Data Warehousing

Final Project

Prof. Don Berdnt

Group Members:

Venkata Dharma Teja

Janhavi Kharmale

Manvith Katkuri

Prashanth Kumar Vootla

Sai Sankalp Thota

Siva Tarun Reddy Korrapati

1. Executive Summary

The Financial Transactions Analytics project aims to create a data warehouse solution that facilitates comprehensive insights into customer financial behaviors, card usage patterns, and transactional trends. By consolidating and transforming data from three primary sources—transactions, users, and cards—this project will provide a unified, subject-oriented repository optimized for high-value decision-making in the financial services industry. The project focuses on developing a robust dimensional model, enabling a wide array of analytical queries and data visualizations.

To address specific managerial decision-making needs, we designed a dimensional data model with fact and dimension tables. The transactional model captures detailed user data and card information in a normalized form to support operational needs, while the dimensional model supports high-level analytics on spending patterns, card usage trends, and customer demographics. This approach enables us to not only streamline transactional processing but also empower data-driven strategies such as customer segmentation, risk assessment, and marketing optimization.

The data collection and preparation processes involved transforming raw datasets into a structured, cleaned format suitable for analysis. Key SQL transformations were used to standardize data, handle missing values, and create meaningful aggregates for reporting. Additionally, data enrichment efforts integrated per capita income and credit scores, providing a broader context for financial insights.

This project delivers a flexible data warehousing solution that supports in-depth analysis and strategic decision-making, with potential applications in fraud detection, personalized marketing, and financial forecasting.

2. Problem Statement

In the highly competitive financial services industry, understanding customer spending behavior, identifying fraud risks, and personalizing marketing strategies are critical for success. Financial institutions need a centralized, data-driven approach to make informed decisions on these fronts. The challenge arises from disparate datasets—such as transaction records, customer demographics, and card usage details—that, when analyzed together, could reveal valuable insights.
Who: Financial analysts, risk managers, and marketing teams. What: The challenge is to integrate and analyze various data sources to enhance customer insights and mitigate risks. When and Where: This need is ongoing and applies across financial institutions dealing with high transaction volumes and diverse customer bases. Why: Analyzing this data can improve risk assessment, enhance customer targeting, and optimize revenue

strategies. How: A well-designed data warehouse that consolidates transactional and demographic data can enable powerful analytics and decision-making.
By implementing a data warehouse with a dimensional model, this project aims to support strategic decisions and enable deeper insights through an accessible, integrated data repository.

3. Literature Review

For this project, a range of resources informed the approach to data warehousing and dimensional modeling, each contributing valuable insights into best practices and methodologies:

Bill Inmon's Data Warehousing Concepts: Inmon's foundational work on data warehousing emphasizes the importance of building a subject-oriented, time-variant, and integrated database to support decision-making. His guidance on organizing data around key business subjects influenced the project's approach to integrating transactional and demographic data, focusing on usability for analytics.

Ralph Kimball's Dimensional Modeling Techniques: Kimball's dimensional modeling techniques, particularly his "star schema" design, were pivotal in shaping the database structure. His work underscores the importance of a simplified schema for analytical efficiency, guiding the development of fact and dimension tables to streamline querying and reporting.

Financial Transaction Data Analysis Blog Posts and Tutorials: Several blog posts and tutorials focused on analyzing transaction data provided practical advice on structuring financial data, handling sensitive information, and detecting fraud patterns. These resources underscored the value of creating a unified data model that could support various analytics use cases, from spending trend analysis to fraud detection.

SQL and Data Warehousing Documentation: Official SQL documentation and data warehousing resources on schema design and ETL processes offered technical details essential for implementing the project. These resources included information on SQL best practices for query optimization, data cleaning, and data aggregation.

Academic Papers on Customer Analytics and Fraud Detection: Research papers detailing advanced customer analytics and fraud detection methodologies highlighted the importance of integrating transactional and customer demographic data. These insights helped shape the project's focus on enabling data-driven insights into spending patterns, customer segmentation, and fraud detection through integrated datasets.

Together, these resources provided a comprehensive foundation for designing a financial data warehouse that aligns with industry standards and addresses real-world analytical needs in financial services.

4. Data Collection and Preparation

The data for this project was sourced from Kaggle, specifically from the "Transactions Fraud Datasets" dataset, which can be accessed from this link:

https://www.kaggle.com/datasets/computingvictor/transactions-fraud-datasets?select=users_data.csv

This dataset provides essential information for financial analytics and fraud detection, including transactions, user demographics, and card details.

Data Collection

The dataset comprises three main CSV files:
Transactions Data: Contains information on each transaction, including transaction amount, card ID, merchant details, and transaction method (chip or swipe).
Users Data: Includes demographic and financial information about clients, such as age, income, debt, and credit score.
Cards Data: Provides details on client cards, including card brand, credit limit, card status, and expiration details.
These CSV files were imported into a database where each file was transformed into a separate table, facilitating the integration and relational analysis across datasets.
Data Preparation
To ensure data quality and consistency, the following data preparation steps were taken:

Data Cleaning:

- Missing Values: Rows with missing or irrelevant data were identified and removed to improve data reliability.
- Standardization: Fields such as dates, currency amounts, and geographic data were standardized. For instance, date formats were unified to support time-based analysis, and financial fields were converted to numeric types for accurate aggregation and comparison.
- Currency Formatting: Transaction amounts and financial fields (such as yearly_income, total_debt, and credit_limit) were cleaned and standardized to remove symbols or inconsistencies.

Data Transformation:

- Transaction Amounts: Converted to a consistent currency format, and negative values were flagged to differentiate debits from credits.
- Geographic Data: Fields such as merchant_state and merchant_city were standardized for consistency in location-based analytics.
- Account Security: Extracted information like card expiration year and last PIN change year from the Cards Data table to evaluate security aspects, such as outdated cards.

ETL Process:

- After cleaning and transforming the data, an ETL (Extraction, Transformation, and Loading) process was performed, which involved importing the data from CSV files into a structured relational database format. This made the data readily accessible for SQL-based queries and further analysis.

5. Database Design

The database design for this project includes both a transactional (OLTP) model and a dimensional (OLAP) model. These two models serve different purposes: the transactional model captures detailed day-to-day data, while the dimensional model focuses on simplifying data for analytics and decision-making.

5.1 Transactional Models (OLTP)

The OLTP model in this project is designed to capture detailed records of financial transactions, user demographics, and card information. This design is normalized to ensure data integrity and prevent redundancy, which is essential for supporting high-frequency transactional operations in real-time. The OLTP schema is structured as follows:

- **Transactions Table:** Stores each transaction, with details such as transaction_id, date, client_id, card_id, amount, merchant_id, merchant_city, and merchant_state. This table is optimized for rapid insertion and retrieval, essential for processing high volumes of transactions.

- **Users Table:** Contains demographic information about each user, including user_id, current_age, retirement_age, birth_year, birth_month, gender, address, latitude, longitude, per_capita_income, yearly_income, total_debt, credit_score, and num_credit_cards.
- **Cards Table:** Holds card-related information such as card_id, client_id, card_brand, card_type, card_number, expires, cvv, has_chip, num_cards_issued, credit_limit, acct_open_date, year_pin_last_changed, and card_on_dark_web.

This OLTP model is normalized (typically to third normal form) to prevent data redundancy, improve data integrity, and allow for fast transactions. Indexing is applied to frequently accessed fields such as client_id, card_id, and merchant_id to enhance query performance. This model supports various real-time applications, such as verifying card transactions and updating user account details.

5.2 Dimensional Models (OLAP)

The dimensional model is simplified to focus on data aggregation and analysis, helping to identify transaction patterns, customer segmentation, and potential fraud. The star schema for the dimensional model consists of a central fact table and surrounding dimension tables. This structure allows for quick retrieval of aggregated data and is ideal for decision support.

Fact Table:

- Transaction Facts: The Transaction_Fact table contains metrics such as transaction_id, date_key, client_id, card_id, merchant_id, and amount. It serves as the primary table for financial analytics, storing transaction-related facts and connecting with relevant dimensions for slicing and dicing the data.

