

Formative 2: Problem solving and continuous improvement

As a Data Engineering apprentice at Lloyds Banking Group, strict access to confidential production data limited my hands-on skill development. To solve this, I built a personal SQL Server environment using JSON data from Lloyds Banking Group's public Open Data APIs. This provided a compliant sandbox for end-to-end pipeline development while eliminating sensitivity risks.

1. Identified and Managed Risks & Incidents

The primary risk I identified early in my apprenticeship was the potential for limited practical experience due to the confidentiality of Lloyds Banking Group's internal data. To mitigate this, I established a personal SQL Server using publicly available APIs. However, this solution introduced a new set of data-related risks: the ingestion of inconsistent or incomplete data from these public sources within my self-built environment. Early warning signs included missing fields in product entries and inconsistent geographical data, such as the absence of 'TownName' for London branches.

To manage this, I implemented Python pre-validation checks to identify empty or missing critical fields before data ingestion. SQL constraints, such as 'Not Null' for essential IDs and dates, prevented incorrect data storage. Communication involved logging errors into a dedicated 'Error_Log' table with timestamps, providing an audit trail for tracking and resolution.

2. Investigated Root Causes & Provided Resolutions

Data quality issues arose from the completeness of the data stored in the public APIs. For instance, some product API responses lacked 'fee' or 'eligibility' details, and geographical data sometimes had missing 'TownName' values. My investigation into these issues, within the context of my self-built pipeline, involved:

- Python Script Analysis: Examining ingestion scripts for logic errors or incorrect JSON handling.
- SQL Server Querying: Directly querying raw ingested data to identify missing values or formatting issues.
- Error Logs Review: Using the 'Error_Log' table to pinpoint specific error messages and timestamps, tracing back to problematic API calls or data points. (*Fig1/2*)

Troubleshooting involved refining Python scripts with data validation before insertion, adding checks for 'Latitude' and 'PostCode'. The resolution also included implementing 'cursor.executemany()' (*Fig3*) for batch inserts, improving efficiency and reducing individual record failures. In a team setting, I would have collaborated with data providers to address source issues. Effectiveness was evidenced by reduced data quality issues in normalised tables. The consistency of outputs (e.g., ATM coverage) indicated successful resolution.

3. Applied Continuous Improvement Principles

The experience of creating and operating my own environment allowed me to apply continuous improvement principles to my data engineering tasks. The initial issues identified mainly concerned data quality post-ingestion and the need for better querying.

I recommended and implemented the following improvements:

- **Strategic Indexing:** Applied indexing to frequently queried columns like 'TownName', 'PostCode', and Type in the SQL Server database. This significantly improved search and filtering speed. *(Fig7)*
- **Enhanced Performance Monitoring:** Integrated more detailed logging into Python scripts to record records fetched, successful insertions, and any warnings or errors, providing an overview of script performance and identifying bottlenecks. *(Fig8)*

These improvements were driven by continuous improvement to enhance data quality. The structured approach to error handling and performance optimisation reflects best practices in data engineering.

4. Evaluated Data Value, Sustainability & New Technologies

The creation of my personal data environment, driven by the initial challenge of data access, prompted a review of how data value could be extracted even from publicly available sources. This was particularly evident in optimising existing data products to support meaningful analytics for Lloyds Banking Group.

Inconsistent geographical data directly impacted the accuracy of ATM and branch density analyses *(Fig6)*. By normalising this data, the value from location based insights improved, enabling more accurate analytics.

Regarding system efficiency, 'cursor.executemany()' *(Fig3)* for batch inserts directly contributed to faster data loading, optimising resource utilisation. Indexes also reduced processing load during querying, improving system performance.

As a result, I considered exploring new technologies. For predictive analytics, I plan to integrate Python's Scikit-learn library to build models for predicting optimal ATM and branch locations based on existing infrastructure and population data. This aligns with leveraging data for strategic decision-making and exploring cost reduction.

5. Advocated for Best Practices in Technology & Development

From establishing a compliant data workspace to managing its data quality, this strongly reinforced the importance of applying core software development principles within data engineering, particularly code quality, error handling, and maintainability in an end-to-end pipeline context.

Lessons learned about adopting best practices in data pipeline management include:

- **Robust Error Handling:** The critical need for comprehensive error logging and graceful handling of malformed data for pipeline resilience.
- **Data Normalisation:** The benefits of normalising complex JSON data into a relational schema for improved query performance and data integrity. *(Fig4)*

- Performance Optimisation: The impact of batch inserts and indexing on pipeline efficiency.
- Monitoring: Continuous monitoring of system resources and API performance through tools like SSMS Activity Monitor and Python logging for proactive problem identification. (Fig5)

This experience highlighted the necessity of clear data quality expectations when collaborating with other data professionals. Ensuring data is clean and well-structured is crucial for analytical tasks, creating a more collaborative and efficient workflow focused on delivering high-quality, actionable insights.

