# Overview

In this project, we will create a new membership history system to drive reports such as the Jasper Membership Statistics report. This data will be in two new tables, membership\_history and membership\_pass\_history, which will be used to drive reports which currently use the existing customer\_membership\_dates table and membership\_since\_dates table.

These tables will be in the ActiveNet databases. In the previous design, they were generated by a proc or procs in the ActiveNet database. In this design, they are generated by the servlet by new logging points in the code.

# Spreadsheet with design details

Some design information is in the “Membership History System Design (servlet implementation).xlsx” file in this same Sharepoint folder:

https://activexperience.active.local/gpmo/projects/communities/ActiveNet\_New/Shared%20Documents/Development/Architecture/Reporting/Membership%20History%20System%20Design%20(servlet%20implementation).xlsx

* The “MH Table” tab contains the schema and enumerations for the membership\_history table.
* The “MPH Table” tab contains the schema and enumerations for the membership\_pass\_history table.
* The “MH Data Examples” tab contains sample membership\_history data for various membership operations.
* The “MPH Data Examples” tab contains sample membership\_history data for various membership pass operations.
* Remaining tabs contain some of the original research into the current system, and can be ignored.

# General requirements

### General

* Each record will have a starting valid datetime (BEG\_DATETIME) and an ending valid datetime (END\_DATETIME).
* If a record has no ending time (membership does not expire), the ending time should be set to “SQL max date” (12/31/9999) rather than “pseudo-null” (12/30/1899) for ease of comparison.
* Once written, the status information in the record is never changed; the only column which will be changed is the ending datetime (END\_DATETIME).

### Membership history

* Keyed by membership\_id.
* Records for a membership\_id should never overlap in time, but there may be gaps if there is a period of time in which a membership is expired and not yet renewed.
* The membership history record should contain key membership status data at the time it is written, which should never change (e.g., package\_id, primarymembercustomer\_id, effective\_date, expiration\_date).
* If the status change was due to a transaction, there should be FK to the originating transaction.
* New records should be written in the following cases:
  + Any membership transaction (subsystem\_code=5\_
  + Any void of a membership transaction
  + Any change made in the membership status page:
    - Effective date
    - Expiration date
    - Suspension begin date
    - Suspension end date
    - Suspended reason
    - Primary member
    - Max users
  + The effective status changes because a date threshold is passed (effective date, date suspended to, date suspended from).
* The history record will have an enum to describe the event creating the record (HISTORY\_TRANSACTION\_TYPE).

### Membership pass history

* Keyed by membership\_id / pass\_id.
* Records for a membership\_id / pass\_id \_id should never overlap in time, but there may be gaps if there is a period of time in which a customer was removed from a membership then re-added.
* The record will identify when a customer began being a member of a specific membership, and when then ended being a member.
* There will be an FK to the originating transaction; if there is no transaction (e.g., manually adding or removing members), it will be null.
* A new record should be written in the following cases:
  + Member added during any transaction.
  + Member added manually.
  + Pass suspended
  + Pass unsuspended
* The ending date of a record should be set if:
  + Member deleted during any transaction.
  + Member deleted manually.
  + A new record is written due to a change in the suspension status
* The history record should have an enum to describe the event creating the record (HISTORY\_TRANSACTION\_TYPE).

# Logging design

This section gives some narrative description of the servlet logic, to accompany the “Data Examples” in the spreadsheet. It is broken down into a series of deliverables.

## General

History record must be written in same DB transaction in which the DB changes are made.

## Common MH column values

Regardless of the cause of the MH insert, the following values are set:

