Fetch Rewards Coding Exercise - Analytics Engineer

Introduction

This documentation presents the solution for the Fetch Rewards coding exercise, which includes data modeling, SQL queries, data quality assessment, communication with stakeholders, resources used and DDL code.

Table of Contents

Note:

SQL Dialect Used: Standard SQL

The SQL queries provided in the documentation adhere to the Standard SQL dialect, which follows the ANSI SQL (American National Standards Institute SQL) standards. This dialect ensures compatibility and interoperability across various relational database management systems (RDBMS) such as MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, and others.

JSON Format Correction:

The provided JSON files exhibited an incorrect format, resulting in parsing errors. The error message encountered during parsing was:

```
Parse error on line 1:
...","topBrand":false}{"_id":{"$oid":"601c
----------------------^
Expecting 'EOF', '}', ',', ']', got '{'
```

The error was caused by missing square brackets [ ] at the beginning and end of the files and the absence of commas between individual objects. This deviation from standard JSON syntax hindered proper parsing and data interpretation.

To address this issue, the JSON files were corrected by adding square brackets to encapsulate the JSON arrays and inserting commas between objects. This adjustment ensures adherence to standard JSON structure, facilitating seamless data processing and evaluation during the assessment.

1. Data Modeling

Overview

In this section, I’ve developed a structured relational data model for Fetch Rewards using provided JSON data and crafted a logical and structural data models for Fetch Rewards, integrating Receipts, Users, Brands, Receipt Items, Items, Categories, User Rewards, and Rewards. This model efficiently organizes JSON data for structured querying and analysis in a database or data warehouse environment.

By establishing clear relationships and adhering to best practices, our model ensures data integrity, normalization, and query optimization. It serves as a robust framework for data storage, retrieval, and analysis, enabling scalability and flexibility to meet evolving business needs.