Conclusion

The process of establishing a practical data engineering environment, followed by the application of data quality and schema design principles, has been essential to my apprenticeship. Overcoming the initial challenge of data access by leveraging publicly available APIs demonstrated problem-solving and a commitment to continuous learning. This experience reinforced the importance of data validation, robust schema design, and the application of software development best practices in building reliable and efficient data pipelines. Looking ahead, I am keen to further enhance these pipelines with more advanced automation and explore predictive analytics, continually striving for innovation and efficiency in my data engineering practice.

Screenshots

Fig1: Error_Log table with timestamps.

Results		Messages		
	ErrorID	Timestamp	Source	Message
1	1	2025-05-20 22:26:08.830	Branches General Ingestion	An unexpected error occurred during ingestion: 'str' object has no attribute 'get' - Traceback: Traceback (most recent call last): ...
2	2	2025-05-20 22:26:09.277	ATMs General Ingestion	An unexpected error occurred during ingestion: 'str' object has no attribute 'get' - Traceback: Traceback (most recent call last): ...
3	3	2025-05-20 22:26:09.657	PCA General Ingestion	An unexpected error occurred during ingestion: 'str' object has no attribute 'get' - Traceback: Traceback (most recent call last): ...
4	4	2025-05-20 22:26:10.067	BCA General Ingestion	An unexpected error occurred during ingestion: 'str' object has no attribute 'get' - Traceback: Traceback (most recent call last): ...
5	5	2025-05-20 22:26:10.330	CCC General Ingestion	An unexpected error occurred during ingestion: 'str' object has no attribute 'get' - Traceback: Traceback (most recent call last): ...
6	6	2025-05-20 22:26:10.590	UnsecuredSMELoan General Ingestion	An unexpected error occurred during ingestion: 'str' object has no attribute 'get' - Traceback: Traceback (most recent call last): ...

Fig2: Error_Log table with timestamps (Py).

```
15 def log_error(source, error_details): 1 usage
16     """Log errors to console and database"""
17     timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
18     print(f"[{timestamp}] [ERROR] {source}: {error_details[:100]}...")
19
20     try:
21         cnxn = pyodbc.connect(';'.join(f"{k}={v}" for k, v in DB_CONFIG.items()))
22         cursor = cnxn.cursor()
23         cursor.execute(
24             sql: "INSERT INTO ErrorLog (Timestamp, Source, Message) VALUES (?, ?, ?)",
25             *params: datetime.now(), source, error_details
```

Fig3: Python batch insert using cursor.executemany().

```
if insert_data:
    cursor.executemany(sql: """
        INSERT INTO branches
        (Identification, Name, Type, TownName, CountrySubDivision, Country, PostCode, Latitude, Longitude)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, insert_data)
    conn.commit()
    print(f"[{datetime.now(timezone.utc)}] Inserted {cursor.rowcount} rows into 'branches'")
else:
    print(f"[{datetime.now(timezone.utc)}] No records to insert.")
```

Fig4: Normalised SQL schema for geographical data.