Dimension Tables:

- User Dimension: The User_Dim table stores demographic attributes such as client_id, current_age, retirement_age, gender, income, debt, and credit_score. This dimension is essential for segmenting users by demographic and financial profiles.
- Card Dimension: The Card_Dim table includes card_id, card_brand, card_type, credit_limit, and has_chip, which are useful for analyzing card usage patterns.
- Merchant Dimension: The Merchant_Dim table stores attributes like merchant_id, merchant_city, and merchant_state, allowing for geographic and merchant-based analysis.

This dimensional model, based on a star schema, enables easy querying and data aggregation, supporting the OLAP system's goal of providing insights for decision-making.

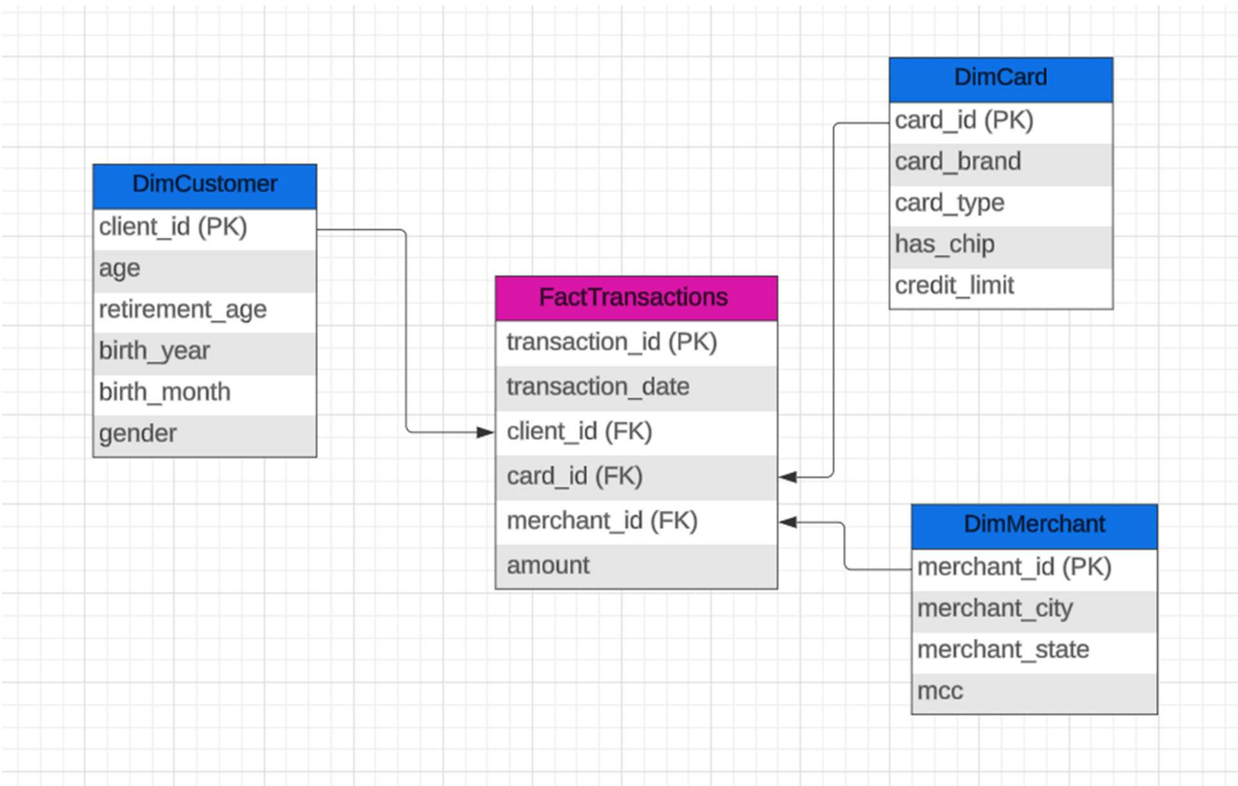
Data Cube Description

Data cubes are developed from the dimensional model to support multi-dimensional analysis. For instance:

- Spending by Geography and Time: A data cube aggregates transaction amounts by geographic regions (state, city) and over different time periods (month, quarter, year). This helps in identifying high-spending areas and seasonal trends.

- User Credit Utilization: A data cube aggregates average credit utilization rates per user across different demographic segments, enabling targeted financial management advice and risk assessment.
  - Merchant Transaction Patterns: A cube aggregates transaction counts and amounts per merchant, helping to identify merchants with unusual transaction patterns that may indicate fraud.

Entity Relationship Diagram:



Creating Staging Tables:

```
CREATE TABLE Transactions_Staging_Data (
    id NUMBER PRIMARY KEY,
    transaction_date DATE,
    client_id NUMBER,
    card_id NUMBER,
```

```sql
    amount NUMBER(10, 2),
    use_chip VARCHAR2(50),
    merchant_id NUMBER,
    merchant_city VARCHAR2(100),
    merchant_state VARCHAR2(2),
    zip NUMBER(5),
    mcc NUMBER,
    errors VARCHAR2(255)
);
```

```sql
select * from Transactions_Staging_Data;
```

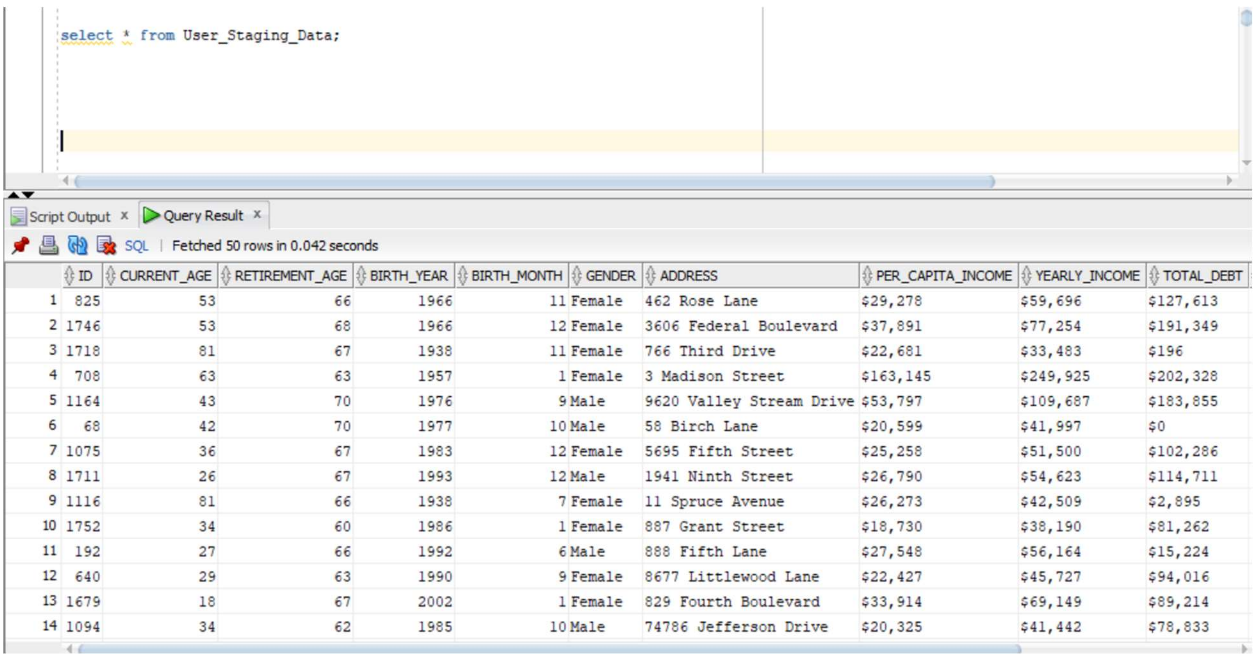
Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.037 seconds

