK -- IX\_RewardMonthlyEcpmDownloadsUK on BeginDate and EndDate

#### Constraints

None.

#### Triggers

None.

#### Indexes

1. IX\_RewardMonthlyEcpmDownloadsUK Unique, Non-clustered

### Reward Project Popularity Table

#### Table Definition

This holds the popularity score of each Project as pulled over each day from the DownloadStatistics database via a SSIS package UserEligibility.dtsx once a day.

#### Columns

1. ID – Primary Key, Auto Incremented
2. PopularityScore – The popularity score the DownloadStatistics database calulated
3. DatePopularityModified – This is the date from the DownloadStatistics.Project.DatePopularityModified
4. DateModified – This should actually be called created date, it is the date time that the SSIS package ran that pulls the data over.

#### Keys

Primary

#### Constraints

None

#### Triggers

None

#### Indexes

Normal