# Overview

When developing the Membership Statistics report, Rae Andrews (in the initial ActiveNet RServer implementation), then the reporting team (in the Jasper implementation), had significant problems with the data quality.

This report requires membership history data, because it reports on things like the number of transfers into or out of a package during a period of time, which can’t be determined simply from the Memberships table record, which contains current status (package, customer, effective and expiration date).

The Membership Statistics reports actually are driven by the Customer\_Membership\_Dates (CMD) table, but that table has both data problems which the existing queries are attempting to work-around, as well as design problems which make it generally difficult to use for membership history reporting.

There is a second table, Membership\_Since\_Dates (MSD), which was developed for the Y market as part of membership retention tracking. In addition to tracking some retention-specific data, it has a somewhat better design as a membership history table. However, its design is still not optimal for general purpose membership history tracking either, and it has some data problems.

Although the original report problem was with CMDs, this analysis will address both CMDs and MSDs, as well as what a good design should be to provide good data for reports in the future.

This document gives a summary of the current status of problems with these two tables which effect reporting, and a proposed strategy for resolving these.

# Related existing Jira tickets

ANE-24721 DB Query on CUSTOMER\_MEMBERSHIP\_DATES causing blocking on ANET Site SeattleYMCA

ANE-21250 Membership transfer may update old row in CUSTOMER\_MEMBERSHIP\_DATES table rather than create new row

ANE-21073 The dates in the CUSTOMER\_MEMBERSHIP\_DATES table are confusing and difficult to report on

ANE-27181 SPIKE: Conduct Analysis on Membership Date issues

ANE-26035 Full stamps to membership dates

ANE-26613 INC2273538 - High Cpu at YmcaGreaterBrandywine causes outage

# General problems identified prior to this analysis

As a general principle, a membership history table should make it possible to easily determine the status of the membership at any point in time, so a simple query including that time should return a single record. In his analysis on ANE-21073, Rae Andrews identified the following issues:

* Lack of good start and end times for history records: When a membership changes state, say by a renewal, the membership history system should write an ending time in the current history record, and write a new record with the same time as the starting time
* Both the CMD and MSD systems record the effective date and expiration date of the membership as it was at the time the record was written. These are dates, not datetimes. Since the actual transitions (renewals, transfers, etc.) occur during the day, this creates overlaps.
* Duplicate history records: Some processes (identified below) create duplicate CMD records.
* Overlapping history records: There are also cases of overlapping CMD records in the data.

# This analysis

The purpose of this analysis is to identify root problems

* Based on ANE-26613, studied Brandywine database for cases of duplicate CMDs and their patterns, and identified some causes.
* Based on ANE-24721, studied cases of high number of duplicates in all US databases, and identified the key usage pattern (refunds of renewals), but not the root cause in code.
* Report usage: Studied usage of CMDs and MSDs in reports.
* Code path research: Studied and code which updates these tables.
* Worked through 19 different use cases, and documented the CMDs and MSDs written by current code, as well as proposed model for a new Membership\_History table.

There is supporting data on this analysis in two uploaded spreadsheets:

* 2015-04-29 CMD research: General research on this ticket, including BrandyWine, Usage in code, and
* 2015-05-04 Duplicate CMD research: Specific study of cases of high number of CMD duplicates across all production DBs.

# Summary of recommendations

* In addition to current bugs, both CMD and MSD designs are fundamentally flawed, in terms of being a good history system.
* The existing bugs should be fixed, especially duplicate creation.
* Fundamental changes to either CMD or MSD systems in place require rebuilding the data from source transactions, and have significant risk of impacting existing reports and code.
* CMDs are difficult to repair in place because of a lack of transaction FKs.
* A better solution would be the creation of a new membership history system, with data rebuilt from transactions, then a phased migration to this table in reports and code.

# CMD problems

Design problems:

* There is no start time for the record.
* There is no single end time column; expiration\_date and termination\_date must be combined.
* A new record is not written for renewals; instead, the existing record is extended. While this works for some reporting purposes, it isn’t a good history model.
* There is no foreign key to the originating transaction (sale, renewal, refund, transfer), so no way of validating or fixing records.