	ID	TRANSACTION_DATE	CLIENT_ID	CARD_ID	AMOUNT	USE_CHIP	MERCHANT_ID	MERCHANT_CITY	MERCHANT_STATE	ZIP	MCC	ERROR
1	7475327	01-JAN-10	1556	2972	-77	Swipe Transaction	59935	Beulah	ND	58523	5499	(null)
2	7475328	01-JAN-10	561	4575	14.57	Swipe Transaction	67570	Bettendorf	IA	52722	5311	(null)
3	7475329	01-JAN-10	1129	102	80	Swipe Transaction	27092	Vista	CA	92084	4829	(null)
4	7475331	01-JAN-10	430	2860	200	Swipe Transaction	27092	Crown Point	IN	46307	4829	(null)
5	7475332	01-JAN-10	848	3915	46.41	Swipe Transaction	13051	Harwood	MD	20776	5813	(null)
6	7475333	01-JAN-10	1807	165	4.81	Swipe Transaction	20519	Bronx	NY	10464	5942	(null)
7	7475334	01-JAN-10	1556	2972	77	Swipe Transaction	59935	Beulah	ND	58523	5499	(null)
8	7475335	01-JAN-10	1684	2140	26.46	Online Transaction	39021	ONLINE	(null)	(null)	4784	(null)
9	7475336	01-JAN-10	335	5131	261.58	Online Transaction	50292	ONLINE	(null)	(null)	7801	(null)
10	7475337	01-JAN-10	351	1112	10.74	Swipe Transaction	3864	Flushing	NY	11355	5813	(null)
11	7475338	01-JAN-10	554	3912	3.51	Swipe Transaction	67570	Pearland	TX	77581	5311	(null)
12	7475339	01-JAN-10	605	5061	2.58	Swipe Transaction	75781	Brooklyn	NY	11210	5411	(null)
13	7475340	01-JAN-10	1556	2972	39.63	Swipe Transaction	59935	Beulah	ND	58523	5499	(null)
14	7475341	01-JAN-10	1797	1127	43.33	Swipe Transaction	33326	Kahului	HI	96732	4121	(null)
15	7475342	01-JAN-10	114	3398	49.42	Swipe Transaction	61195	North Hollywood	CA	91606	5541	(null)
16	7475343	01-JAN-10	1634	2464	1.09	Swipe Transaction	20519	San Benito	TX	78586	5942	(null)
17	7475344	01-JAN-10	646	2093	73.79	Swipe Transaction	1636	Erie	PA	16511	7538	(null)
18	7475345	01-JAN-10	1129	5492	100	Swipe Transaction	27092	Vista	CA	92084	4829	(null)

```sql
CREATE TABLE User_Staging_Data (
  id NUMBER PRIMARY KEY,
  current_age NUMBER,
  retirement_age NUMBER,
  birth_year NUMBER,
  birth_month NUMBER,
  gender VARCHAR2(10),
```

    address VARCHAR2(255),
    per_capita_income VARCHAR2(20),
    yearly_income VARCHAR2(20),
    total_debt VARCHAR2(20),
    credit_score NUMBER,
    num_credit_cards NUMBER
);



```
select * from User_Staging_Data;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.042 seconds

	ID	CURRENT_AGE	RETIREMENT_AGE	BIRTH_YEAR	BIRTH_MONTH	GENDER	ADDRESS	PER_CAPITA_INCOME	YEARLY_INCOME	TOTAL_DEBT
1	825	53	66	1966	11	Female	462 Rose Lane	$29,278	$59,696	$127,613
2	1746	53	68	1966	12	Female	3606 Federal Boulevard	$37,891	$77,254	$191,349
3	1718	81	67	1938	11	Female	766 Third Drive	$22,681	$33,483	$196
4	708	63	63	1957	1	Female	3 Madison Street	$163,145	$249,925	$202,328
5	1164	43	70	1976	9	Male	9620 Valley Stream Drive	$53,797	$109,687	$183,855
6	68	42	70	1977	10	Male	58 Birch Lane	$20,599	$41,997	$0
7	1075	36	67	1983	12	Female	5695 Fifth Street	$25,258	$51,500	$102,286
8	1711	26	67	1993	12	Male	1941 Ninth Street	$26,790	$54,623	$114,711
9	1116	81	66	1938	7	Female	11 Spruce Avenue	$26,273	$42,509	$2,895
10	1752	34	60	1986	1	Female	887 Grant Street	$18,730	$38,190	$81,262
11	192	27	66	1992	6	Male	888 Fifth Lane	$27,548	$56,164	$15,224
12	640	29	63	1990	9	Female	8677 Littlewood Lane	$22,427	$45,727	$94,016
13	1679	18	67	2002	1	Female	829 Fourth Boulevard	$33,914	$69,149	$89,214
14	1094	34	62	1985	10	Male	74786 Jefferson Drive	$20,325	$41,442	$78,833

CREATE TABLE Card_Staging_Data (
    id NUMBER PRIMARY KEY,
    client_id NUMBER,
    card_brand VARCHAR2(20),
    card_type VARCHAR2(20),
    card_number VARCHAR2(19),
    expires VARCHAR2(7),
    cvv NUMBER(3),

```
    has_chip VARCHAR2(3),
    num_cards_issued NUMBER,
    credit_limit NUMBER,
    acct_open_date VARCHAR2(7),
    year_pin_last_changed NUMBER(4),
    card_on_dark_web VARCHAR2(3)
);
```

```
select * from Card_Staging_Data;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.027 seconds

	ID	CLIENT_ID	CARD_BRAND	CARD_TYPE	CARD_NUMBER	EXPIRES	CVV	HAS_CHIP	NUM_CARDS_ISSUED	CREDIT_LIMIT	ACCT_OPEN_DATE	YEAR_
1	4524	825	Visa	Debit	4.34468E+15	Dec-22	623	YES	2	24295	Sep-02	
2	2731	825	Visa	Debit	4.95697E+15	Dec-20	393	YES	2	21968	Apr-14	
3	3701	825	Visa	Debit	4.58231E+15	Feb-24	719	YES	2	46414	Jul-03	
4	42	825	Visa	Credit	4.87949E+15	Aug-24	693	NO	1	12400	Jan-03	
5	4659	825	Mastercard	Debit (Prepaid)	5.72287E+15	Mar-09	75	YES	1	28	Sep-08	
6	4537	1746	Visa	Credit	4.4049E+15	Sep-03	736	YES	1	27500	Sep-03	
7	1278	1746	Visa	Debit	4.00148E+15	Jul-22	972	YES	2	28508	Feb-11	
8	3687	1746	Mastercard	Debit	5.62722E+15	Jun-22	48	YES	2	9022	Jul-03	
9	3465	1746	Mastercard	Debit (Prepaid)	5.71138E+15	Nov-20	722	YES	2	54	Jun-10	
10	3754	1746	Mastercard	Debit (Prepaid)	5.76612E+15	Feb-23	908	YES	1	99	Jul-06	
11	5144	1718	Mastercard	Debit	5.4952E+15	Mar-22	677	YES	2	31599	Oct-09	
12	2029	1718	Mastercard	Debit	5.8045E+15	Jul-23	258	NO	2	27480	Mar-02	
13	2379	1718	Mastercard	Debit	5.76635E+15	Feb-20	992	YES	1	26743	Mar-19	
14	2732	1718	Visa	Debit	4.24202E+15	Jun-20	928	YES	1	31463	Apr-14	
15	4706	1718	Mastercard	Debit	5.19103E+15	Jun-24	360	YES	1	16055	Sep-09	
16	281	708	Visa	Credit	4.01726E+15	May-15	877	YES	2	98100	Jan-11	
17	1106	708	Mastercard	Debit (Prepaid)	5.58197E+15	Jun-20	448	YES	1	62	Feb-07	

Creating Dimension Tables

The DimCustomer table is a dimension table created to store information about customers in a format optimized for analysis in a data warehouse. It is derived from raw data in the User_Staging_Data table, which likely holds unprocessed or less structured data.

Purpose and Use of DimCustomer Table

1. **Central Repository for Customer Attributes**:

- The DimCustomer table consolidates important customer-related attributes such as age, gender, birth details, income, debt, credit score, and the number of credit cards.
- This centralization provides a single, clean source of customer data that can be easily queried and analyzed.

