PayPal: Transactional Data

Dhruvin Shah Illinois Institute of Technology

ITMD 526: Data Warehousing

Professor Raj Krishnan

March 14, 2025

A blue and white logo

AI-generated content may be incorrect.

Author Note

Under the direction of Professor Raj Krishnan, the Illinois Institute of Technology's Information Technology & Management Department provided the research and foundation for this publication. Dhruvin Shah, who works for the College of Computing's Information Technology & Management Department and specializes in Management Information Systems, can be reached at 35th St., Chicago, IL 60616, if you have any questions or concerns about this article.

Contact: dshah104@hawk.iit.edu

Index

Introduction ….………………………………………………………………………………………………………….

Data Sources…………………………………………………………………………………………………………………………….

Normalized Database…………………………………………………………………………………………………………………

ETL Implementation……………………………………………………………………………………………………………………

Dimensional Model…………………………………………………………………………………………………………………….

Analytical Queries………………………………………………………………………………………………………………………

Report Summary …………………………………………………………………………………………………………………………

**Introduction**  
  
PayPal company is a leading provider of secure and efficient payment solutions, facilitating seamless transactions for millions of users and merchants across US. It specializes in innovative financial technology, empowering businesses and individuals with reliable payment processing, robust security measures, and user-friendly platforms. The company strives to drive financial inclusion and enhance the digital economy through cutting-edge solutions.  
  
We have taken a dataset, and this project is focused on analyzing transactional data for a leading payment processing company, like PayPal, to gain insight into user behavior, transaction patterns, and overall business performance. The dataset comprised of many tables, but we took of three primary tables: 'Cards,' 'Users,' and 'Transactions.' The 'Cards' table contained information about customer payment cards, including card type (Visa, Mastercard, etc.) and associated user IDs. The 'Users' table held demographic data about the company's customer base. Finally, the 'Transactions' table recorded details of each transaction, such as transaction ID, timestamp, status (successful, failed, pending), and the associated card ID. This rich dataset provided a comprehensive view of the company's operational activities and customer interactions.

The tables within the dataset were interconnected through key identifiers. The 'Users' and 'Cards' tables were linked via the user ID, allowing us to associate card details with specific customer profiles. The 'Cards' and 'Transactions' tables were linked via the card ID, enabling us to trace transactions back to the originating card and user. To further enrich our analysis, we incorporated a fourth table: 'Merchants.' This table contained information about the businesses accepting payments through the platform, including merchant ID, location (state), and business category. The 'Merchant’s' table was linked to the 'Transactions' table via the merchant ID, providing insights into where transactions occurred, and the types of businesses involved. This interconnected data structure allowed for a holistic analysis of the payment ecosystem, from user demographics and card usage to transaction patterns and merchant locations.

Part 1: Data Sourcing  
  
**1.1 Raw data files in original format**  
  
In the ever-evolving landscape of digital transactions, staying ahead of the competition is very important.

The foundation of this project began with sourcing raw data from multiple sources, ensuring they were rich in transactional information and suitable for integration. The primary dataset, obtained from Kaggle’s **Transactions Dataset**, provided a wealth of real-world financial transactions, including fraudulent and legitimate activities. However, to enhance analytical depth, I identified two additional datasets—**which I created using python scripts**, both containing essential attributes that could be combined to provide a holistic view of transaction behaviors.

Each dataset met key criteria:

* They were in raw formats ensuring flexibility in preprocessing and transformation.
* They contained related yet distinct information, enabling a meaningful integration within a dimensional model.
* The datasets had sufficient records (over 500 transactions per source), ensuring analytical viability.

By sourcing and integrating these datasets, I laid the groundwork for an advanced business insights framework, paving the way for effective insights through dimensional modeling and data-driven analysis.  
  
**Kaggle Dataset: -** [**https://www.kaggle.com/datasets/computingvictor/transactions-fraud-datasets**](https://www.kaggle.com/datasets/computingvictor/transactions-fraud-datasets)

A screenshot of a computer

AI-generated content may be incorrect.

* 1. – Kaggle Dataset raw format.   
       
        
       
     **Building the Data Lakehouse on Azure**

With the foundational datasets identified, the next step was to establish a scalable and efficient infrastructure for data storage and processing. To achieve this, we leveraged **Microsoft Azure**, a powerful cloud computing platform, to create a **Data Lakehouse**—a hybrid architecture combining the best features of data lakes and data warehouses.