Data problems:

* There are many cases of CMDs which are exact duplicates other than primary keys. (At least two causes have been identified below).
* There are cases of CMDs which overlap in time.

Bugs:

* If during renewal, the expiration date is decreased manually (this does happen), the ending date of the previous record is not changed.
* (JIRA) When the effective date or expiration date of a membership is changed through the membership status UI, the dates of all CMD records for the membership are changed, not just the last one, destroying history and creating duplicates.
* When a renewal is voided, the change date logic is invoked, creating duplicates.
* Under some circumstances TBD, refunding a renew doubles the number of CMD records for the membership, in groups of exact duplicates. In the worst case, there are over 1M duplicated CMDs as a result.
* On a void, the CMD of the record is removed completely. This does not give accurate history of the status of the record.

# MSD problems

Benefits:

* There is a starting date (but no end date)
* There is a foreign key to the originating transaction
* Renewal does add a new record

Design issues

* Because of the purpose for which is was designed, the MSD table only includes records for retention-eligible packages.
* On a transfer, date\_expires is overloaded to be ending date, so the history of the membership status is destroyed.

Data problems:

Based on Brandywine data (not analysis across all orgs):

* Based on comparison against transactions, there are a small number of missing MSDs
* Some MSDs are missing a customer\_id
* There are a small number of duplicate MSDs

Bugs:

* (JIRA) When the effective date or expiration date of a membership is changed through the membership status UI, the dates of all CMD records for the membership are changed, not just the last one, destroying history and creating duplicates.
* When a renewal is voided, the change date logic is invoked, creating duplicates.
* When a member is removed, the MSDs are not changed.
* On a void, the MSD is not modified.

# Design problems common to CMD and MSD

* When the effective date or expiration date of a membership is changed through the membership status UI, a new record should be written.
* When a membership sale, renewal, refund or transfer is voided, a new history record should be written.
* When a member is added or removed to a membership, this should always be logged in such a
* In order to be able to report counts of members, and not “families” (memberships), both CMDs and MSDs duplicate the same data for each member, creating a lot more data than necessary, and don’t consistently record member adds and deletes. Instead, membership history and membership\_pass history should be separated.
* CMD duplicates from renewal refunds

# New Jira bugs

The following Jira bugs have been created as a result of this analysis to repair existing system as much as possible without structural rework:

ANE-29412: Membership\_Since\_dates with null customer\_id's

ANE-29413: Duplicate Membership\_Since\_Dates created by changing membership dates

ANE-29414: Duplicate Customer\_Membership\_Dates created by changing membership dates

ANE-29415: Duplicate Membership\_Since\_Dates created by voiding transaction

ANE-29416: High counts of duplicate Customer\_membership\_dates created by refunding renewals

ANE-29423: DB update to delete duplicate CMD and MSD records

X: MSD’s are not changed when a member is deleted from membership

# Requirements for a good membership history system

### General

* Each record should have a starting valid datetime and an ending valid datetime.
* If a record has no ending time, the ending time should be set to “max date” (12/31/9999) rather than “null” (12/30/1899) for ease of comparison.

### Membership history

* The membership history record should contain key membership status data at the time it is written, which should never change (package\_id, primarymembercustomer\_id, effective\_date, expiration\_date).
* The starting date should never be changed, but the ending date should be changed when a new record is written.
* Records 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.
* There should be FK to the originating transaction; if there is no transaction (e.g., membership date change), it should be null.
* New records should be written in the following cases:
  + Membership sale
  + Membership renewal
  + Membership transfer
  + Refund of any of the above
  + Void of any of the above
  + Change of effective or expiration date
* The history record should have information (an enum) to describe the event creating the record.