2. **Data Cleansing and Transformation**:
   - The table cleans and validates numeric fields like per_capita_income, yearly_income, and total_debt using REGEXP_LIKE to ensure they contain valid numeric data. Invalid entries are replaced with NULL.
   - By transforming data during creation, it ensures that only validated and correctly formatted data is stored in the data warehouse, improving data quality.
3. **Primary Key for Data Integration**:
   - The primary key, client_id, uniquely identifies each customer. This key is essential for linking customer data in this dimension table to related records in **fact tables** (e.g., FactSales, FactTransactions).

CREATING TABLE

```
CREATE TABLE DimCustomer AS
SELECT
   id AS client_id,
   current_age AS age,
   retirement_age,
   birth_year,
   birth_month,
   gender,
  CASE WHEN REGEXP_LIKE(per_capita_income, '^[0-9]+(\.[0-9]+)?$')
       THEN TO_NUMBER(per_capita_income, '999999.99')
       ELSE NULL END AS per_capita_income,
   CASE WHEN REGEXP_LIKE(yearly_income, '^[0-9]+(\.[0-9]+)?$')
       THEN TO_NUMBER(yearly_income, '999999.99')
       ELSE NULL END AS yearly_income,
   CASE WHEN REGEXP_LIKE(total_debt, '^[0-9]+(\.[0-9]+)?$')
```

```
        THEN TO_NUMBER(total_debt, '999999.99')
        ELSE NULL END AS total_debt,
    credit_score,
    num_credit_cards
FROM
    User_Staging_Data;


INSERT INTO DimCustomer (client_id, age, retirement_age, birth_year, birth_month, gender, per_capita_income, yearly_income, total_debt, credit_score, num_credit_cards)
SELECT
    id AS client_id,
    current_age AS age,
    retirement_age,
    birth_year,
    birth_month,
    gender,
    CASE WHEN REGEXP_LIKE(per_capita_income, '^[0-9]+(\.[0-9]+)?$')
        THEN TO_NUMBER(per_capita_income, '999999.99')
        ELSE NULL END AS per_capita_income,
    CASE WHEN REGEXP_LIKE(yearly_income, '^[0-9]+(\.[0-9]+)?$')
        THEN TO_NUMBER(yearly_income, '999999.99')
        ELSE NULL END AS yearly_income,
    CASE WHEN REGEXP_LIKE(total_debt, '^[0-9]+(\.[0-9]+)?$')
        THEN TO_NUMBER(total_debt, '999999.99')
        ELSE NULL END AS total_debt,
    credit_score,
    num_credit_cards
FROM
```

User_Staging_Data;

-- Add primary key to DimCustomer

ALTER TABLE DimCustomer

ADD CONSTRAINT pk_dimcustomer PRIMARY KEY (client_id);

```
select * from DimCustomer;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.034 seconds

	CLIENT_ID	AGE	RETIREMENT_AGE	BIRTH_YEAR	BIRTH_MONTH	GENDER	PER_CAPITA_INCOME	YEARLY_INCOME	TOTAL_DEBT	CREDIT_SCORE	NUM_CREDIT_C
1	1479	32	60	1987	11	Male	(null)	(null)	(null)	805	
2	1594	39	64	1980	11	Male	(null)	(null)	(null)	643	
3	1485	64	69	1955	10	Male	(null)	(null)	(null)	618	
4	1390	37	71	1982	6	Male	(null)	(null)	(null)	850	
5	644	58	65	1961	3	Female	(null)	(null)	(null)	741	
6	1912	37	60	1982	7	Female	(null)	(null)	(null)	684	
7	1584	27	69	1992	9	Male	(null)	(null)	(null)	692	
8	1803	26	60	1993	4	Female	(null)	(null)	(null)	578	
9	1824	23	65	1996	7	Male	(null)	(null)	(null)	838	
10	340	20	63	1999	8	Male	(null)	(null)	(null)	809	
11	650	45	65	1974	8	Male	(null)	(null)	(null)	707	
12	762	49	59	1970	9	Male	(null)	(null)	(null)	683	
13	1061	25	67	1994	12	Male	(null)	(null)	(null)	739	
14	1931	58	72	1961	6	Male	(null)	(null)	(null)	503	
15	702	46	68	1973	7	Female	(null)	(null)	(null)	721	
16	918	43	75	1976	5	Female	(null)	(null)	(null)	703	
17	1137	52	67	1967	10	Male	(null)	(null)	(null)	669	
18	66	20	71	1999	6	Female	(null)	(null)	(null)	721	

The DimCard table is a dimension table designed to store information about credit or debit cards. It is derived from raw data in the Card_Staging_Data table, where data undergoes initial cleaning and transformation before being loaded into this structured dimension table in the data warehouse.

Purpose and Use of DimCard Table

1. Central Repository for Card Attributes:
    - The DimCard table consolidates key attributes about cards, such as card_brand, card_type, has_chip, and credit_limit.

- This setup provides a single, clean source of card-related data that is easy to query and analyze.

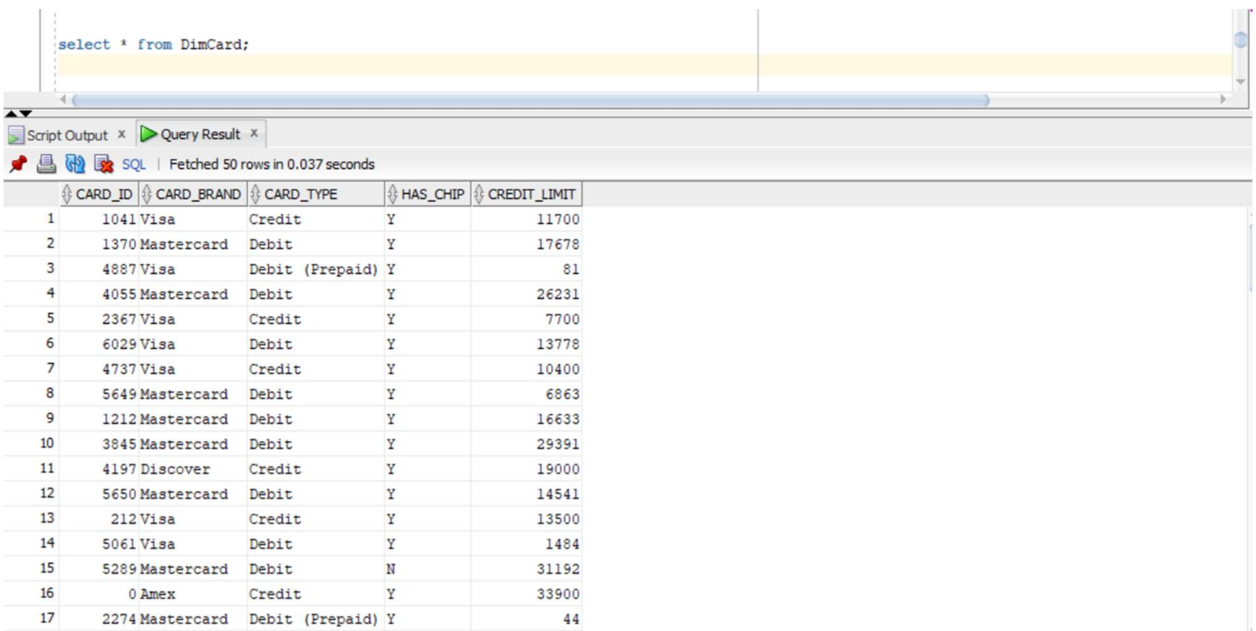
2. Data Transformation and Standardization:
    - The table standardizes the has_chip field, converting values of 'YES' to 'Y' and other values to 'N'. This standardization makes the data more consistent and easier to work with in queries.
    - It also converts the credit_limit to a numeric format using TO_NUMBER, ensuring that the field can be used for numeric analysis, such as calculating average credit limits.
3. Unique Identifier for Data Integration:
    - The primary key, card_id, uniquely identifies each card. It is essential for establishing relationships with fact tables that store transactional data (e.g., FactTransactions), where each transaction can be associated with a specific card.

