**OneThree Prompt**

1. The database schema I would use involves three entities below:
   1. **Drug**: I have chosen to use the DrugBank ID as the PK with the Smiles string being included as a string value. If we would like to include more unique drug information that is possible to do so as well with this given structure while keeping the DrugBank ID’s unique.
   2. **Identifier**: I chose an unpivoted table structure with an autoincremented ID and a foreign key to the Drug table. The relationship between Drug and Identifier is a *one to many relationship*. The reason why I have chosen this table structure is due to the everchanging nature of the external identifiers, with sites being added and updated we want an easy way to remove new and old sources, I wanted to include a way for us to refresh and update external identifiers and deprecated identifiers as well.
   3. **Target**: I chose an unpivoted table structure with an autoincremented ID and a foreign key to the Drug table. The reason I chose this was to account for any changes whether added or subtracted with targets either being added or removed, we also could potentially have an unknown number of targets to account for. In addition, for every target there is an unknown number of a combination of actions that could coincide together, therefore, it was better functional as bit values to flip on and off. The relationship between the drug table to this table would be a *one to many relationship*.

For this problem when thinking about ways to refine the structure, it would be important to consider creating a dimension table for the external identifier names to minimize data redundancy. The same can be considered for target id’s as well as actions in a more verbose, normalized solution.

1. When building ingestion tools, we need to make sure we have best practices both in our cloud architecture, as well as our data pipelines.
   1. Reducing Complexity –
      1. identifying transformations required for each data source and reducing complexity both in job orchestration for pipelines, as well as transformation code,
      2. some transformations need to be reusable and naming conventions should be applied.
      3. Just like the database schema above, making entities simple to identify with proper data modeling in check
   2. Reliability –
      1. Have a well-defined source to target mapping
      2. Have proper measurement metrics for ingestion and transformation
      3. Through queries/cloud configurations/auto-scaling make sure there is proper load balancing
      4. In line validation for process + functional checks for data, data quality checks must be present
      5. Logging, automated system responses and reports as well as dependency failure must be implemented
   3. Scalability/Flexibility –
      1. Point mentioned above about necessary load balancing/ auto scaling configurations enabled
      2. When considering data storage and cost, we have to think about if it would make sense to store into a data lake, then transform into a data warehouse
      3. Expensive transformations/iteration through data must be removed/minimized
         1. Any time we are iterating through a dataframe/performing expensive transformations we must understand how it will scale
      4. Data Model must be kept general and high-level enough to account for different sources, same can be said for transformations.
      5. Triggering/batch processing/upserting data must be planned accordingly if going with a relational model

In conclusion, there are many steps that can be taken to optimize each of our ingestion scripts to collaborate/share transformations and processing with other scripts as well. A proper data model, and design document is necessary for all this, and I hope at this time the information I provided is a step in the right direction.