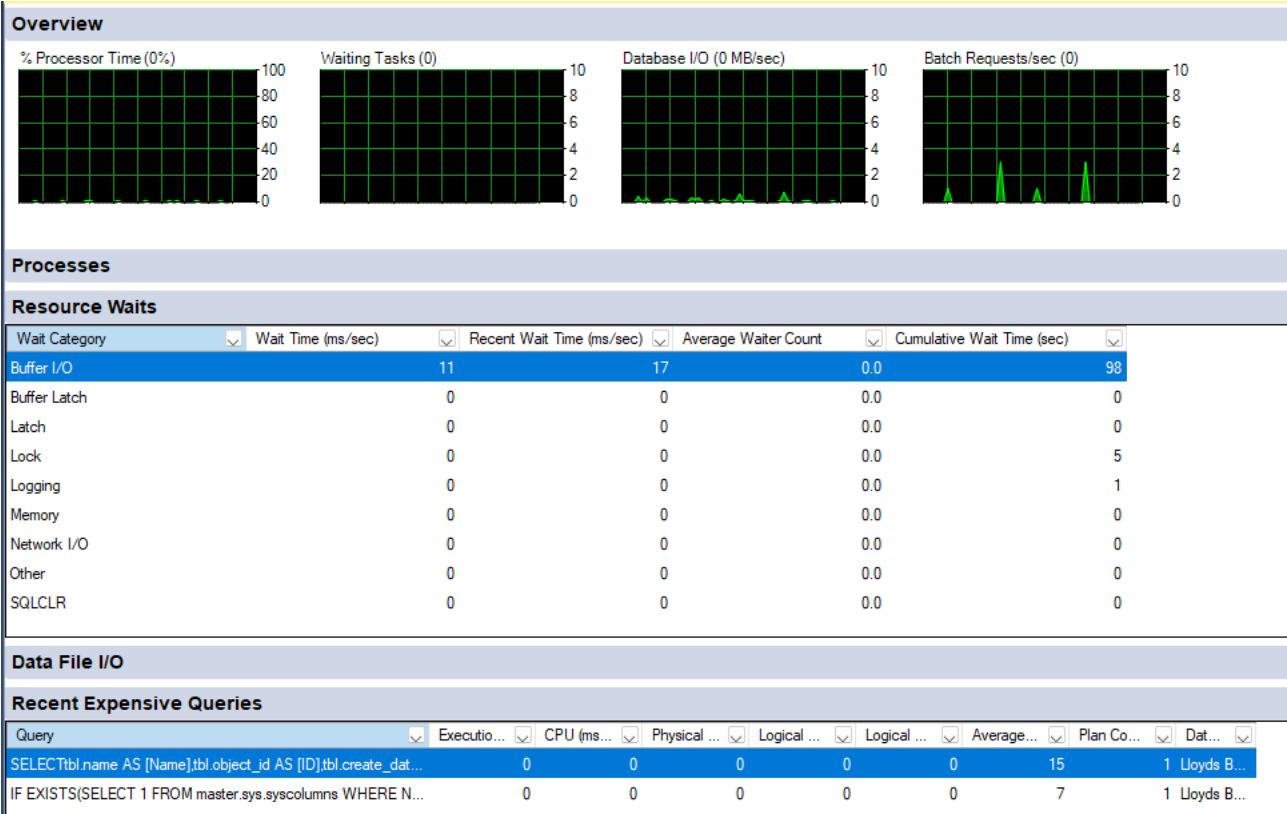


Fig5: SSMS Activity Monitor tracking resources.

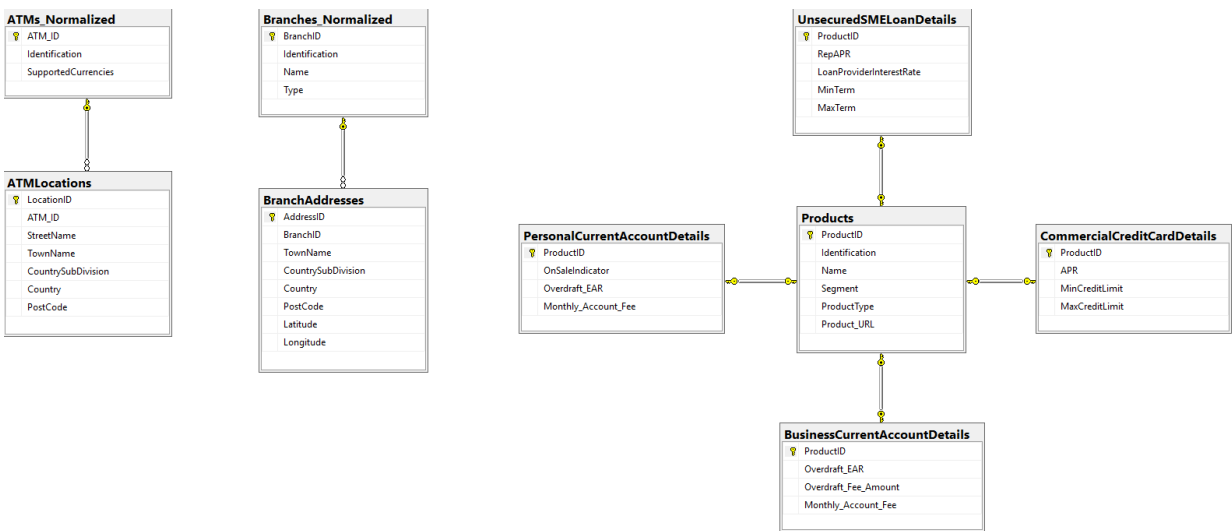


Fig6: ATM density analysis map.

