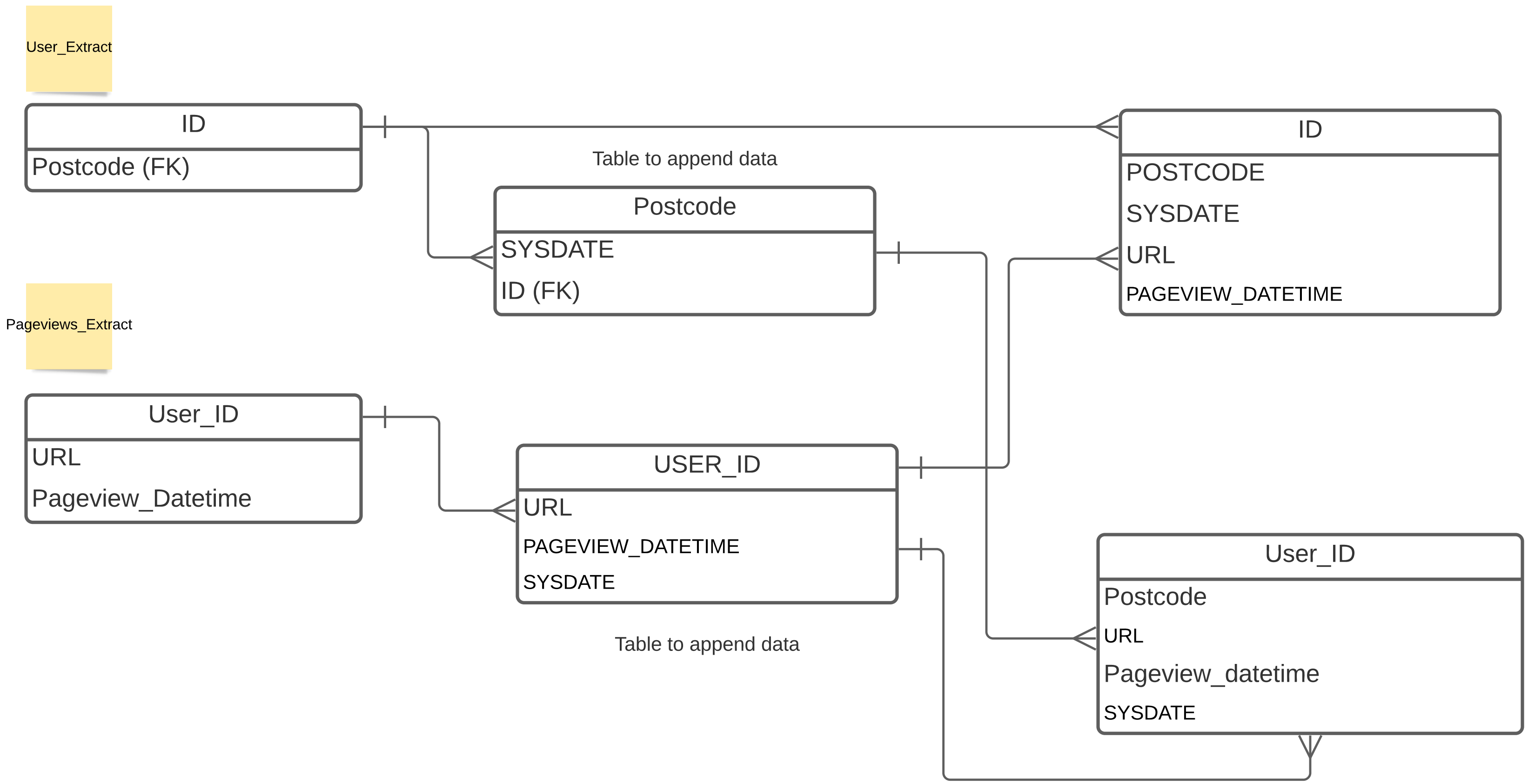
Logical Model



User_Extract

ID

Postcode (FK)