* END\_DATESTAMP:
  + if M.DATE\_EXPIRES=12/30/1899, END\_DATESTAMP = 12/31/1899
  + If M.DATE\_EXPIRES<>12/30/1899, END\_DATESTAMP = M.DATE\_EXPIRES + 1 day
  + When a new record is written, the previous records END\_DATESTAMP will be set to the MIN(prev.END\_DATESTAMP, new. BEG\_DATESTAMP)
* MEMBERSHIP\_STATUS: This is the “effective status”, taking into account the status and the dates effective, expired, suspended from and suspended to.
  + If M.STATUS not in (1,5) then MEMBERSHIP\_STATUS = 0 (Inactive) (sale not complete, or membership refunded or voided),
  + If now<M.DATEEFFECTIVE, then MEMBERSHIP\_STATUS = 0 (Inactive) (membership not effective yet).
  + If M.DATEEXPIRES<>12/30/1899 and now>=M.DATEEXPIRES +1, then MEMBERSHIP\_STATUS =0 (Inactive) (Membership expired)
  + If M.DATESUSPENDEDTO<>12/30/1899, and now>=M.DATESUSPENDEDTO, then MEMBERSHIP\_STATUS = 1 (Active) (Membership was suspended, but suspension is over).
  + If M>DATESUSPENDEDFROM<>12/30/1899 and now>=M.DATESUSPENDEDFROM, then MEMBERSHIP\_STATUS=2 (suspended)
* DATE\_EFFECTIVE = M.DATEEFFECTIVE
* DATE\_EXPIRES = M.DATEEXPIRES
* DATE\_SUSPEND\_FROM = M.DATESUSPENDFROM
* DATE\_SUSPEND\_TO = M. M.DATESUSPENDFROM
* SUSPENDED\_REASON\_ID = M.E SUSPENDED\_REASON\_ID
* PACKAGE\_ID = M.PACKAGE\_ID
* SITE\_ID=PACKAGES.SITE\_ID
* PACKAGE\_CATEGORY\_ID = PACKAGES.PACKAGECATEGORY\_ID
* RETENTION\_ELIGIBLE = ABS(ISNULL(PACKAGE\_CATEGORIES.RETENTION\_ELIGIBLE, 0))
* PRIMARYMEMBERCUSTOMER \_ID = M.PRIMARYMEMBERCUSTOMER\_ID
* AUTORENEWAL\_TYPE = M. AUTORENEWALTYPE
* MAX\_USES = M.MAXUSES

## Membership\_History from Transactions

Whenever a new membership transaction is saved, a corresponding membership\_history record should be written. General considerations are:

* Should include all membership transactions:
  + Process only transaction types with subsystem\_code=5.
  + BEG\_DATESTAMP = t.DATESTAMP
  + SYSTEMUSER\_ID = t. SYSTEMUSER\_ID
  + VOIDED=0
* Should include voids of membership transactions:
  + Process only transaction types 29 (sale), 30 (cancellation), 32 (transfer out) and 33 (renewal), for voided<>0.
  + BEG\_DATESTAMP= t.VOIDEDON
  + SYSTEMUSER\_ID = t.VOIDEDBY
  + VOIDED=-1
* MEMBER\_SINCE\_DATE is populated like this:
  + If the package has no package category, or the package is not retention-eligible, MEMBER\_SINCE\_DATE = null. Otherwise, if the package is retention-eligible:
  + If there is no previous history record, or the previous record had a null MEMBER\_SINCE\_DATE (e.g., this was a transfer from a non-retention-eligible package), MEMBER\_SINCE\_DATE = BEG\_DATESTAMP. Otherwise, if the previous record did have a since date:
  + If the previous record has not expired, or the difference of previous end\_datestamp and the new beg\_datestamp is less than the configured grace period, MEMBER\_SINCE\_DATE = the value from the previous record. (I.e., their membership remains active).
  + Otherwise, MEMBER\_SINCE\_DATE = BEG\_DATESTAMP. (I.e., the customer’s membership starts over).
  + The grace period (in days) is determined with the following query:   
    **select convert(int, convert(varchar, keywordvalue)) from systeminfo where keyword='retention\_period'**
* HISTORY\_TRANSACTION\_TYPE = function of t.transactiontype and t.voided; see spreadsheet.