The process began with setting up an **Azure account**, followed by provisioning a **resource group** and a **storage account** to serve as the backbone of our data storage environment. To ensure flexibility and scalability, we deployed EC2 **General-Purpose Storage (Gen Storage)**, providing the necessary capacity to handle large volumes of structured and semi-structured data.

Once the infrastructure was in place, all sourced datasets, including the Kaggle **Transactions Fraud Dataset**, **User Account Information**, and **Merchant Details**—were ingested into the storage account in BLOB. This centralization transformed our data repository into a **Data Lakehouse**, enabling seamless integration, efficient querying, and optimized analytics.

This setup not only ensured data persistence and security but also laid the groundwork for further processing, transformation, and modeling, paving the way for advanced fraud detection insights.

A screenshot of a computer

AI-generated content may be incorrect.

1.2 – Azure Storage Account--------- Data Lakehouse.

**Connecting Azure Data Lakehouse to SSMS**

With the **Data Lakehouse** successfully established in **Azure**, the next step was to enable seamless data access for processing and analysis. To achieve this, we connected the Azure storage account to **SQL Server Management Studio (SSMS)**, allowing us to extract and manipulate data efficiently.

The integration process began by configuring **Azure Data Lake Storage (ADLS)** as a linked service within **SSMS**. Using **Azure Blob Storage connectors**, we established a secure connection, enabling **SQL Server** to interact with the raw datasets stored in the Lakehouse. This connection facilitated **data extraction, transformation, and querying**, ensuring smooth data flow into our analytical environment.

By bridging the gap between **Azure Data Lakehouse** and **SSMS**, we created a structured pipeline for managing and refining transactional data. This connection empowered us to perform **dimensional modeling, execute SQL-based transformations, and optimize the dataset for transaction analytics**. The foundation was now set for the next phase—building the data warehouse schema and implementing analytical processes.  
  
A screenshot of a computer

AI-generated content may be incorrect.

1.3 – SSMS connected to Azure

**1.2 Documentation describing the sources and their relationships  
1.3 Data dictionary explaining the fields in each source**

**Table: credit\_card\_info**

* **Description**: This table stores information about credit cards associated with clients. It includes details such as card brand, type, expiration, credit limits, and more.

**Columns and Data Types:**

1. **id**
   * **Data Type**: INT
   * **Description**: A unique identifier for each record in the table.
2. **client\_id**
   * **Data Type**: INT
   * **Description**: A unique identifier for each client. refers to the client in a separate client table.
3. **card\_brand**
   * **Data Type**: VARCHAR(50)
   * **Description**: The brand of the credit card (e.g., Visa, MasterCard, American Express).
   * **Example**: "Visa", "MasterCard", "American Express"
4. **card\_type**
   * **Data Type**: VARCHAR(50)
   * **Description**: The type of card (e.g., Credit, Debit, Prepaid).
   * **Example**: "Credit", "Debit",
5. **card\_number**
   * **Data Type**: VARCHAR(16) (or CHAR(16) depending on your system's storage needs)
   * **Description**: The credit card number (usually 16 digits, but it can vary). This column typically doesn't store the actual number in encrypted or masked form for security reasons.
   * **Example**: "4111111111111111", "5105105105105100"
6. **expiry\_date**
   * **Data Type**: DATE or VARCHAR(7)
   * **Description**: The expiration date of the card in MM/YYYY format.
7. **cvv**
   * **Data Type**: VARCHAR(4) or CHAR(3)
   * **Description**: The Card Verification Value (CVV), typically a 3-4 digit number found on the back of the card.
   * **Example**: "123", "456"
8. **has\_chip**
   * **Data Type**: BOOLEAN or BIT
   * **Description**: Indicates whether the card has a chip (1 for true, 0 for false).
9. **num\_cards**
   * **Data Type**: INT
   * **Description**: The number of cards associated with the client, possibly referring to additional cards linked to the same account.
   * **Example**: 1, 2, 3
10. **credit\_limit**
    * **Data Type**: DECIMAL(10, 2) or FLOAT
    * **Description**: The credit limit assigned to the card.
    * **Example**: 5000.00, 15000.50
11. **account\_open**
    * **Data Type**: DATE
    * **Description**: The date when the credit card account was opened.
    * **Example**: "2020-01-01", "2022-07-15"
12. **year\_pin\_last\_changed**
    * **Data Type**: DATE or YEAR
    * **Description**: The year (or exact date) when the card's PIN was last changed.
13. **card\_on\_demand**
    * **Data Type**: BOOLEAN or BIT
    * **Description**: A flag indicating if the card is an on-demand card (e.g., a virtual card created for temporary use).
    * **Example**: 1 (Yes), 0 (No)

**Table: merchants**

* **Description**: This table stores information about merchants. It includes basic details about the merchant such as their name, location, and transaction information.

