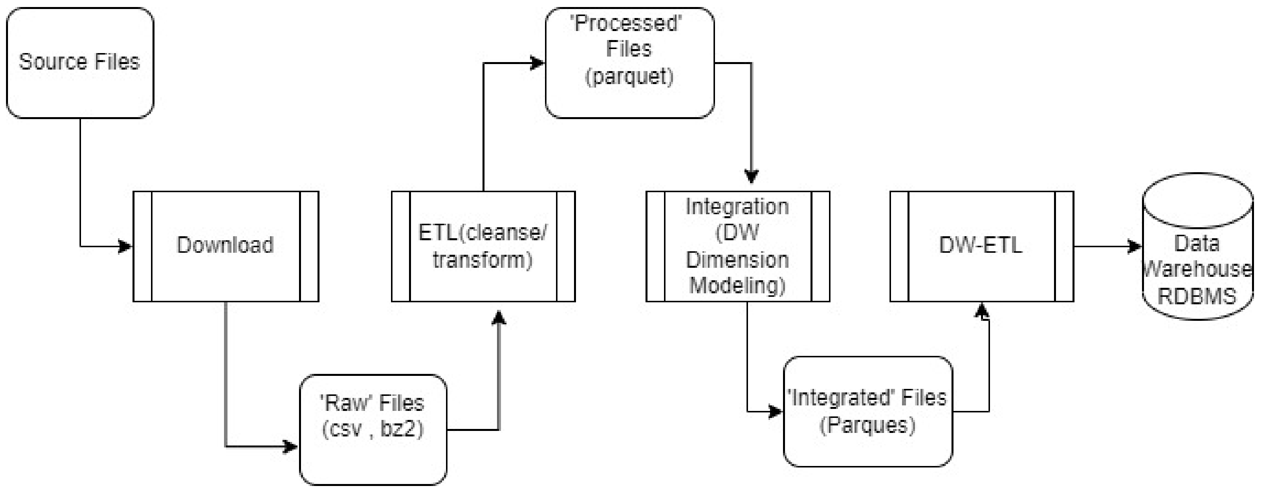
Airline On Time Analytic System (AOTA) Design and Implementation

1. Data flow chart and major processes



2. The major processes

The system include 4 major processes:

Process	Description
Download	Download data from source and save the original files to 'raw' layer. The file formats are csv or bz2.'
ETL	Read data from "raw" layer, conduct cleansing/transformation and save to the 'processed' layer. File format is parquet.
Integration	To get data from "processed'' layer, conduct data warehouse dimension modeling, i.e. create fact and dimension tables, and save to the 'integrated' layer, File format is parque.
DW-ETL (*) (to be done)	Get data from the "integrated" layer and load it to data warehouse RDBMS tables.

Here are the consideration of design and implementation for each process:

1) Download

The ETL process will download date from source and save to 'law' layer.

- The data comes from 4 source datasets:
    - **'Flight'** , which consists of 20+ data files, each for one year and sized a few hundred MB.
    - **'Airport', 'carrier', 'plane**' each consist of a single small file.

- For implementation,
    - We will implement a generic class **Downloader,** which gets a file from a **source url** and saves to the **target locatio**n.
    - We will implement a child class of Downloader, **downloaderOfFlightFlight**, which will download the datafile of one given year. This allows us to choose which year's data will be downloaded

2) ETL

The ETL process will read 'raw' data, conduct data cleansing and transformation, and save to 'processed' layer.

- Main considerations:
    - Similarly to the previous step, **'airport', 'carrier', 'plane'** will read data from a single raw file, and **'flight'** will read from multiple raw data files.
    - For 'airport', 'carrier', 'plane', 'read' and 'write' operations are similar but 'cleansing/transformation' may be different
    - We would want the processed 'flight' data to be saved to a single paquet file to feed the downstream integration process.

- Our solution:
    - We will implement a generic class, **'ETLProcessor**', which has methods including 'Extract', 'Transform' ,'Save', 'ETL_run' , to implement the corresponding functions..

- Each dataset has its own child class of ‘**ETLProcesor**’ to suit its needs.
    - . For ‘airport’, ‘carrier’, ‘plane’, the child class basically needs to override ‘transform’ method.
    - For ‘flight’, it will also override ‘read’ and a few other methods as the logic is quite complex.(handling multiple data files)

- For ‘flight’
    - We will save the data by partition of year using parquet. In this way, the multiple year data can be saved to a single file/location.
    - We will implement ETLProcessorOfFlight in a way that it can process one year of data at each time. This allows us to choose any year combination of data to process.