## Membership\_History from membership admin operation

Write a new record if:

* If any of the following columns in the last membership\_history record differ from the corresponding column in the membership record:
  + AUTORENEWAL\_TYPE
  + PRIMARYMEMBERCUSTOMER \_ID
  + DATE\_EFFECTIVE
  + DATE\_EXPIRES
  + DATE\_SUSPEND\_FROM
  + DATE\_SUSPEND\_TO
  + SUSPENDED\_REASON\_ID
  + Effective MEMBERSHIP\_STATUS
  + MAX\_USES

Record contents:

* BEG\_DATESTAMP = time of generation
* HISTORY\_TRANSACTION\_TYPE = see spreadsheet
* TRANSACTION\_ID = NULL
* SYSTEMUSER\_ID = current user
* MEMBER\_SINCE\_DATE = Carry forward from previous record
* Other columns based on corresponding values in membership record

## Membership\_History from periodic comparison of current membership data to last history record in a thread

Handles the following cases:

* Membership becomes effective
* Membership suspension comes into effect
* Membership suspension period ends

To handle these cases, a daily thread will write a new record if:

* Memberships becoming effective: Membership was created with an effective date in the future. As a result, it should begin with membership\_status=0 (inactive). At the time of the first run after effective date, membership\_status should = 1 (active)
* Memberships becoming suspended: Membership was active, then a suspension from date was entered. If that date was after the date of the status update, at the time of the first run after suspensions from date, membership\_status should = 2 (suspended)
* Memberships becoming un-suspended: Membership was suspended, and a suspension to date was entered. At the time of the first run after suspension to date, membership\_status should = 1 (active).

Record contents:

* BEG\_DATESTAMP = MAX (beginning of day of execution, previous BEG\_DATESTAMP)
* HISTORY\_TRANSACTION\_TYPE = see spreadsheet for type of transition
* TRANSACTION\_ID = NULL
* SYSTEMUSER\_ID = Configured use for background thread
* MEMBER\_SINCE\_DATE = Carry forward from previous record

## Membership\_Pass\_History table

Each time the job runs, it must compare the contents of the membership\_passes (MP) table with the contents of the last membership\_pass\_history (MPH) table record for that membership\_id/pass\_id, and insert new MPH records as needed. All MPH records written will have the following common column values:

* BEG\_DATESTAMP = Time of insert
* END\_DATESTAMP = 12/31/9999 (they don’t expire, even when the membership does)
* MEMBERSHIP\_ID = Value from the MP record
* PASS\_ID = Value from the MP record
* CUSTOMER\_ID = PASSES.CUSTOMER\_ID

Also, any time a new MPH record is added for a membership\_id/pass\_id with existing MPH records, the last MPH record must have its END\_DATESTAMP set to the time of the job run, so it equals the BEG\_DATESTAMP of the next record.

Here are the three cases which must be handled:

### Effective status

Similar to MH, the effective status of a MP record is defined as follows:

* If M.DATESUSPENDEDTO<>12/30/1899, and now>=M.DATESUSPENDEDTO, then MEMBERSHIP\_PASS\_STATUS = 1 (Active) (Membership was suspended, but suspension is over).
* If M.DATESUSPENDEDFROM<>12/30/1899, and now>=M. DATESUSPENDEDFROM, then MEMBERSHIP\_PASS\_STATUS = 2 (Suspended).
* Else MEMBERSHIP\_PASS\_STATUS=1 (Active)/

### New pass, or same pass re-added

There is a MP record for a membership\_id/pass\_id, but either no MPH record, or an MPH record with pass\_status=0. Insert an MPH record with contents:

* MEMBERSHIP\_PASS\_STATUS = Effective status if of pass
* HISTORY\_TRANSACTION\_TYPE = 1 (Added)

### Pass removed

If there is no MP record matching an existing MPH record, for which PASS\_STATUS<>0, insert an MPH record with contents:

* PASS\_STATUS = 0
* HISTORY\_TRANSACTION\_TYPE= 0 (Removed)

### Membership\_pass\_history from periodic comparison of current membership data to last history record in a thread

Handles the following cases:

* Membership\_pass suspension comes into effect
* Membership\_pass suspension period ends
* To handle these cases, a daily thread will write a new record if:

To handle these cases, a daily thread will write a new record if:

* Membership\_pass becoming suspended: If the status is active, an a suspension from date was entered. If that date was entered after the date of the status update, at the time of the first run after suspensions from date, MEMBERSHIP\_PASS\_STATUS should = 2 (suspended)
* Membership\_pass becoming un-suspended: Membership\_pass was suspended, and a suspension to date was entered which was after the date of the status update. At the time of the first run after suspension to date, MEMBERSHIP\_PASS\_STATUS should = 1 (active).