Pageviews_Extract

User_ID

URL

Pageview_Datetime

Table to append data

Postcode

SYSDATE

ID (FK)

USER_ID

URL

PAGEVIEW_DATETIME

SYSDATE

Table to append data

ID

POSTCODE

SYSDATE

URL

PAGEVIEW_DATETIME

User_ID

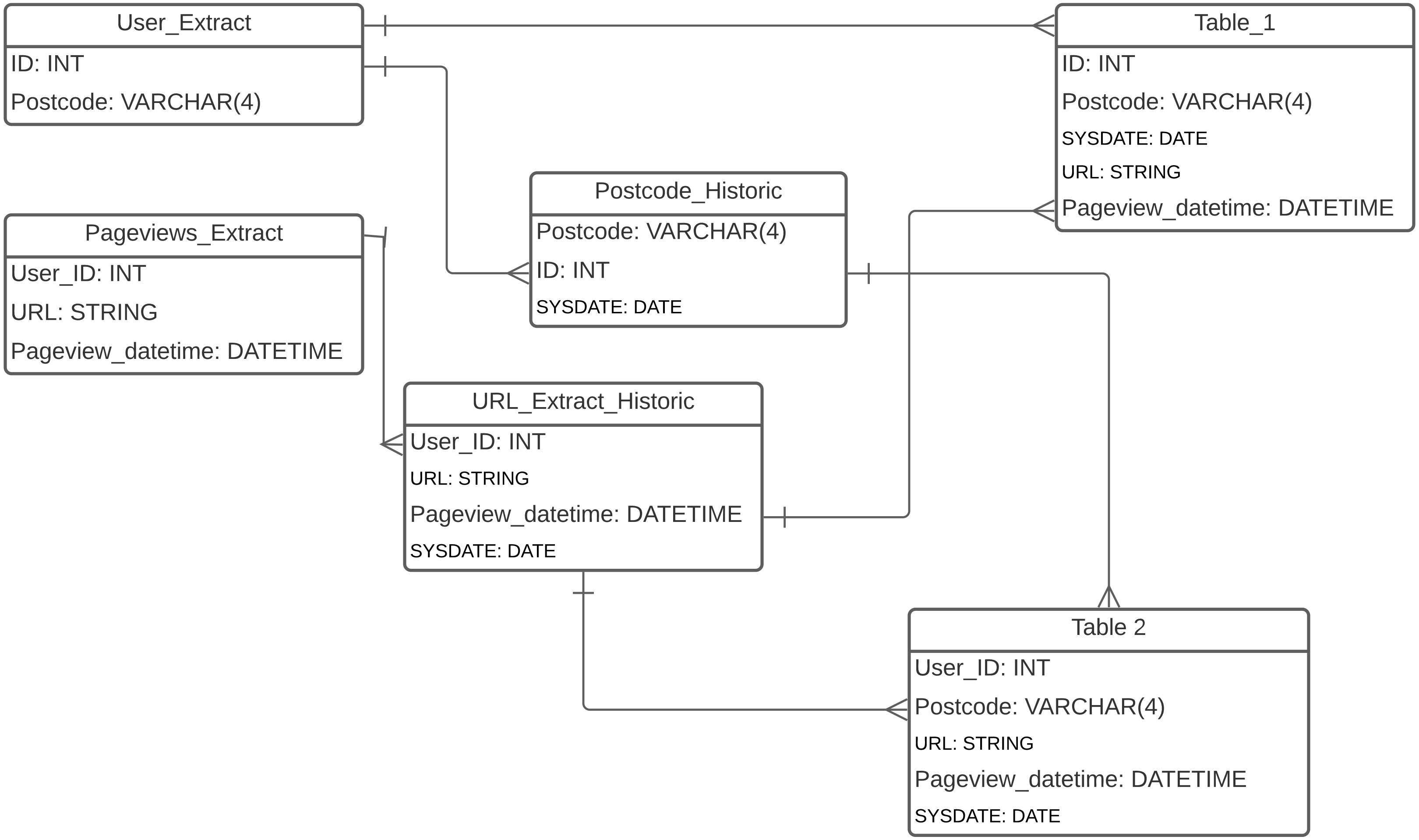
Postcode

URL

Pageview_datetime

SYSDATE

Physical Model



User_Extract
- ID: INT
- Postcode: VARCHAR(4)