❖ Cleansing and transformation logic

Dataset	**Operations**
Flight	.Exclude ‘canceled’/’diverted’ flights .Add one monotonically_increasing_id column, named as ‘row_id1’ .Drop non-used columns: ● "ActualElapsedTime", "CRSElapsedTime", "TaxiIn", "TaxiOut", "Cancelled", "CancellationCode", "Diverted", etc
Airport	.Check the unique constraint on column "iata" .Add column 'airport_id' and assign "iata" to it
Plane	.Filter out the blank records
Carrier	.Nothing needs to be done

3) Integration

The integration process will read data from ‘processed’ layer, conduct dimension modeling related transformation to create dimension tables and fact table, save them ‘integration’ layer.

❖ Main thought

● This process will create one fact table dataset – **fact_flight** , and four dimension datasets – date, origin and destination, plane and carrier.

- ‘**fact_fligh**t’ will be created from the source flight dataset, which contains the details of the flight time information, and adding a few foreign key columns which will pointing the dimension datasets.

- For dimension datasets
  - **‘dim_carrier’** and **‘dim_plane’** can be created by directly copying the source dataset and adding some ‘id’ column if needed

  - **‘dim_date’** and ‘**dim_orig_dest’**, will be created from out of source flight dataset. Basically, they will be populated with the unique combination of relevant columns of flight data

❖ Implementation

There will be two steps

1. Create dim_carrier and dim_plane by basically just copying the cleansed source data.
2. Conduct transformation on the cleansed flight dataset, and then create fact_flight, dim_date and dim_orig_dest.

Here are more details:

Steps	Implementation
1. Create dim_plane dim_carrier	1. Create dim_plane and by copying ‘plane’ from ‘processed’ layer and adding column ‘plane_id’, populated from source column ‘tailnum’ 2. Create by copying ‘carrier’ from ‘processed’ layer and .add column ‘carrier_id’, populated from source column ‘Code’
2. Create Fact_flight, Dim_date, Dim_origin_d est	1. Create a staging table dataset by copy ‘flight’ dataset from ’processed’ layer, and adding two columns: "date_id” -- concatenation of source "Year", "Month", "DayofMonth" column "orig_dest_key" -- concatenation of source ""Dest", "Origin" column 2. Create ‘dim_date’ by extract the unique combination of the following columns out from the staging table: "date_id", "Year", "Month", "DayofMonth", "DayOfWeek" 3. Create ‘dim_orig_dest’ by: 1) Extracting the unique combination of the following columns out from the staging table created in step 1; "orig_dest_key", "origin", "dest", "distance" 2) Join dataset in step 1) with ‘airport’ to get airport, city, state, country information of origin and destination 4) Create ‘fact_flight’ by extracting the userful ‘flight’ columns and all the foreign key columns, from the staging table

	“date_id” “orig_dest_key" “plane_id" : renamed from source column ,"tailNum" "Carrier_id": renamed from source column "uniqueCarrier"

3. Class specification

1) Common utilities

Name	Description
ut_store	A placeholder for all sorts of parameters (such as folder names), that will be used by all the classes and processes.
ut_base	Some most used and common utilities.
ut_log	Utilities related to logging
ut_spark	Utilities related to Spark

2) Classes for Download Process:

Class Name	Description
Downloader	A utility to download a file from source url to a target location. **Attributes**: ● source url ● target_file name ● target_folde
downloaderOfFlight	A utility to downloads one flight dataset of a given year. **Attribute:** ● year
DownloadManager	The process to run the downloaders to download the flight dataset and other datasets

3) Classes for ETL process

Class Name	Description
ETLProcessor	A generic class to run ETL (extract, transform/clean, load)

	process on one dataset. It will read data from a 'raw' file, perform cleansing/transformation and save to 'processed' folder. **Attributes:** ● entity_name ● raw_file_name ● cln_file_name ● file_schema (for read)
ETLProcessorOfCarrier ETLProcessorOfAirport ETLProcessorOfPlane	Child classes of ETLProcessor, which process dataset 'carrier', 'airport', 'plane'. They inherit everything from ETLProcessor except the method 'Transform'. **Attributes**: attributes inherited from ETLProcessor
ETLProcessorOfFlight	Child class of ETLProcessor that processes 'flight'. As flight dataset consists of many datafiles, this class overrides a few methods to suit the need. **Attributes:** ● attributes inherit all ETLProcessor's ● year
ETLManager	A module to organize and run ETL process of each dataset.

4) Classes for IIntegration process:

Class name	**Description**
intgrFactTbGeneration	A module to create fact_flight, dim_date, and dim_orig_dest
integrationManager	A module to execute the integration process. It will: 1. Create dim_plane, dim_carrier 2. Call intgrFactTbGeneration to create fact_flight,dim_date, and dim_orig_dest