**Columns and Data Types:**

1. **name**
   * **Data Type**: VARCHAR(255)
   * **Description**: The name of the merchant. This is typically the business name or the store name.
2. **id**
   * **Data Type**: INT
   * **Description**: A unique identifier for each merchant. This typically serves as the primary key for the table and is usually auto-incremented.
3. **city**
   * **Data Type**: VARCHAR(100)
   * **Description**: The city where the merchant is located.
4. **state**
   * **Data Type**: VARCHAR(100)
   * **Description**: The state or region where the merchant operates. This might be a U.S. state or another region depending on the country.
5. **zip**
   * **Data Type**: INT
   * **Description**: The postal code or ZIP code for the merchant's location.
6. **transaction\_id**
   * **Data Type**: INT
   * **Description**: A unique identifier for the transaction. This links the merchant to the specific transaction. It could be used to track sales, payments, or related activities.

**Transaction Table:**

* Description: This table consists of information about transaction like clients the card used, and merchant details of where it was used.

id

INT

The unique identifier for each transaction. This is the primary key.

client\_id

INT

The identifier of the client making the transaction. It references a client in the client table.

card\_id

INT

The unique identifier of the card used in the transaction. It can refer to a specific card issued to a client.

amount

DECIMAL(10,2)

The total monetary amount of the transaction, including taxes and fees. The amount is in the currency used in the transaction.

use\_chip

BOOLEAN

Indicates whether the chip feature of the card was used during the transaction.

merchant\_city

VARCHAR(255)

The city where the merchant is located.

merchant\_state

VARCHAR(255)

The state where the merchant is located.

zip

VARCHAR(10)

The zip code of the merchant’s location.

mcc

VARCHAR(4)

Merchant Category Code (MCC) that classifies the type of business of the merchant.

**Additional Transaction: This table contains extra meaningful information about the transaction like the status, time taken to complete or fail. The timestamp of the transaction and the transaction id which the unique identifier for each transaction.**

transaction\_time\_in\_seconds

INT

Number of seconds required for the transaction to reflect a proper status

Status

VARCHAR(50)

The status of the transaction (e.g., "Completed", "Pending", "Failed").

date

DATE

The date when the transaction occurred, formatted as YYYY-MM-DD.

transaction\_id

INT

The unique identifier for the transaction is in the additional details table. It corresponds to the id from the Transactions table.

Part 2. Normalized Database  
  
1NF--------- 2NF ------------- 3NF  
  
2.1 DDL scripts for creating the normalized database schema  
2.2 Scripts or documentation for loading raw data into normalized tables  
  
1NF – Merchants – Table (NULL Value) -----   
  
A screenshot of a computer

AI-generated content may be incorrect.

2NF – Users – Table (Address)  
  
1NF

USE [DHRUVIN]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[users](

  [id] [int] NOT NULL,

  [current\_age] [int] NULL,

  [retirement] [int] NULL,

  [birth\_year] [int] NULL,

  [birth\_month] [int] NULL,

  [gender] [varchar](10) NULL,

  [address] [varchar](50) NULL,

  [latitude] [float] NULL,

  [longitude] [float] NULL,

  [per\_capita] [int] NULL,

  [yearly\_income] [int] NULL,

  [total\_debt] [int] NULL,

  [credit\_score] [int] NULL,

  [num\_credit\_cards] [int] NULL,

PRIMARY KEY CLUSTERED

(

  [id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

A screenshot of a computer

AI-generated content may be incorrect.