Table_1
- ID: INT
- Postcode: VARCHAR(4)
- SYSDATE: DATE
- URL: STRING
- Pageview_datetime: DATETIME

Postcode_Historic
- Postcode: VARCHAR(4)
- ID: INT
- SYSDATE: DATE

Pageviews_Extract
- User_ID: INT
- URL: STRING
- Pageview_datetime: DATETIME

URL_Extract_Historic
- User_ID: INT
- URL: STRING
- Pageview_datetime: DATETIME
- SYSDATE: DATE

Table 2
- User_ID: INT
- Postcode: VARCHAR(4)
- URL: STRING
- Pageview_datetime: DATETIME
- SYSDATE: DATE

Documentation for Pipeline

Key considerations for running Pipeline -

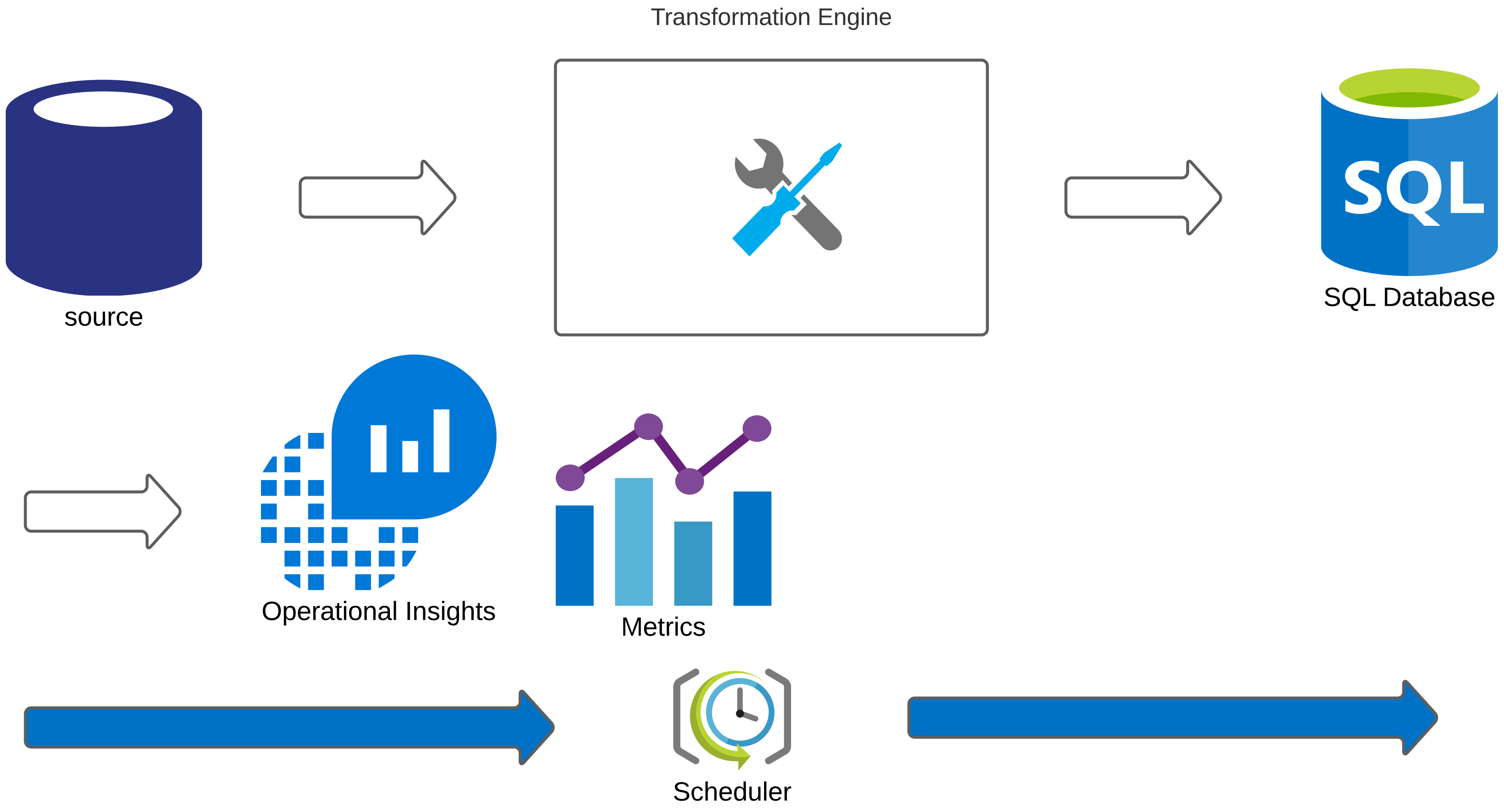
1) 'user extract' data to be loaded in 'Postcode'_Historic' table as an append . It should contain a reference to the load date, this should be the system time/date. This will help to determine the historical records of the users locations on a particular day. Updated Daily - Midnight

