**Use Case #1: Refunds**

We have transaction data flowing in from a source. The types of transactions would be either a purchase, a refund or a void transaction. Both refund transactions and the void transactions would have a parent transaction. Considering that we have millions of transactions flowing in from the source; we need to find the total refunds per customer each month.

When finding total refunds:

* Any transaction which has any of its child transactions as a void transaction should be ignored.
* Any refund transaction for which the parent transaction happened in the same month also should be ignored.

**Assumption:**

1. Source data is generated and hosted externally. They can either be pulled via API interface or push to specified target location
2. Source data is available in 3 extracts:

**Transaction**:

|  |  |  |  |
| --- | --- | --- | --- |
| Layout | Description | Type | Example |
| transaction\_id | Unique id for each transaction | uniqueidentifier | Eg. 6F9619FF-8B86-D011-B42D-00C04FC964FF |
| parent\_id | Parent Transaction id | uniqueidentifier | Eg. 6F9619FF-8B86-D011-B42D-00C04FC964FF |
| type | Transaction type | string | Ie. purchase,refund,void |
| subscription\_id | Unique id for subscriber | uniqueidentifier | Eg. e7700192-ef8f-4fb0-aa4e-938d7464305b |
| transaction\_timestamp | Date and Time transaction occurs | Timestamp with timezone | Eg. 2021-10-05T12-5218.642Z |
| amount | Total amount | decimal | Eg.2800.15 |
| currency | currency code | string | Eg. AU |

**Subscription**:

|  |  |  |  |
| --- | --- | --- | --- |
| Layout | Description | Type | Example |
| subscription\_id | Unique id for subscriber | uniqueidentifier | Eg. e7700192-ef8f-4fb0-aa4e-938d7464305b |
| customer\_id | Unique id for customer | uniqueidentifier | Eg. e42fcdb5-d069-4955-8935-3c2faf344673 |
| product\_name | Name of the product | string | Eg. business trade |

**Customer**:

|  |  |  |  |
| --- | --- | --- | --- |
| Layout | Description | Type | Example |
| customer\_id | Unique id for customer | uniqueidentifier | Eg. e7700192-ef8f-4fb0-aa4e-938d7464305b |
| firstname | Customer first name | string |  |
| lastname | Customer last name | string |  |
| email | Email address | string |  |

1. Source data is available in batch on a daily basis.
2. AWS service to be provided such as s3 as storage, lambda as serverless computing service.

ETL is available to use such as Talend.

Cloud datawarehouse is available such as redshift or snowflake.

**High Level of Solution Diagram:**

1. To use lambda function call external API to get the 3 source data extracts in 3 individual files and stored them in one s3 bucket with separate subfolders.

For transaction subfolder, two additional subfolders to be created for Year and Month. Output file is in csv format with header. Filename to be ExtractName\_TimeStampwithTimeZone.csv.

(ie. Transaction\_2021-10-05T12-5218.642Z.csv,

Subscription\_2021-10-05T12-5218.642Z.csv,

Customer\_2021-10-05T12-5218.642Z.csv)

Extracts are persisted in the s3

1. To use Talend job to read files in s3 with key prefix to get newly arrival objects and load them into raw layer in the data warehouse.

There would be three individual tables being appended with the new records in the files to be loaded.

(ie. Raw.Transaction, Raw.Subscription,Raw.Customer)

Raw files name and loaded timestamp is appended in each row.

**Raw.Transaction**:

|  |  |
| --- | --- |
| Column | Type |
| transaction\_id | uniqueidentifier |
| parent\_id | string |
| type | string |
| subscription\_id | string |
| transaction\_timestamp | timestamp |
| amount | Decimal(38,2) |
| currency | string |
| cdp\_rawfilename | string |
| cdp\_loadtime | timestamp |

**Raw.Subscription:**

|  |  |
| --- | --- |
| Column | Type |
| subscription\_id | uniqueidentifier |
| customer\_id | string |
| product\_name | string |
| cdp\_rawfilename | string |
| cdp\_loadtime | timestamp |