3NF (Address ---Street Name, Apartment Number)  
  
USE [DHRUVIN]

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE VIEW [dbo].[users1] AS

SELECT

  id,

  current\_age,

  retirement,

  birth\_year,

  birth\_month,

  gender,

-- Extract the Apartment Number (if present)

CASE

WHEN CHARINDEX(' ', address) > 0 THEN

LEFT(address, CHARINDEX(' ', address) - 1)

ELSE

NULL

END AS apartment\_number,

-- Extract the Street Name (everything after the first space)

CASE

WHEN CHARINDEX(' ', address) > 0 THEN

LTRIM(RIGHT(address, LEN(address) - CHARINDEX(' ', address)))

ELSE

address

END AS street\_name,

  latitude,

  longitude,

  per\_capita,

  yearly\_income,

  total\_debt,

  credit\_score,

  num\_credit\_cards

FROM users;

GO

A screenshot of a computer

AI-generated content may be incorrect.

**2.3 - ER diagram of the normalized database.**  
  
A screenshot of a computer

AI-generated content may be incorrect.

**Part 3. ETL Implementation**  
  
3.1 Complete ETL code  
3.4 Logs or screenshots showing successful ETL execution  
  
**Transformation 1 - This view is created to create a table containing different information from transaction and credit card info table. It performs joints and connects different tables to give a comprehensive sense of all tables into one.**

CREATE VIEW [view1] AS

SELECT

    transactions.[id],

    users.[id] AS [client\_id],

    [credit\_card\_info].[id] AS [card\_id],

    transactions.[amount],

    transaction\_additional\_details.[transaction\_time\_in\_seconds],

    transaction\_additional\_details.[date],

    transaction\_additional\_details.[status],

    transactions.[use\_chip],

    --merchants.[merchant\_id],

    merchants1.[id]  AS [merchant\_id] ,

    transactions.[merchant\_city],

    transactions.[merchant\_state],

    transactions.[zip],

    transactions.[mcc]

    FROM [transactions]

INNER JOIN

    [transaction\_additional\_details] ON transactions.[id] = transaction\_additional\_details.[transaction\_id]

INNER JOIN

    [merchants1] ON merchants1.[transaction\_id] = transactions.[id]

INNER JOIN

    [users] ON users.id = transactions.[client\_id]

INNER JOIN

    [credit\_card\_info] ON [credit\_card\_info].[id] = transactions.[card\_id]

A screenshot of a computer

AI-generated content may be incorrect.

**Transformation 2 - This view is used to process the available states and categorize them into regions that can help better plot correct graphs regarding the transactions.**  
  
CREATE VIEW [merchants\_region] AS

SELECT

    [name],

    [id],

    [city],

    [state],

    [zip],

    CASE

        WHEN state IN ('NY', 'NJ', 'PA', 'CT', 'MA', 'RI', 'VT', 'NH', 'ME') THEN 'Northeast'

        WHEN state IN ('IL', 'IN', 'MI', 'OH', 'WI', 'IA', 'MN', 'ND', 'SD') THEN 'Midwest'

        WHEN state IN ('TX', 'FL', 'GA', 'AL', 'SC', 'NC', 'KY', 'TN', 'MS', 'LA', 'AR', 'OK', 'MO', 'WV', 'VA', 'DC') THEN 'South'

        WHEN state IN ('CA', 'WA', 'OR', 'NV', 'AZ', 'CO', 'UT', 'ID', 'MT', 'WY') THEN 'West'

        WHEN state = 'Mexico' THEN 'Other'

        ELSE 'Unknown'

    END AS region,

    [transaction\_id]

FROM merchants1

Select \* from [merchants\_region

A screenshot of a computer

AI-generated content may be incorrect.

**3.3 Data Quality: We update the null values present in the merchant state column using the following query to improve data quality. The raw data contained empty values because none of the online transactions contained the physical address of the merchant.**

UPDATE transactions  
SET merchant\_state= 'NO STATE'  
WHERE merchant\_state IS NULL;

And the same was done for zip codes as well but we use 0 as the column data type is of integer.

UPDATE transactions  
SET zip= '0'  
WHERE zip IS NULL;

A screenshot of a computer screen

AI-generated content may be incorrect.