```
CREATE TABLE DimCard AS
SELECT
  id AS card_id,
  card_brand,
  card_type,
  CASE
    WHEN has_chip = 'YES' THEN 'Y'
    ELSE 'N'
  END AS has_chip,
  TO_NUMBER(credit_limit) AS credit_limit
FROM
  Card_Staging_Data;
```

```
INSERT INTO DimCard (card_id, card_brand, card_type, has_chip, credit_limit)
SELECT
```

```sql
    id AS card_id,
    card_brand,
    card_type,
    CASE
        WHEN has_chip = 'YES' THEN 'Y'
        ELSE 'N'
    END AS has_chip,
    TO_NUMBER(credit_limit) AS credit_limit
FROM
    Card_Staging_Data;

ALTER TABLE DIMCARD
ADD CONSTRAINT uq_dimcard UNIQUE (CARD_ID);
```



	CARD_ID	CARD_BRAND	CARD_TYPE	HAS_CHIP	CREDIT_LIMIT
1	1041	Visa	Credit	Y	11700
2	1370	Mastercard	Debit	Y	17678
3	4887	Visa	Debit (Prepaid)	Y	81
4	4055	Mastercard	Debit	Y	26231
5	2367	Visa	Credit	Y	7700
6	6029	Visa	Debit	Y	13778
7	4737	Visa	Credit	Y	10400
8	5649	Mastercard	Debit	Y	6863
9	1212	Mastercard	Debit	Y	16633
10	3845	Mastercard	Debit	Y	29391
11	4197	Discover	Credit	Y	19000
12	5650	Mastercard	Debit	Y	14541
13	212	Visa	Credit	Y	13500
14	5061	Visa	Debit	Y	1484
15	5289	Mastercard	Debit	N	31192
16	0	Amex	Credit	Y	33900
17	2274	Mastercard	Debit (Prepaid)	Y	44

The DimMerchant table is a dimension table designed to store information about merchants. It is created from raw data in the Transactions_Staging_Data table, which is likely a staging area for transactional data before it is organized and loaded into the data warehouse.

Purpose and Use of DimMerchant Table

1. Central Repository for Merchant Information:
    - The DimMerchant table consolidates essential merchant-related attributes, including merchant_id, merchant_city, merchant_state, and mcc (Merchant Category Code).
    - This setup provides a single, clean, and organized source of merchant data, making it easy to perform queries and analyses involving merchants.
2. Data Transformation and Standardization:
    - The table uses TO_CHAR(mcc) to ensure that mcc values are stored as text. This standardization is helpful if MCC codes need to be used as categorical identifiers rather than numerical values.
    - By grouping on merchant_id, merchant_city, merchant_state, and mcc, the table removes duplicate entries, ensuring that each merchant is represented only once in the table.
3. Primary Key for Data Integration:
    - The primary key, merchant_id, uniquely identifies each merchant, allowing it to be linked with fact tables (e.g., FactTransactions) that store transactional data.

```
CREATE TABLE DimMerchant AS
SELECT
  merchant_id,
  merchant_city,
  merchant_state,
  TO_CHAR(mcc) AS mcc
FROM
  Transactions_Staging_Data
GROUP BY
  merchant_id,
  merchant_city,
```

```
    merchant_state,
    mcc;


    INSERT INTO DimMerchant (merchant_id, merchant_city, merchant_state, mcc)
SELECT
    merchant_id,
    merchant_city,
    merchant_state,
    TO_CHAR(mcc) AS mcc
FROM
    Transactions_Staging_Data
GROUP BY
    merchant_id,
    merchant_city,
    merchant_state,
    mcc;

ALTER TABLE DimMerchant
ADD CONSTRAINT pk_dimmerchant PRIMARY KEY (merchant_id);
```

```sql
select * from DimMerchant;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.03 seconds

	MERCHANT_ID	MERCHANT_CITY	MERCHANT_STATE	MCC
1	60359	Green Bay	WI	8043
2	25740	Santaquin	UT	5813
3	34713	Bessemer	AL	7995
4	94511	Joliet	IL	5912
5	29415	San Rafael	CA	5921
6	28519	Chicago	IL	5661
7	1927	Lombard	IL	7538
8	83750	Dawsonville	GA	5310
9	28702	Mosheim	TN	5812
10	82529	Edgerton	WI	5921
11	42602	Little Rock	AR	5812
12	49923	Las Vegas	NV	5921
13	97873	Gloucester	VA	8021
14	71652	Oakland Gardens	NY	7349
15	57992	Round Rock	TX	6300
16	23695	West Covina	CA	5411
17	57886	Bayfield	WI	8041
18	14461	Morrisville	NC	7349
19	29748	East Bernard	TX	5912

Creating Fact Table

The FactTransactions table is a fact table in a data warehouse, designed to store transactional data. Unlike dimension tables, which store descriptive attributes, a fact table contains quantitative data (measurable events) and links to relevant dimension tables to provide context.

Purpose and Use of FactTransactions Table

1. Captures Transactional Data:

- The FactTransactions table records individual transactions, including the transaction_id, transaction_date, client_id, card_id, merchant_id, and the amount of each transaction.
- This table serves as the central location for transaction data, capturing each transaction's key details for analysis.

2. Linking Facts with Dimensions:
   - Foreign Keys (client_id, card_id, merchant_id) link this table to related dimension tables — DimCustomer, DimCard, and DimMerchant.
   - These links allow analysts to combine transactional data with descriptive attributes from the dimension tables (e.g., customer demographics, card details, and merchant locations).
   - For example, a transaction can be associated with a customer's age, the type of card used, and the merchant's category, enabling multidimensional analysis.
3. Measures for Analysis:
   - The primary measure in this table is amount, which represents the monetary value of each transaction.

```
CREATE TABLE FactTransactions (
   transaction_id NUMBER PRIMARY KEY,
   transaction_date DATE,
   client_id NUMBER,
   card_id NUMBER,
   merchant_id NUMBER,
   amount DECIMAL(10, 2),
   FOREIGN KEY (client_id) REFERENCES DimCustomer(client_id),
   FOREIGN KEY (card_id) REFERENCES DimCard(card_id),
   FOREIGN KEY (merchant_id) REFERENCES DimMerchant(merchant_id)
);
INSERT INTO FactTransactions (transaction_id, transaction_date, client_id, card_id,
merchant_id, amount)
SELECT
```

id,   -- Mapping id from Transactions_Staging_Data to transaction_id in FactTransactions

transaction_date,   -- Mapping transaction_date from Transactions_Staging_Data

client_id,   -- Mapping client_id from Transactions_Staging_Data

card_id,   -- Mapping card_id from Transactions_Staging_Data

merchant_id,   -- Mapping merchant_id from Transactions_Staging_Data

amount   -- Mapping amount from Transactions_Staging_Data

FROM

Transactions_Staging_Data

WHERE

client_id IN (SELECT client_id FROM DimCustomer) -- Ensure the client_id exists in DimCustomer

AND card_id IN (SELECT card_id FROM DimCard)   -- Ensure the card_id exists in DimCard