2) 'Pageviews_extract' is appended into 'Url__Historic' each hour. This table should be updated hourly and should contain a dependency to run after the 'Pageviews_extract' has been updated each hour. Use of a trigger would be useful. The table will append data each hour to determine historical records.

3) Table_1 is using data from 'User_extract' to determine the latest postcode position and data from 'URL_Extract_Historic' to allow for a view on a given time period since it will contain historic data rather than the latest hourly data. Table is updated at midnight.

4) Table_2 is using data from 'URL_Extract_Historic' & ' Postcode_Historic' tables to determine the position of the user at the time the pageview was made.

Pipieline Summary & Proposition



Proposed Mechanism for scheduling

Option 1: Airflow & Python
Benefits: Workflow Management, automation, Task Dependency, Monitoring and alerts, DAGs, Community
Disadvantages: Not intuitive for new users, No versioning of your data pipelines, Sharing data between tasks is limited

Option 2: Panoply
Benefits: Scalability, Simplicity, Multipurpose Datastack, allows quick building/setup, automated data ingestion, Agile Modeling and Learning Algorithms, Smart Data Infrastructure (Use Case, Query and Server Optimization)
Disadvantages: Basic UI, Can be slow, limited ELT connectors, No alert for job fails

Option 3: Talend
Benefits: User-friendly interface, Numerous connectors, Continuous integration reduces overhead of repository management and deployment
Disadvantages: Basic scheduling, Limited community, Struggles to transform millions of row of data, Requires Java developers for complex backgrounds

Option 3: Xplenty
Benefits: No code / Low code platform, Quickly build complex data pipelines, Intuitive interface, Code uses various languages, maintains a large library of ready-made integration, preload transformations, Easy to use work flow sequence
Disadvantages: Difficult to debug errors, Limited 3rd party connectors, Limited support

Option 5 : Stitch
Benefits: No API maintenance, scripting, cron jobs or JSON, Quick connections between first-party data sources, Self-service data ingestion,
Disadvantages: Fewer available options for data extraction, warehousing and loading, No support of SQL Server and Azure, Not designed for complexity , Not intuitive for new users, limited preloaded transformations

Verdict:

Option 1 provides the strongest end to end solutions for ETL management with the least significant drawbacks

Other Notable Options -

Informatica
Skyvia
Fivetran