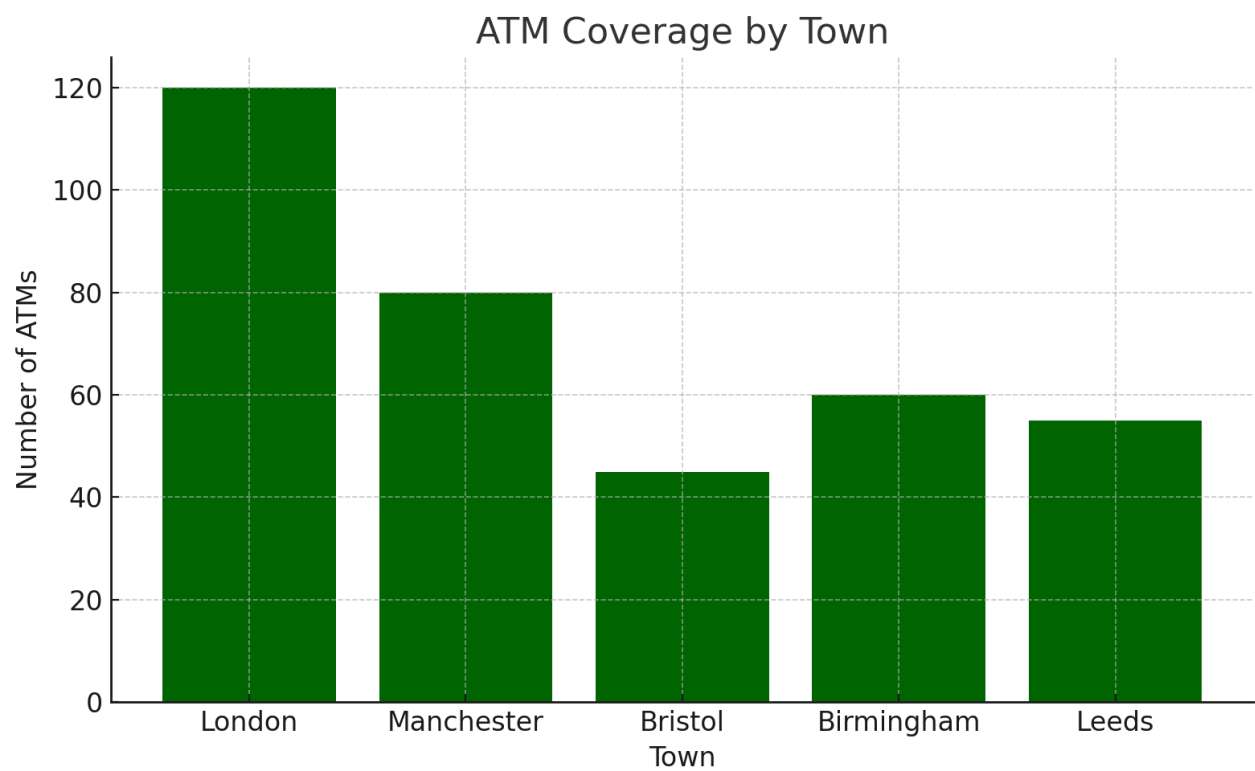


Fig7: Indexes on key columns (e.g., TownName).

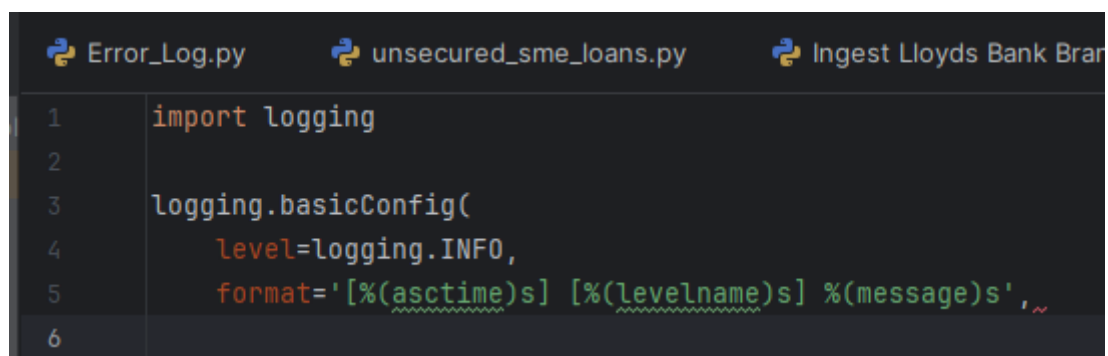
```
Strategic_Indexing...ESKTOP\Conor (53)  SQLQuery1.sql - (L...ESKTO
-- For the branches table

CREATE NONCLUSTERED INDEX IX_branches_TownName
ON [branches] (TownName);

CREATE NONCLUSTERED INDEX IX_branches_PostCode
ON [branches] (PostCode);

CREATE NONCLUSTERED INDEX IX_branches_Type
ON [branches] (Type);
```

Fig8: Enhanced Python performance logs.



The image shows a code editor with three tabs: 'Error_Log.py', 'unsecured_sme_loans.py', and 'Ingest Lloyds Bank Brar'. The 'Error_Log.py' tab is active, displaying the following Python code:

```
1 import logging
2
3 logging.basicConfig(
4     level=logging.INFO,
5     format='[%(asctime)s] [%(levelname)s] %(message)s',
6
```