AND merchant_id IN (SELECT merchant_id FROM DimMerchant); -- Ensure the merchant_id exists in DimMerchant

```
select * from FactTransactions;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.032 seconds

	TRANSACTION_ID	TRANSACTION_DATE	CLIENT_ID	CARD_ID	MERCHANT_ID	AMOUNT
1	7477955	01-JAN-10	909	3719	56431	-73
2	7477958	01-JAN-10	387	4601	18563	11.07
3	7477960	01-JAN-10	1135	4977	18563	23.93
4	7477962	01-JAN-10	585	5881	18563	20.74
5	7477963	01-JAN-10	948	3376	16030	33.93
6	7477964	01-JAN-10	1074	4108	32175	15.55
7	7477967	01-JAN-10	845	5943	45149	24.42
8	7477969	01-JAN-10	1941	2030	52100	53.79
9	7477970	01-JAN-10	1385	3807	27092	160
10	7477971	01-JAN-10	1896	4272	64656	22.43
11	7477972	01-JAN-10	394	4717	1967	46.71
12	7477973	01-JAN-10	1662	4541	81833	202.12
13	7477974	01-JAN-10	1741	5571	81833	72.57
14	7477975	01-JAN-10	1863	2213	19496	88.6
15	7477976	01-JAN-10	976	3455	27601	112.19
16	7477977	01-JAN-10	1214	5508	43293	-68

Exploratory Data Analysis

1. DimCustomer Table Analysis

-- Count the total number of customers

SELECT COUNT(*) AS total_customers FROM DimCustomer;

	TOTAL_CUSTOMERS
1	2000

-- Calculate the average age of customers

SELECT AVG(age) AS avg_age FROM DimCustomer;

	AVG_AGE
1	45.3915

-- Distribution of customers by gender

SELECT gender, COUNT(*) AS count_by_gender

FROM DimCustomer

GROUP BY gender;

	GENDER	COUNT_BY_GENDER
1	Male	984
2	Female	1016

```sql
-- Distribution of customers by credit score range
SELECT
    CASE
        WHEN credit_score < 600 THEN 'Poor'
        WHEN credit_score BETWEEN 600 AND 700 THEN 'Fair'
        WHEN credit_score BETWEEN 701 AND 750 THEN 'Good'
        WHEN credit_score > 750 THEN 'Excellent'
    END AS credit_score_category,
    COUNT(*) AS count_by_score_category
FROM DimCustomer
GROUP BY
    CASE
        WHEN credit_score < 600 THEN 'Poor'
        WHEN credit_score BETWEEN 600 AND 700 THEN 'Fair'
        WHEN credit_score BETWEEN 701 AND 750 THEN 'Good'
        WHEN credit_score > 750 THEN 'Excellent'
    END;
```

	CREDIT_SCORE_CATEGORY	COUNT_BY_SCORE_CATEGORY
1	Excellent	524
2	Poor	132
3	Fair	704
4	Good	640

we have the following analysis of the **credit score distribution** among customers:

- **Excellent**: There are **524** customers with an "Excellent" credit score, indicating that these customers have a high level of creditworthiness. They likely have a credit score above 750, making them low-risk customers.

- **Good**: There are **640** customers with a "Good" credit score. These customers generally have reliable credit histories and are considered to be fairly low risk. Their credit scores likely fall between 701 and 750.
- **Fair**: The largest group is **704** customers with a "Fair" credit score. This group has a moderate level of creditworthiness, typically with scores between 600 and 700. They may be subject to higher scrutiny or interest rates for loans due to the moderate risk level.
- **Poor**: The smallest group is **132** customers with a "Poor" credit score. These customers have a credit score below 600, which may indicate higher credit risk. Financial institutions may be cautious with this group, possibly offering limited credit options.

2. DimCard Table Analysis

-- Count the total number of cards

SELECT COUNT(*) AS total_cards FROM DimCard;

	TOTAL_CARDS
1	6146

-- Distribution of cards by brand

SELECT card_brand, COUNT(*) AS count_by_brand

FROM DimCard

GROUP BY card_brand;

	CARD_BRAND	COUNT_BY_BRAND
1	Amex	402
2	Mastercard	3209
3	Visa	2326
4	Discover	209

-- Distribution of cards by type

SELECT card_type, COUNT(*) AS count_by_type

FROM DimCard

GROUP BY card_type;

	CARD_TYPE	COUNT_BY_TYPE
1	Debit	3511
2	Credit	2057
3	Debit (Prepaid)	578

-- Count of cards with and without chip

SELECT has_chip, COUNT(*) AS count_by_chip

FROM DimCard

GROUP BY has_chip;

	HAS_CHIP	COUNT_BY_CHIP
1	Y	5500
2	N	646

-- Average and maximum credit limit of cards

SELECT AVG(credit_limit) AS avg_credit_limit, MAX(credit_limit) AS max_credit_limit

FROM DimCard;

	AVG_CREDIT_LIMIT	MAX_CREDIT_LIMIT
1	14347.4939798242759518385942076147087537	151223

3. DimMerchant Table Analysis

-- Count the total number of unique merchants

SELECT COUNT(*) AS total_merchants FROM DimMerchant;

	TOTAL_MERCHANTS
1	15311

-- Distribution of merchants by state

SELECT merchant_state, COUNT(*) AS count_by_state

FROM DimMerchant

GROUP BY merchant_state;

	MERCHANT_STATE	COUNT_BY_STATE
1	OK	210
2	MN	249
3	NJ	421
4	SD	58
5	WV	72
6	AK	7
7	(null)	115
8	AL	228
9	CA	1778
10	WY	15
11	ND	20
12	WI	284
13	AR	106
14	VA	312
15	MA	227
16	NM	126
17	TN	393
18	OH	602
19	MD	290
20	NE	92
21	UT	56

```sql
-- Distribution of merchants by city (Top 10 cities with most merchants)
SELECT merchant_city, COUNT(*) AS count_by_city
FROM DimMerchant
GROUP BY merchant_city
ORDER BY count_by_city DESC
FETCH FIRST 10 ROWS ONLY;
```

	MERCHANT_CITY	COUNT_BY_CITY
1	Houston	175
2	Miami	127
3	ONLINE	115
4	Chicago	104
5	New York	98
6	Indianapolis	98
7	Atlanta	94
8	Dallas	88
9	Brooklyn	85
10	Orlando	85

