

Customized ETL (Extract, Transform, Load) System

Objective:

This system was developed for a large Bank in South Africa. They have a heterogeneous data storage environment. The core banking system runs on Mainframe and produces flat files as output on a daily basis. The Bank has various other systems including a SAP system where the data has to be reconciled. The Bank has similar systems to handle the loan products as well. The requirement was for a system that can take the data from one system and transform it to produce the output for any other system. The system also should have the capability to connect to any storage device and transform the data to other formats (Example: pull the data from flat files and pump it to a RDBMS database).

Process followed:

To develop this ETL tool the design had to be very flexible and the architecture had to be very generic, so that it could adapt to any system. Keeping this purpose in mind the core component was built. This component could take any data and transform it to another format. This engine could be customized with a few configuration XML files.

Initially this tool was created with a few basic functionalities; like reading the data from a flat file and populates a SQL Server database. Later on other capabilities were added to it like the ability to recognize the type of input file format and incorporate business logic to do transformation to the data before pushing it to the destination system. At a later stage APIs were designed around it and it could be integrated into other applications to build ETL capabilities into them.

Key Benefits

- ✓ The bank could get the data from heterogeneous systems and push it into a homogeneous RDBMS system which could be used for MIS purpose.
- ✓ These tools helped the bank to reduce the complexities around the various systems that were being used earlier to transform the data.
- ✓ It could be integrated with the End Of Day (EOD) process to produce the desired output without any human intervention.
- ✓ The lightweight version of the API could be integrated with any application that was required to interact with any data source.
- ✓ This helped reduce the application development time for other banking applications as well.

Technologies Used

- ✓ Java was used as the platform for building this tool.
- ✓ XML was used to define the source and destination systems.
- ✓ Multithreading was used extensively to help the system process parallel requests. This improved the system performance.
- ✓ Java Reflection was used to build the generic capability into the system.