### Membership Pass history

* Each record should have a starting valid datetime and an ending valid datetime.
* The record should identify when a customer began being a member of a specific membership, and when then ended being a member.
* The starting date should never be changed, but the ending date should be changed when the member is removed.
* There should be FK to the originating transaction; if there is no transaction (e.g., manually adding or removing members), it should be null.
* A new record should be written in the following cases:
  + Member added during any transaction.
  + Member added manually.
* The ending date of a record should be set if
  + Member deleted during any transaction.
  + Member deleted manually.

Note: The MPH table should be able to implemented at two points in the code, in the data access code in which MPs are inserted or deleted.

# Repair Strategy

The following is an overview of a phased conversion to

### Short term repair

Execute the Jira tickets listed above to solve known cases of duplicates and other bad data in the CMD and MSD tables which cause report performance problems, without making structural changes to these tables. (“New Jira bugs”: ANE-29412 – ANE-29416).

Execute CMD duplicate cleanup script

(Repeat if necessary until no duplicates)

### Phase 1 POC membership history and Jasper report

Create new table membership\_history (MH), including all features of CMDs and MSDs except customer\_id

Create new table membership\_pass\_history (MPH)

Design review with data engineering team.

Write proc to build for a single membership: MH from transactions, and MPH from membership\_passes, and from CMD data when available. It appears to be possible to do this with entirely with set-oriented SQL, so will be very efficient.

Work with data engineering to execute on one test org and ETL into Jasper Dev.

Build new Jasper membership statistics report

Possibly write proc to update MH and MPH, which could be used as a daily

Debug to get same (or better) report outcome, and faster execution.

Determine how to phase in data change.

### Phase 2 Implement MP/MPH in servlet in parallel with CMD/MSD

DB update (immediate if faster enough, else thread)

Update MH on documented transitions (above), for memberships which have history built.

Update MPH as side-effect of insert/deletes on Membership\_passes

Rollout to all or selected orgs

Rollout new Jasper membership statistics report

### Phase 3 Conversion from CMD/MSD to MH/MPH

Jasper reports

Existing ANet reports using CMD or MSD (listed below)

Change all internal servlet usages of CMD or MSD

### Phase 4 Disable CMD/MSD

Eliminate CMD and MSD classes when all usages are removed

# Technical notes

### Usage of CMDs and MSDs in reports

Although CMDs and MSDs are somewhat similar, they are used for different groups of reports. Any changes to the current data must also study effects on these reports.

CMDs:

* Membership statistics report
* Membership summary report
* Membership retention report
* MembershipRenewalExpired report
* MembershipPass report

MSDs:

* Members vs. non-member statistics report
* Gender statistics report
* CustomQuestionAnswers report

### MH rebuild

Because change date operations change the dates of all existing CMDs and MSDs, they leave no useful information about the date history, so only transactions and current membership status will be used as a source for dates

* Insert data from transactions, including voids, to set: BEG\_DATESTAMP, BEG\_REASON, BEG\_TRANSACTION\_ID, Package\_id, PrimaryMemberCustomer\_id, DATE\_EXPIRES (also goes to end date)
* If last date\_expires doesn’t match membership, create one more MH for a date change
* Set DATE\_EFFECTIVE of all records to membership’s value
* Set END\_DATESTAMP, END\_TRANSACTION\_ID, END\_REASON to the beginning values from the next record, unless there is a gap.
* Compute SINCE\_DATE and Y\_MEMBERSHIP\_TYPE

### MPH rebuild

First, create MPH data for any existing membership/customer\_id in CMD table, if not already in MPH:

* Start = min(effective\_date)
* End = min(expiration\_date/termination date), excluding “null” dates

Second, create MPH data for any existing membership/customer\_id in MSD table, if not already in MPH:

* start = min(datestamp)
* end = max(expiration\_date)

Finally, create MPH data for any existing membership/customer\_id in the membership\_passes, if not already in MPH

* Start = min(datestamp) from transactions for this membership.
* End = “far future”

Cleanup out-of-range dates:

* If beg is “null”, set to min(datestamp) for transactions
* If end in the future or “null”, to the “far future”.