Record contents:

* BEG\_DATESTAMP = MAX (beginning of day of execution, previous BEG\_DATESTAMP)
* HISTORY\_TRANSACTION\_TYPE = see spreadsheet for type of transition
* TRANSACTION\_ID = NULL
* SYSTEMUSER\_ID = Configured user for background thread
* MEMBER\_SINCE\_DATE = Carry forward from previous record

# Test cases

### Validation script

A SQL script has been provided in SVN, which will check the internal integrity of the MH table, and compare it to the membership and transaction data. In addition to the specific test cases below, it should be run each time a set of changes are made, and the job has then had a chance to run. The result of the script is a display of the number of records failing each test; all values should be 0. The script is at:

https://fndsvn.dev.activenetwork.com/ActiveNet/trunk/DBSchema/membership\_history\_validation.sql

### General instructions

The test cases below all involve doing something to a membership or its passes, then checking the results in the MH or MPH tables. The common procedure is:

* Make the specified change.
* Wait for the MH generation job to run.
* For the MH table tests, look at the last MH record for that membership\_id, and compare it to the specified values.
* For the MPH table tests, look at the last MPH record for that membership\_id/pass\_id, and compare it to the specified values.
* In addition to this manual testing, run the validation script.

### Membership\_History

#### Membership transactions

Create the following cases, using a membership package with a period (e.g., weekly):

* Membership 1: Sale
* Membership 2: Sale
* Membership 3: Sale / Transfer
* Membership 4: Sale / Transfer
* Membership 5: Sale / Renewal
* Membership 6: Sale / Renewal
* Membership 7: Sale / Refund

Allow the updates to catch up, so these membership transactions appear in MH. Confirm the contents of the last MH record as follows:

* Membership 1: Sale
  + BEG\_DATESTAMP = Date/time of the sale receipt
  + MEMBERSHIP\_STATUS=1
  + DATEEXPIRES = 1 period
* Membership 2: Sale
  + BEG\_DATESTAMP = Date/time of the sale receipt
  + MEMBERSHIP\_STATUS=1
  + DATEEXPIRES = 1 period
* Membership 3: Sale / Transfer
  + BEG\_DATESTAMP = Date/time of the transfer receipt
  + MEMBERSHIP\_STATUS=1
  + PACKAGE\_ID = New package
  + DATEEXPIRES = 1 period
* Membership 4: Sale / Transfer
  + BEG\_DATESTAMP = Date/time of the transfer receipt
  + MEMBERSHIP\_STATUS=1
  + PACKAGE\_ID = New package
  + DATEEXPIRES = 1 period
* Membership 5: Sale / Renewal
  + BEG\_DATESTAMP = Date/time of the renewal receipt
  + MEMBERSHIP\_STATUS=1
  + DATEEXPIRES = 2 periods
* Membership 6: Sale / Renewal
  + BEG\_DATESTAMP = Date/time of the renewal receipt
  + MEMBERSHIP\_STATUS=1
  + DATEEXPIRES = 2 periods
* Membership 7: Sale / Refund
  + BEG\_DATESTAMP = Date/time of the refund receipt
  + MEMBERSHIP\_STATUS=0

Then make the following changes:

* M1: Refund
* M2: Void
* M3: Refund
* M4: Void
* M5: Refund
* M6: Void
* M7: Void

Allow the updates to catchup, and validate that the last MH record for each membership has the specified values:

* Membership 1: Sale / Refund
  + BEG\_DATESTAMP = Date/time of the refund receipt
  + MEMBERSHIP\_STATUS=0
* Membership 2: Sale / Void
  + BEG\_DATESTAMP = Date/time of the void
  + MEMBERSHIP\_STATUS=0
* Membership 3: Sale / Transfer / Refund
  + BEG\_DATESTAMP = Date/time of the refund receipt
  + MEMBERSHIP\_STATUS=1 (??? Confirm what Activenet does)
  + DATE\_EXPIRES= 1 period
  + PACKAGE = Original package
* Membership 4: Sale / Transfer / Void
  + BEG\_DATESTAMP: Date/time of the void
  + MEMBERSHIP\_STATUS=1
  + DATE\_EXPIRES= 1 period
  + PACKAGE = Original package
* Membership 5: Sale / Renewal / Refund
  + BEG\_DATESTAMP: Date/time of the refund receipt
  + MEMBERSHIP\_STATUS=1
  + DATE\_EXPIRES= 1 period
* Membership 6: Sale / Renewal / Void
  + BEG\_DATESTAMP: Date/time of the void
  + MEMBERSHIP\_STATUS=1
  + DATE\_EXPIRES= 1 period
* Membership 7: Sale / Refund / Void
  + BEG\_DATESTAMP: Date/time of the void
  + MEMBERSHIP\_STATUS=1
  + DATE\_EXPIRES= 1 period

#### Membership status change:

Allow the updates to catch up. Then make the following changes to the membership in the membership status page, and confirm that they are changed as specified in the last membership\_history record (or a report based on it) after the next update run.

Note that this test procedure involves a lot of make one change and wait repetitions. This could be streamlined by creating a bunch of memberships in advance to test each case.

* Effective date:
  + Set the effective date in the future. After the next run, the membership\_status should be 0.
  + Set the effective date in the past. After the next run, the membership\_status should be 0.
* Expiration date (note that the membership only expires on midnight of the day after the expiration date):
  + Set the expiration date in the past. After the next run, membership\_status should be 0.
  + Set the expiration date in the future. After the next run, membership\_status should be 1.
* Suspension:
  + Set the suspension from date in the past, set the suspension to date in the future, and set a suspension reason. After the next run, membership\_status should be 2, and should have the selected suspension reason.
  + Set the suspension from date in the past, and the suspension to date in the past. After the next run, membership\_status should be 1.
* Expiration date set into the past
  + Begin with an active membership. Edit the expiration date to put it in the past. After the next run, membership\_status should be 0.
* Primary member changes
  + Change the primary member. After the next run, the customer\_id should be the new primary member.
* Membership autorenewal type:
  + Change the membership autorenewal type. After the next run, the autorenewal\_type should match the membership.
* Package site change:
  + Change the site of the package of an active or suspended membership. After the next run, the site\_id should match that of the package.
* Package category change:
  + Change the package category of the package of an active or suspended membership. After the next run, the package\_category\_id should match that of the package.
* Package category retention eligibility change:
  + Change the package category of the package of an active or suspended membership. After the next run, the package\_category\_id should match that of the package.

### Membership\_Pass\_History

#### General instructions

For each test you’ll confirm the contents of the last MPH record for the membership\_id / pass\_id combination. For all tests:

* BEG\_DATESTAMP = Time job ran
* END\_DATESTAMP = 12/31/99
* CUSTOMER\_ID = Customer\_id associated with the PASSES record

Also, if there was already one or more MPH records for the membership\_id / pass\_id, look at the previous MPH record, and confirm:

* END\_DATESTAMP = Time job ran (BEG\_DATESTAMP of next record)

#### Test cases

* Create a membership with one pass. Wait for the job to catchup.
  + Confirm there is one MPH record for that pass, and PASS\_STATUS=1
* Add a new customer to the membership, and remove the first one. Wait for the job to catchup.
  + Confirm that there is a new MPH record for the original pass, with PASS\_STATUS=0.
  + Confirm there is an MPH record for the new pass, with PASS\_STATUS=1.
* Enter a suspension from date for the pass that’s in the past. Wait for the job to catchup.
  + Confirm there is a new MPH record for the new pass, with PASS\_STATUS=2.
* Enter a suspension to date for the pass that’s in the past. Wait for the job to catchup.
  + Confirm there is a new MPH record for the new pass, with PASS\_STATUS=1.

## Tasks

* MH KM change
* MH Initial population script
* MH Validation script
* MH generation on membership transaction
* MH generation on membership admin
* MH generation by daily thread
* MPH KM change
* MPH Initial population script
* MPH Validation script
* MPH generation at transaction save
* MPH generation from membership admin