Delta Deployment

PassiveSocialEvents

1. Open ParsePassiveSocialEvents.sql
   1. Run create table scripts to create [dbo].[PassiveSocialEvents] table
   2. Run the insert scripts to parse action requests in [dbo].[ActionRequestLogs] table and insert the relevant records into [dbo].[PassiveSocialEvents] table

***I updated ParsePassiveSocialEvents procedure to filter out GetComments robot calls. Since the combination of ControllerName and ActionName from Chris’s notes made the below first 4constraints unnecessary, the singleLogId only applies to GetComments action, so I merged it into Feed/GetComments query constraint (see ParsePassiveSocialEvents.sql line 54)***

***~~and ActionName != 'RecentFeedItems'~~***

***~~and ActionName != 'PostComment'~~***

***~~and ActionName != 'PostCommentAsync'~~***

***~~and ActionName != 'PostFeedComment'~~***

***and (ActionName != 'GetComments' or ActionParameters like '%singleLogId%' and ActionParameters != 'singleLogId=[null]|||')***

1. Open CreatePassiveSocialEventLogTable.sql
   1. Run create table scripts to create [dbo].[PassiveSocialEventProcessLog]
   2. Run the rest scripts to create the CRUD procedures

* Note the purpose of [PassiveSocialEventProcessLog] table is for incrementally processing Azure storage tables.
  + To prevent ActionRequestLogEntity growing too big, the log process automatically creates a new storage table every month as ActionRequestLogEntity[Year][Month].
  + When a new storage table is created, a log record is inserted into [PassiveSocialEventProcessLog] table with only the storage table name populated.
  + When on-demand processing ActionRequestLogEntity is issued through “*Process Azure Table Storage*” link on Admin view under “Data Analytics Utils” section, the PassiveSocialEventUtilProc will pull the data from those unprocessed Azure storage tables and process the records in batches.
  + Each success batch processing will update the [ProcessedRecordCounts] field in [PassiveSocialEventProcessLog] table, so that if the process is interrupt, the next run can resume from where it was stopped.
  + When all records in one Azure storage table are processed, the [ProcessCompleted] bit in [PassiveSocialEventProcessLog] will be set to 1.
* This process is necessary when the latest passive social activities need to be included in the data visualization presentation.

1. Setup pre-calculated ActiveSocialEvents
   1. Open CreateActiveSocialEventsTable.sql and run the scripts to create [dbo].[ActiveSocialEvents] table and the cover index (it’s recommended to create the index before the large chunk of batch data being inserted, Azure database times out easily, if the localhost doesn’t have enough horsepower this also helps)
   2. Deploy [dbo].[GetActiveSocialEventProcessInfo].sql which is for incrementally processing visualization data and active social events
   3. Logon to OSBIDE and go to Admin view, run “Update Active Social Events” command to generate the pre-calculated visualization data with two-letter social event codes

* Note the visualization presentation can run in two different ways: pre-calculated and real-time. When the host machine has limited resources, it’s helpful to break down the large amount of data processing into separate steps.
* The “Update Active Social Events” process is to run the timeline visualization query in batches and put the query results, pre-calculated records, into [dbo].[ActiveSocialEvents] table
* The process can run incrementally, it goes by the order of Id column in [dbo].[EventLogs] table. When the process is interrupt, the next run will pick up from where it was left in the previous run.
  1. Run [dbo].[GetPrecalculatedStateMachineEvents].sql to deploy the stored procedure that pulls the timeline visualization data from the pre-calculated tables
  2. By default, the visualization is running in pre-calculated mode.

1. Alter [dbo].[GetStateMachineEvents] stored procedure to include passive social events in the timeline query.

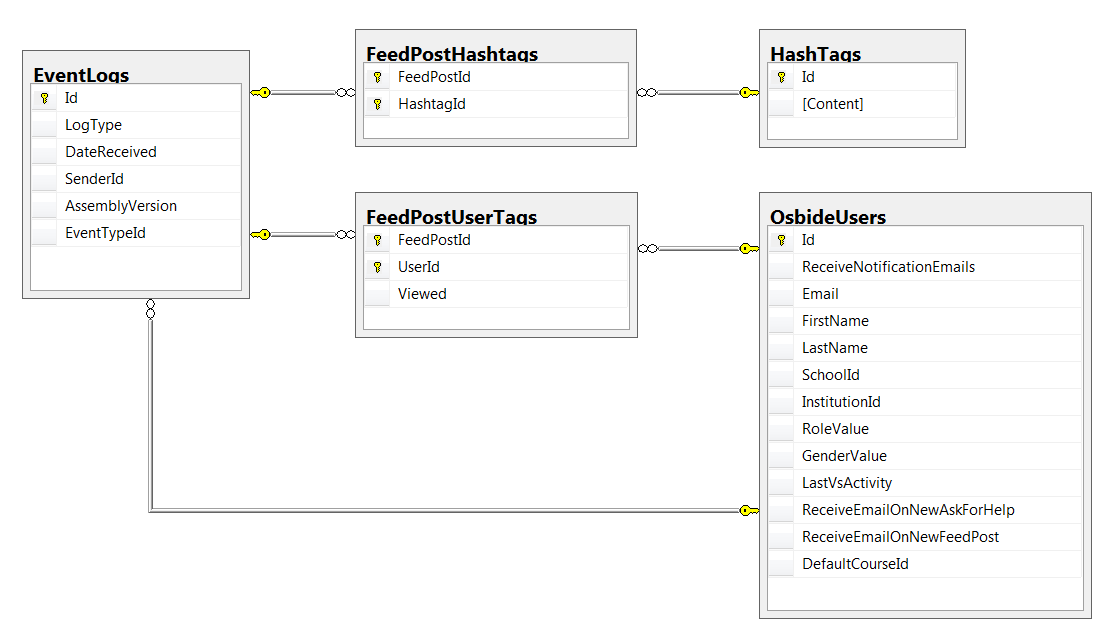
Export data file

A new GetCSV method is added in TimelineChartDataProc and the link above the timeline chart can trigger the action.



Enable HashTag and UserTag (mention) Features

1. Run CreateHashtagTables scripts to create tables
   1. HashTags
   2. FeedPostHashtags
   3. FeedPostUserTags tables
2. Deploy stored procedures
   1. [dbo].[InsertPostTags]
   2. [dbo].[GetHashtags]
   3. [dbo].[GetNotifications]
   4. [dbo].[GetTrends]



1. Switching to use EventTypeId instead of LogType string for improving query performance. Open AlterEventLogs.sql
   1. Add EventTypeId integer column to [dbo].[EventLogs] table
   2. Populate the new data column
   3. Create a new cover index for optimized activity feed query
   4. Add [dbo].[EventTypes] lookup table
   5. Re-index activity feed related columns

* Note the code areas, event log saving, filtering, querying, EventFactory, etc., have been modified to switch from LogType string to EventTypeId integer

1. Alter [dbo].[GetActivityFeeds] to include search filter and get rid of the dynamic SQL and the placeholders. With the compile-able query, SQL database may choose a better execution plan.

***The restored OSBIDE production database seems to be slow, which triggered some performance related modifications.***