**Part 4: Dimensional Model:**

**4.1** DDL scripts for creating the dimensional model

**1. Merchants Dimension Table**

USE [DHRUVIN]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[merchants1](

[name] [varchar](50) NULL,

[id] [varchar](50) NOT NULL,

[city] [varchar](50) NULL,

[state] [varchar](50) NULL,

[zip] [int] NULL,

[transaction\_id] [int] NULL,

PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

**2. Credit Card**

USE [DHRUVIN]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[credit\_card\_info](

[id] [int] NOT NULL,

[client\_id] [int] NULL,

[card\_brand] [varchar](50) NULL,

[card\_type] [varchar](50) NULL,

[card\_number] [varchar](50) NULL,

[expiry\_date] [datetime] NULL,

[cvv] [int] NULL,

[has\_chip] [varchar](3) NULL,

[num\_cards] [int] NULL,

[credit\_limit] [int] NULL,

[account\_open] [datetime] NULL,

[year\_pin\_last\_changed] [int] NULL,

[card\_on\_demand] [varchar](3) NULL,

PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

**3. Clients/Users**

USE [DHRUVIN]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE VIEW [dbo].[users1] AS

SELECT

id,

current\_age,

retirement,

birth\_year,

birth\_month,

gender,

-- Extract the Apartment Number (if present)

CASE

WHEN CHARINDEX(' ', address) > 0 THEN

LEFT(address, CHARINDEX(' ', address) - 1)

ELSE

NULL

END AS apartment\_number,

-- Extract the Street Name (everything after the first space)

CASE

WHEN CHARINDEX(' ', address) > 0 THEN

LTRIM(RIGHT(address, LEN(address) - CHARINDEX(' ', address)))

ELSE

address

END AS street\_name,

latitude,

longitude,

per\_capita,

yearly\_income,

total\_debt,

credit\_score,

num\_credit\_cards

FROM users;

GO  
  
 **4.2 Star/snowflake schema diagram**  
  
A screenshot of a computer

AI-generated content may be incorrect.

**4.3 Documentation of dimension and fact table designs**  
  
**Fact Table Documentation**

This fact table integrates data from several key tables in the database, including **transactions**, **transaction\_additional\_details**, **merchants1**, **users**, and **credit\_card\_info**, to provide a comprehensive view of each transaction's details. The **id** represents the unique transaction identifier, while **client\_id** links to the customer making the transaction, and **card\_id** corresponds to the credit card used. The **amount** column captures the monetary value of the transaction, while **transaction\_time\_in\_seconds** reflects the duration of the transaction process. The **date** field records the exact date of the transaction, and **status** indicates whether the transaction was successful or failed. Additionally, the **use\_chip** column specifies whether a chip card was used for the transaction. The **merchant\_id**, **merchant\_city**, **merchant\_state**, **zip**, and **mcc** fields provide detailed merchant-specific information, including the location of the merchant and the type of business. This table is created by joining the respective tables using their foreign keys to provide a unified view of the transaction, client, credit card, and merchant details. This consolidated data is essential for various business analytics purposes, such as tracking spending patterns, evaluating transaction success rates, and understanding customer behavior across different regions and merchants.

**Dimensional Table**

* + - The **clients’ dimension table** holds essential information about the customers in the system. The **id** column represents the unique identifier for each client, while additional attributes such as **id, age, gender** and **address** capture important contact and location details of the customer. This table serves as a key reference to link each transaction or card to the individual client, enabling insights into customer behavior and demographics.
    - The **credit card info dimension table** stores details regarding the credit cards used by clients in transactions. The **id** column represents the unique identifier for each card, with additional attributes such as **card\_number**, **expiry\_date**, and **card\_type** describing each card's specific information. The **client\_id** field associates each card to a specific client, allowing for a comprehensive view of the client's credit card usage, and enabling analyses of credit usage patterns, card types, and limits.
    - The **merchants dimension table** contains detailed information about the merchants where transactions take place. The **id** column identifies the merchant uniquely, while attributes such as **merchant\_name**, **merchant\_type**, **merchant\_address**, **city**, **state**, **zip\_code**, and **mcc** provide detailed location and business type data. This table allows for merchant-based analysis, such as identifying top-performing merchants, understanding regional spending trends, and classifying merchants by business type or category using the Merchant Category Code (MCC).

**4.4 Explanation of how dimensions were derived from normalized sources**  
  
We extract the **client\_id** along with other client-specific attributes (like **id**, **age**, **address**, **retirement, credit score and income**) from the transactional data. This allows us to create a separate **clients** dimension table that includes these descriptive attributes for each unique client.

Attributes like **card\_number**, **expiry\_date**, **card\_type**, and the **client\_id** (foreign key) are extracted from the credit card information associated with each transaction. This forms the **credit card info** dimension table.