**Raw.Customer:**

|  |  |
| --- | --- |
| Column | Type |
| customer\_id | uniqueidentifier |
| firstname | string |
| lastname | string |
| email | string |
| cdp\_rawfilename | string |
| cdp\_loadtime | timestamp |

1. To use Talend to call procedures in the data warehouse :
2. Source data from Raw.Transaction with transaction type of purchase and loaded into DWH.STAGE.Transaction\_Refund\_Incremental

**DWH.STAGE.Transaction\_Refund\_Incremental:**

|  |  |
| --- | --- |
| Column | Type |
| transaction\_id | uniqueidentifier |
| parent\_id | string |
| type | string |
| subscription\_id | string |
| transaction\_timestamp | timestamp |
| amount | Decimal(38,2) |
| currency | string |
| cdp\_rawfilename | string |
| cdp\_loadtime | timestamp |

1. Source data from Raw.Transaction with transaction type of void and loaded into DWH.STAGE.Transaction\_Void\_Incremental

**DWH.STAGE.Transaction\_Void\_Incremental:**

|  |  |
| --- | --- |
| Column | Type |
| transaction\_id | uniqueidentifier |
| parent\_id | string |
| type | string |
| subscription\_id | string |
| transaction\_timestamp | timestamp |
| amount | Decimal(38,2) |
| currency | string |
| cdp\_rawfilename | string |
| cdp\_loadtime | timestamp |

1. Source data from Raw.Transaction with transaction id and parent id and build DWH.Dim.Transaction\_Hierarchy

**DWH.Dim.Transaction\_Hierarchy:**

|  |  |
| --- | --- |
| Column | Type |
| transaction\_id(PK) | uniqueidentifier |
| parent\_id | string |
| parent\_status | string |
| parent\_transactionmonth | date |
| Child\_id | string |
| Child\_status | string |
| Effective\_start\_date | timestamp |
| Effective\_end\_date | timestamp |
| current\_version | boolean |

1. Upsert from Raw.Subscription and Raw.Customer into DWH.Dim.CustomerDimension

**DWH.Dim.CustomerDimension:**

|  |  |
| --- | --- |
| Column | Type |
| subscription\_id(PK) | uniqueidentifier |
| customer\_id | string |
| firstname | string |
| lastname | string |
| email | string |
| Effective\_start\_date | timestamp |
| Effective\_end\_date | timestamp |
| current\_version | boolean |

1. To use Talend to call procedures in the data warehouse :
2. Append DWH.STAGE.Transaction\_Refund\_Incremental into DWH.Fact.Transaction\_Refund
3. Update deletion flag in DWH.Fact.Transaction\_Refund by joining parent\_id of DWH.STAGE.Transaction\_Void\_Incremental to transaction id in fact table.
4. Update deletion flag in DWH.Fact.Transaction\_Refund by joining transaction id onto DWH.Dim.Transaction\_Hierarchy where parent\_transactionmonth is the same as transaction month in fact table.

**DWH.Fact.Transaction\_Refund:**

|  |  |
| --- | --- |
| Column | Type |
| transaction\_id | uniqueidentifier |
| parent\_id | string |
| type | string |
| subscription\_id | string |
| transaction\_month | date |
| amount | Decimal(38,2) |
| currency | string |
| deletion\_flag | boolean |
| cdp\_rawfilename | string |
| cdp\_loadtime | timestamp |

1. To use Talend to call procedures in the data warehouse:

Build DWH.View.Refund\_Dashboard with deletion\_flag is not ‘Y’

**DWH.View.Refund\_Dashboard:**

|  |  |
| --- | --- |
| Column | Type |
| transaction\_id | uniqueidentifier |
| Customer.id | string |
| refund\_month | date |
| amount | Decimal(38,2) |

**Orchestration:**

1. Schedule lambda function with EventBridge to run once a day to ensure complete source data is extracted.
2. In TAC, schedule talend job for the data pipeline and to be triggerd on a daily basis.

**Automated Testing:**

Pytest for lambda function and Talend CI framework for talend job.

**CI/CD:**

GitHub Action for lambda function deployment and Talend job deploy to TAC.

**Monitoring:**

Logging for lambda function and talend job. Upon talend job completes successfully, send success notification to support inbox.