```sql
-- Count of merchants by Merchant Category Code (MCC)
SELECT mcc, COUNT(*) AS count_by_mcc
FROM DimMerchant
GROUP BY mcc
ORDER BY count_by_mcc DESC;
```

	..	COUNT_BY_MCC
1	5411	1577
2	5812	1317
3	5912	998
4	4900	823
5	7538	782
6	5300	762
7	7230	761
8	5310	736
9	5813	710
10	5921	580
11	5211	555
12	7832	450
13	5651	426
14	4121	415
15	8021	364
16	6300	312
17	7349	306

4. FactTransactions Table Analysis

-- Count of total transactions

SELECT COUNT(*) AS total_transactions FROM FactTransactions;

	TOTAL_TRANSACTIONS
1	250156

-- Total and average transaction amount

SELECT SUM(amount) AS total_transaction_amount, AVG(amount) AS avg_transaction_amount

FROM FactTransactions;

	TOTAL_TRANSACTION_AMOUNT	AVG_TRANSACTION_AMOUNT
1	11020630.4	44.05503126049345208589839939877516429748

-- Transaction count by customer

SELECT client_id, COUNT(*) AS transaction_count

FROM FactTransactions

GROUP BY client_id

ORDER BY transaction_count DESC

FETCH FIRST 10 ROWS ONLY;

	CLIENT_ID	TRANSACTION_COUNT
1	1098	1040
2	909	987
3	96	914
4	1963	913
5	1776	859
6	1888	836
7	114	815
8	1696	729
9	208	693
10	285	650

-- Transaction count by card type (joining DimCard to get card_type)

SELECT dc.card_type, COUNT(ft.transaction_id) AS transaction_count

FROM FactTransactions ft

JOIN DimCard dc ON ft.card_id = dc.card_id

GROUP BY dc.card_type;

	CARD_TYPE	TRANSACTION_COUNT
1	Debit	154311
2	Debit (Prepaid)	17083
3	Credit	78762

-- Transaction count by merchant category (joining DimMerchant to get mcc)

SELECT dm.mcc, COUNT(ft.transaction_id) AS transaction_count

FROM FactTransactions ft

JOIN DimMerchant dm ON ft.merchant_id = dm.merchant_id

GROUP BY dm.mcc

ORDER BY transaction_count DESC;

	MCC	TRANSACTION_COUNT
1	5411	28654
2	5499	28187
3	5541	28111
4	5812	18402
5	5912	14696
6	5300	11636
7	4829	11430
8	4784	11146
9	4121	9544
10	5814	9401
11	5311	9264
12	7538	8896
13	5813	4662
14	4900	4564
15	5310	4524
16	5942	4083

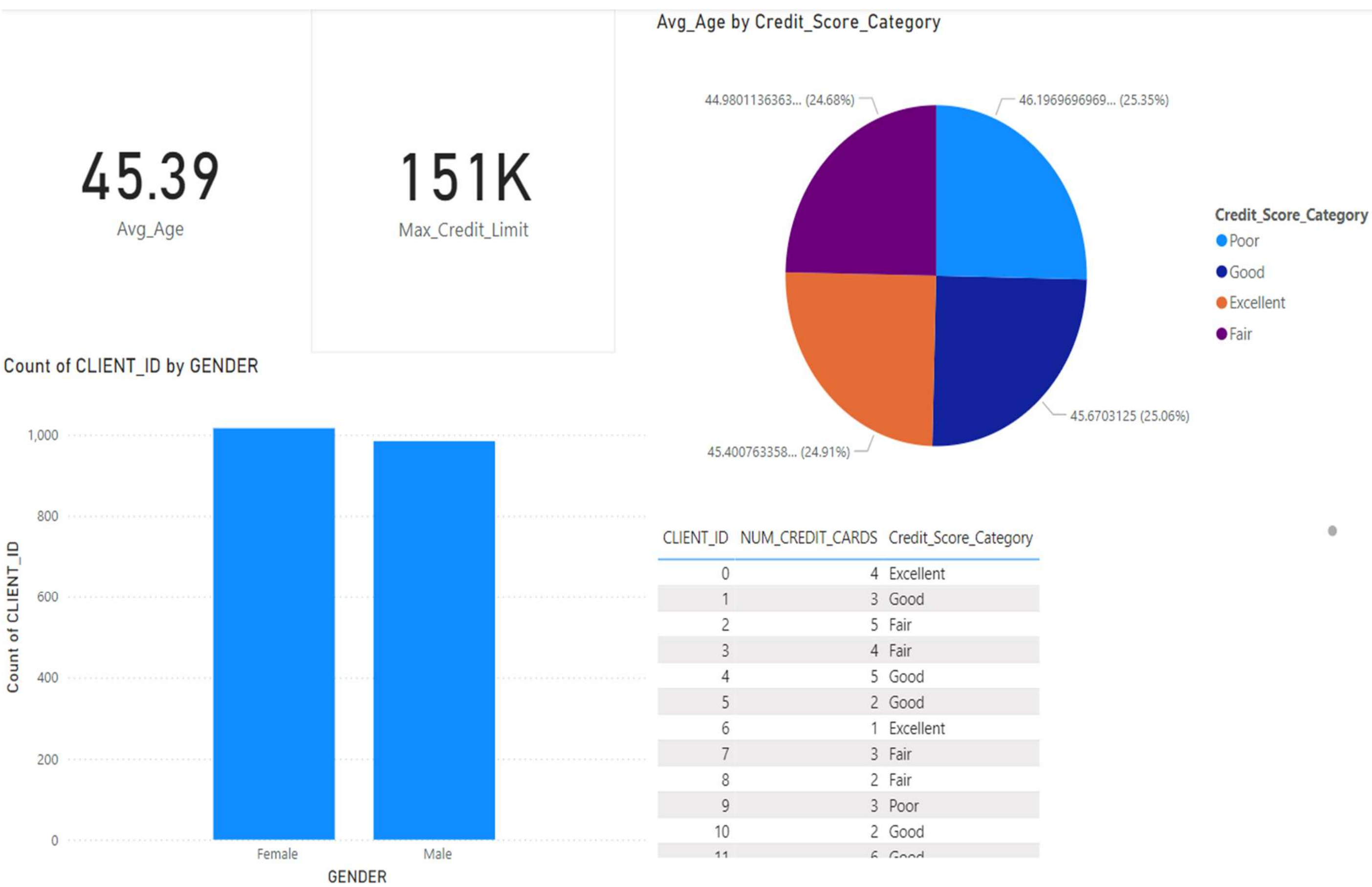
Monthly transaction amount

```
SELECT
    TO_CHAR(transaction_date, 'YYYY') AS year,
    TO_CHAR(transaction_date, 'MM') AS month,
    COUNT(transaction_id) AS monthly_transaction_count,
    SUM(amount) AS monthly_transaction_amount
FROM FactTransactions
GROUP BY TO_CHAR(transaction_date, 'YYYY'), TO_CHAR(transaction_date, 'MM')
ORDER BY year, month;
```

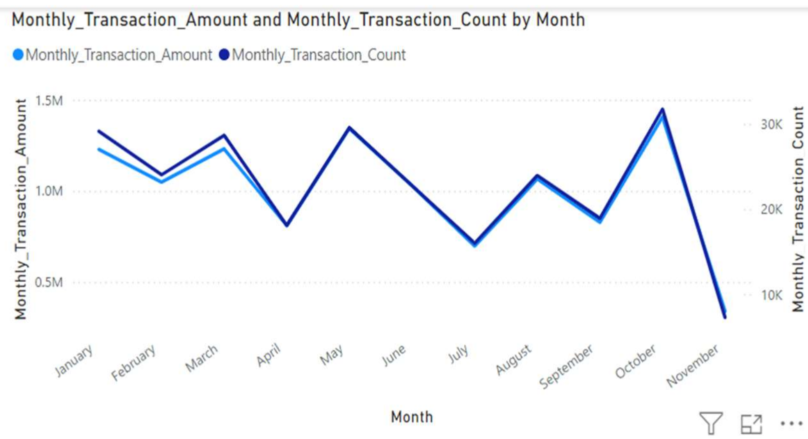
	YEAR	MONTH	MONTHLY_TRANSACTION_COUNT	MONTHLY_TRANSACTION_AMOUNT
1	2010	01	29161	1228193.34
2	2010	02	24048	1047892.35
3	2010	03	28676	1230827.46
4	2010	04	18081	808998.37
5	2010	05	29595	1342062.3
6	2010	06	22753	1028402.97
7	2010	07	15995	697741.34
8	2010	08	23934	1065552.54
9	2010	09	18908	827611.81
10	2010	10	31750	1404784.09
11	2010	11	7255	338563.83

Reports

1. Customer Demographics and Credit Profile Report

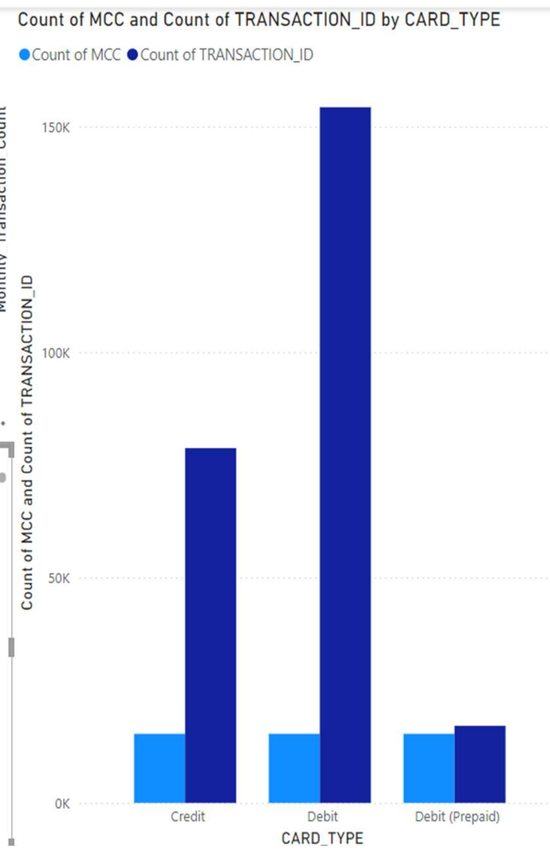


2. Transaction Summary Report



Top Customers by Transaction Amount:

CARD_ID	Monthly_Transaction_Amount	Monthly_Transaction_Count
0	6,233.68	75
1	4,962.06	97
2	7,959.19	219
3	7,158.40	90
4	1,081.93	17
5	3,033.96	57
8	4,868.22	96
9	2,851.63	117
10	2,611.48	192
11	796.55	33
12	9,289.08	68
13	3,892.76	115
14	3,882.48	122
Total	**11,020,630.40**	**250156**



Modeling and Storytelling

Credit Risk Assessment Model

This model will classify customers into different credit risk categories based on their credit score, income, credit utilization, and spending behavior. This Logistic Regression model aims to classify customers into different credit risk categories based on their financial profile and transaction behavior. Using features such as per capita income, credit score, transaction count, and average transaction amount, the model predicts risk levels categorized as Low Risk, Moderate Risk, High Risk, and Very High Risk.

The data processing steps include:

1. **Feature Engineering**: Transaction data is aggregated for each customer, calculating total spending, average transaction amount, and transaction count.
2. **Risk Category Definition**: A custom function is applied to classify each customer's risk category based on their credit score.
3. **Data Preprocessing**: Missing values in transaction-related features are filled with zeros, an imputer is used to handle any remaining NaN values, and the data is standardized.
4. **Training and Evaluation**: The model is trained using Logistic Regression, with cross-validation applied to ensure consistency.