The **merchant\_id** along with its related descriptive data (e.g., merchant name, location, and type) are extracted from the fact table, forming a **merchants** dimension table.

**Part 5. Analytical Queries**  
  
**Query 1: This dashboard analyzes merchant performance, revealing that "NO STATE" (online transactiona) dominates transaction volume, followed by TX, OH, and MD.**  
  
 Success rates vary significantly across states and regions, with the "Unknown" region (also likely linked to online transaction data) showing high transaction counts but not necessarily high success rates.   
  
The analysis suggests a need for a focus on high-volume states, investigation into low success rates.  
A screenshot of a computer

AI-generated content may be incorrect.

A close-up of a chart

AI-generated content may be incorrect.

**Query 2: This dashboard reveals that the majority of customers (81.05%) have high credit limits but low spending, indicating potential underutilization of available credit.**

A small percentage (1.09%) are overspent, suggesting effective credit management overall. A detailed table shows specific examples of customers with unused credit, while a donut chart displays the distribution of total spending across customers.

The analysis highlights a need to investigate why customers aren't using their full credit limits and potentially tailor credit products and marketing strategies accordingly.  
A screenshot of a computer

AI-generated content may be incorrect.  
  
A screenshot of a data report

AI-generated content may be incorrect.

**Query 3: This dashboard profiles customer age, revealing that customers in their mid-50s have the highest average yearly income and transaction counts, while credit scores remain consistently high across the analyzed age range (40s to mid-60s).**

The sum of yearly income varies significantly by state, with California leading. These findings suggest targeted marketing strategies based on age and state, focusing on high-earning, active customers in their 50s and high-income states like California.

A screenshot of a computer

AI-generated content may be incorrect.  
  
A screenshot of a graph

AI-generated content may be incorrect.

**Query 4: Our transaction volume has fluctuated over the past five years, indicating inconsistent performance.  Furthermore, the number of failed and pending transactions is alarmingly close to the number of successful ones. This necessitates immediate attention from our software team to identify and resolve the underlying technical issues hindering smooth transaction processing**.

We've also observed that successful transactions are taking significantly longer than 100 seconds to complete. This extended wait time negatively impacts the customer experience, potentially leading to dissatisfaction and churn.

Geographic analysis reveals that California and New York are our primary transaction hubs.  To expand our market reach and diversify our customer base, we should strategically increase our payment merchant presence in underrepresented states like Nevada, Utah, and Colorado.

A screenshot of a computer

AI-generated content may be incorrect.  
  
A close-up of a graph

AI-generated content may be incorrect.

**Part 6: Final Report**  
  
**6.1 Summary of the Project Approach and Implementation**

The project aimed to design and implement a data warehousing solution to manage large-scale transaction data. The approach included:

Data Collection: Extracted and cleaned transaction, client, and merchant data.

Dimensional Modeling: Designed a star schema with fact and dimension tables (transactions, clients, merchants).

ETL Process: Developed an ETL pipeline for data processing.

Data Storage: Used cloud-based storage with partitioning and indexing for optimization. Used Azure Data Lake and containers for storing raw and normalized data.

Reporting: Integrated Power Bi tool for data analysis and reporting.

**6.2 Discussion of Challenges Encountered and Solutions Applied**

Data Quality: Inconsistent and missing data was cleaned using validation and default values.

Dimensional Modeling Complexity: Unfortunately, historical data was not present in this dataset. We plan to implement this in the future analysis completely, changing the dimensional dynamics.

System Integration: Custom integration was done to directly dump data into Data Warehouse.

**6.3 Analysis of the Effectiveness of the Dimensional Model**

The dimensional model was highly effective in enabling efficient querying and reporting. It supported fast data retrieval, flexible analysis, and was easy for users to navigate. However, performance depends on regular maintenance and updates to the system.

**6.4 Recommendations for Future Improvements**

Real-time Processing: The data is static however it gets processed in real time as the dashboard refreshes.

AI Integration: AI was not integrated into this system. However, this remains a great option to provide more information and insights into the data.

Historical Data Management: As there was no historical data present there was no need to manage it. However, when implemented it will properly be handled by using SQL queries to validate the current data required for the analysis.

Cross-Domain Analytics: to perform the analysis and the assignment Azure Data Storage, SQL server and Power BI tools were used.