## Repo design:

The database repo and python repo for this project are separate.

* The DB repo contains all the SQL scripts and an untracked, weekly DB backup for all projects and schemas. I removed non-relevant scripts for this upload. This repo mirrors the structure of the database itself with Schema prefixed in the file name.
* The AddeparRecon repo contains all other non-database scripts associated with the project.

## Project design:

All the data processing is performed by SQL. Python is used for:

1. Posting the job to and downloading the job from the Addepar Jobs API.
2. Executing the SQL procedures.
3. Logging

There are currently only two Addepar API Jobs. However, the database structure was designed to easily allow for further jobs to be added.

1. Accounts – cache updated account data into local database
2. Holdings – cache T-1 holdings into local database

Database design is to first import all records into a table in the dbimport schema with all field datatypes being varchar. After this, a post-import procedure is executed to scrub the data, convert it to the required datatype, and move it from the dbimport table to a table in the Addepar schema.

Since it can take 5-10 minutes for the Addepar API job to run, the Python portion of the project was designed to be stateless and all the data required for the job is stored in the database. For example, after the job is posted to the Addepar API, the Python app updates the job in the database, signifying that the job has been posted along with the Addepar API job ID.

## Project flow diagram:

1. Queue the jobs:
   1. EXEC Addepar.usp\_QueueJobs
   2. Will be executed by a SQL job once pushed to a server  
        
      ------------------------------------------------------------------------------------------------  
       EVERYTHING BELOW HERE IS LAUNCHED / MANAGED BY PYTHON  
      -----------------------------------------------------------------------------------------------
2. Queue the API Job – Python
3. Query job status until ‘percent\_complete’ = 1.000 – Python
4. Download the job to a JSON file – Python
5. Import the JSON data into SQL to a dbimport table
   1. EXEC dbimport.usp\_ImportAddeparAccounts @filePath='\_\_FilePath\_\_'
   2. EXEC dbimport.usp\_ImportAddeparHoldings @filePath='\_\_FilePath\_\_'
6. Execute the post-import proc to scrub the data and move it to the final, target table
   1. EXEC Addepar.usp\_AccountsPostImport
   2. EXEC Addepar.usp\_HoldingsPostImport @targetDate='\_\_TargetDate\_\_'
7. After each step of the job, python will run the below database procedure to update the job status in the database along with the data required to perform the next step in the job. This procedure will ensure the ModifiedDateTime and ModifiedByUserID fields are updated and will also ensure the details required for the next stage of the job are passed and updated in the JobQueue table.  
    EXEC Addepar.usp\_UpdateJobQueueStatus @JobQueueIdToUpdate=##, @JobDetails='\_\_required\_job\_detial\_\_', @UpdateToStatusId='\_\_update\_to\_status\_id\_\_'

## Remaining development:

1. Add infrastructure to import csv Trade / Allocation file from Flyer Co-Pilot (OMS)
   1. Still awaiting file from Flyer Co-Pilot – requires development work on their end
2. Reconciliation script and presentation – final presentation has not yet been decided. Possibilities include:
   1. Reconciliation data view queried into Excel – will likely start with this option
   2. Create a stored procedure that performs the reconciliation and caches the results into a table that can be queried later
   3. ^ and automatically email any reconciliation breaks to a defined distribution list.