```python

import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report

# Load data from tables
dim_customer = pd.read_csv(r"C:\Users\manvi\sqldeveloper-23.1.1.345.2114-x64\sqldeveloper\sqldeveloper\bin\DIMCUSTOMER_DATA_TABLE.csv")
fact_transactions = pd.read_csv(r"C:\Users\manvi\sqldeveloper-23.1.1.345.2114-x64\sqldeveloper\sqldeveloper\bin\FACTTRANSACTIONS_DATA_TABLE.csv")

# Feature Engineering

# 1. Aggregate transaction data per client
transaction_data = fact_transactions.groupby('CLIENT_ID').agg({
    'AMOUNT': ['sum', 'mean', 'count']
}).reset_index()
transaction_data.columns = ['CLIENT_ID', 'total_spent', 'avg_transaction_amount', 'transaction_count']

# 2. Merge transaction data with customer data
customer_data = pd.merge(dim_customer, transaction_data, on='CLIENT_ID', how='left')

# 3. Fill NaN values in transaction-related columns with 0 (for customers with no transactions)
```

```python
customer_data[['total_spent', 'avg_transaction_amount', 'transaction_count']] =
customer_data[['total_spent', 'avg_transaction_amount', 'transaction_count']].fillna(0)


# Define a risk category based on credit score
def credit_risk_category(credit_score):
    if credit_score > 750:
        return 'Low Risk'
    elif 700 <= credit_score <= 750:
        return 'Moderate Risk'
    elif 600 <= credit_score < 700:
        return 'High Risk'
    else:
        return 'Very High Risk'


# Apply risk category function to the data
customer_data['RiskCategory'] =
customer_data['CREDIT_SCORE'].apply(credit_risk_category)


# Select features and target variable
X = customer_data[['PER_CAPITA_INCOME', 'CREDIT_SCORE', 'transaction_count',
'avg_transaction_amount']]
y = customer_data['RiskCategory']


# Convert categorical target to numerical for modeling
y = pd.factorize(y)[0]


# Handle missing values in features
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

```
# Standardize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_imputed)


# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)


# Initialize and train Logistic Regression model

logreg_model = LogisticRegression(max_iter=1000, random_state=42)

logreg_model.fit(X_train, y_train)


# Cross-validation

cv_scores_logreg = cross_val_score(logreg_model, X_scaled, y, cv=5)

print(f"Cross-validation scores (Logistic Regression): {cv_scores_logreg}")

print(f"Average cross-validation score (Logistic Regression): {cv_scores_logreg.mean()}")


# Predict and evaluate

y_pred_logreg = logreg_model.predict(X_test)

print(classification_report(y_test, y_pred_logreg, target_names=['Low Risk', 'Moderate Risk', 'High Risk', 'Very High Risk']))

```

```
    C:\Users\manvi\anaconda3\Lib\site-packages\sklearn\impute\_base.py:577: UserWarning: Skipping features without any observed values: ['PER_CAPITA_INCOME']. At least one non-missing value is needed for imputation with strategy='mean'.
      warnings.warn(
```

Cross-validation scores (Logistic Regression): [0.9725 0.9875 0.97   0.985 0.9825]

Average cross-validation score (Logistic Regression): 0.9795

	precision	recall	f1-score	support
Low Risk	1.00	0.96	0.98	155
Moderate Risk	0.97	1.00	0.99	209
High Risk	0.97	0.99	0.98	198
Very High Risk	1.00	0.89	0.94	38
accuracy			0.98	600
macro avg	0.99	0.96	0.97	600
weighted avg	0.98	0.98	0.98	600

```python

```

Model Performance

The results indicate strong model performance with the following key metrics:

- **Cross-Validation Scores**: High scores (ranging from 0.97 to 0.985) indicate consistent performance across multiple validation sets.
- **Classification Metrics**: Precision, recall, and F1-scores for each risk category are high, especially for the major categories like Low Risk and Moderate Risk.

Storytelling Insight

The model effectively segments the customer base into clear risk categories, which can be highly valuable for the financial institution in determining customer eligibility for various products. The insights can help tailor financial offerings, such as providing higher credit limits for low-risk customers, or implementing risk mitigation measures for high-risk customers.

In a business context:

- **For Low and Moderate Risk Customers**: The model identifies these as reliable borrowers, and the institution can confidently extend credit or offer premium products.
- **For High and Very High Risk Customers**: The model enables targeted risk management strategies, such as higher scrutiny on loan applications or recommending financial counseling services to reduce debt levels.

This model not only strengthens risk assessment but also aligns with the institution's goal of maximizing customer satisfaction while managing risk exposure effectively.

Conclusion

The project successfully develops a comprehensive data warehousing solution tailored for financial analytics, integrating customer demographics, card details, and transaction data. By employing a dual-model approach—transactional for real-time operations and dimensional for analytical efficiency—it ensures robust data management. Through meticulous data preparation and standardization, the project delivers high-quality, enriched datasets, enabling multi-dimensional analysis for customer segmentation, fraud detection, and marketing optimization. The inclusion of data cubes further enhances the ability to identify patterns and trends across geographic and temporal dimensions.

Additionally, the credit risk assessment model demonstrates strong predictive capabilities with high precision and recall, providing actionable insights for customer risk profiling. This enables financial institutions to offer tailored products to low-risk customers and implement targeted risk mitigation for high-risk segments. The project exemplifies how data-driven strategies can transform decision-making, delivering value through enhanced customer insights, improved risk management, and optimized business strategies, ensuring alignment with organizational goals in a competitive financial landscape.

References:

1. W. H. Inmon, *Building the Data Warehouse*, Wiley & Sons, 1992.
2. W. H. Inmon, *Corporate Information Factory*, Wiley & Sons, 2005.
3. R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, Wiley & Sons, 2002.

4. R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker, *The Data Warehouse Lifecycle Toolkit*, Wiley & Sons, 2010.
5. Databricks, "Analyzing Financial Data with Spark," [Online]. Available: https://databricks.com. [Accessed: Nov. 2024].
6. Towards Data Science, "Fraud Detection in Financial Transactions," [Online]. Available: https://towardsdatascience.com. [Accessed: Nov. 2024].
7. Oracle, "SQL and PL/SQL Documentation," [Online]. Available: https://docs.oracle.com. [Accessed: Nov. 2024].
8. Microsoft, "SQL Server Documentation," [Online]. Available: https://learn.microsoft.com/sql. [Accessed: Nov. 2024].
9. PostgreSQL, "PostgreSQL Documentation," [Online]. Available: https://www.postgresql.org/docs/. [Accessed: Nov. 2024].
10. S. Ghosh and D. L. Reilly, "Credit card fraud detection with a neural-network," *Proc. 27th Annu. Hawaii Int. Conf. Syst. Sci.*, IEEE, 1994.
11. R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical Science*, vol. 17, no. 3, pp. 235–255, 2002.
12. E. W. Ngai, Y. Hu, Y. H. Wong, Y. Chen, and X. Sun, "The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature," *Decision Support Systems*, vol. 50, no. 3, pp. 559–569, 2011.