Data Model Diagram: Below are the logical and Relational Diagrams

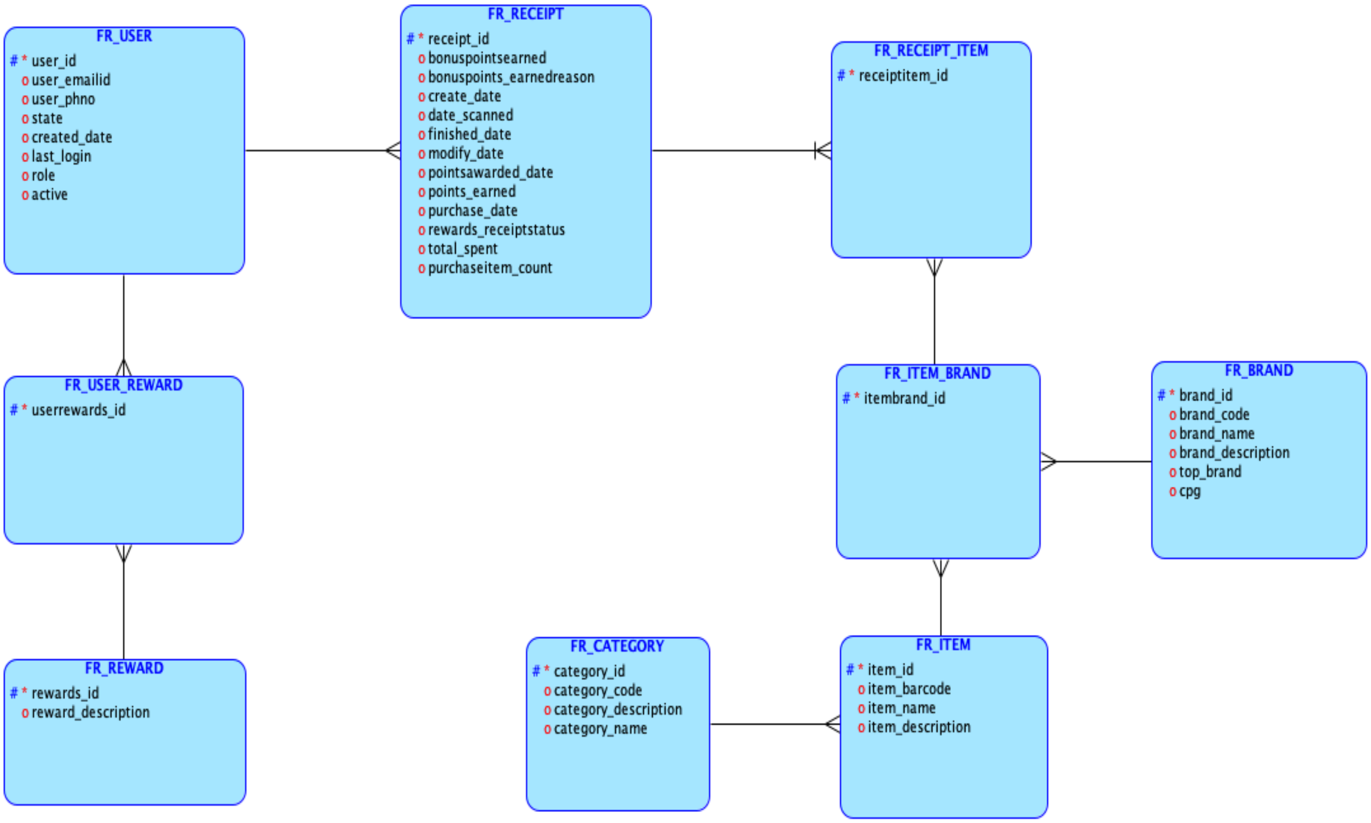

Figure 1(a). Logical Diagram of Fetch Rewards Database

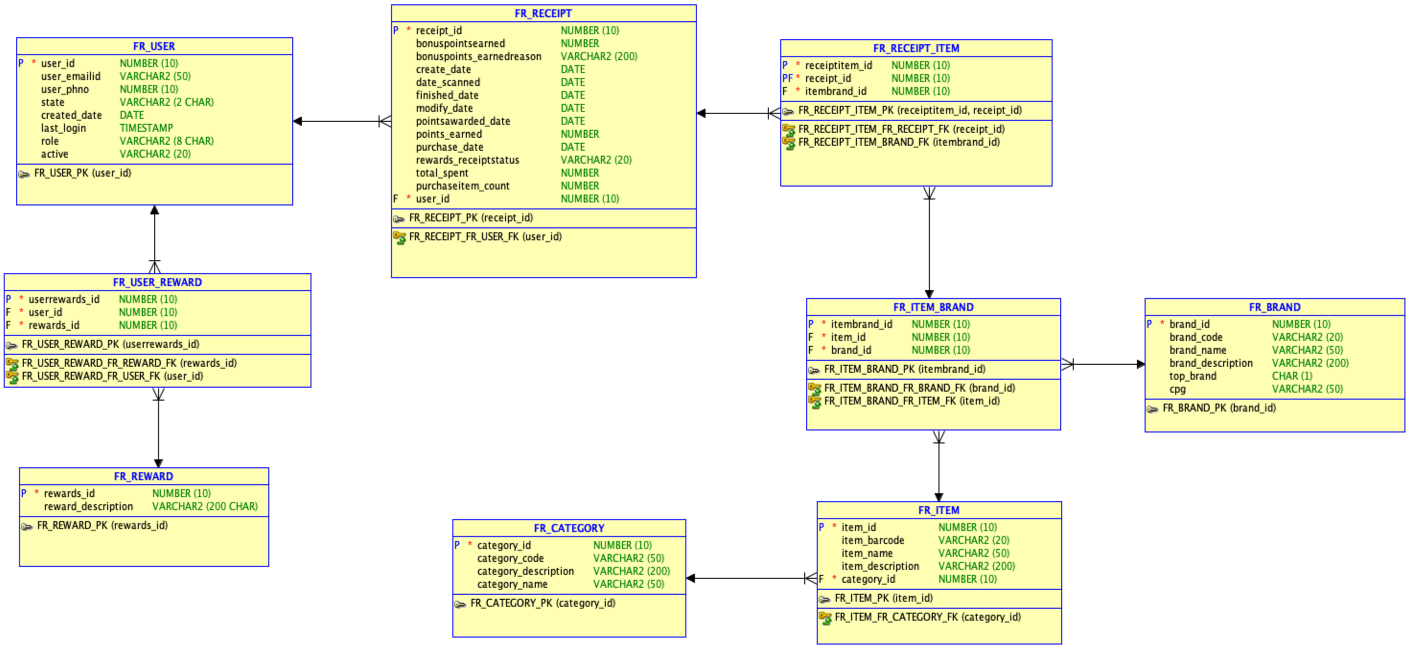
FR_USER
P * user_id NUMBER (10)
user_emailid VARCHAR2 (50)
user_phno NUMBER (10)
state VARCHAR2 (2 CHAR)
created_date DATE
last_login TIMESTAMP
role VARCHAR2 (8 CHAR)
active VARCHAR2 (20)
FR_USER_PK (user_id)
FR_RECEIPT
P * receipt_id NUMBER (10)
bonuspointsearned NUMBER
bonuspoints_earnedreason VARCHAR2 (200)
create_date DATE
date_scanned DATE
finished_date DATE
modify_date DATE
pointsawarded_date DATE
points_earned NUMBER
purchase_date DATE
rewards_receiptstatus VARCHAR2 (20)
total_spent NUMBER
purchaseitem_count NUMBER
F * user_id NUMBER (10)
FR_RECEIPT_PK (receipt_id)
FR_RECEIPT_FR_USER_FK (user_id)
FR_RECEIPT_ITEM
P * receiptitem_id NUMBER (10)
PF * receipt_id NUMBER (10)
F * itembrand_id NUMBER (10)
FR_RECEIPT_ITEM_PK (receiptitem_id, receipt_id)
FR_RECEIPT_ITEM_FR_RECEIPT_FK (receipt_id)
FR_RECEIPT_ITEM_BRAND_FK (itembrand_id)
FR_USER_REWARD
P * userrewards_id NUMBER (10)
F * user_id NUMBER (10)
F * rewards_id NUMBER (10)
FR_USER_REWARD_PK (userrewards_id)
FR_USER_REWARD_FR_REWARD_FK (rewards_id)
FR_USER_REWARD_FR_USER_FK (user_id)
FR_ITEM_BRAND
P * itembrand_id NUMBER (10)
F * item_id NUMBER (10)
F * brand_id NUMBER (10)
FR_ITEM_BRAND_PK (itembrand_id)
FR_ITEM_BRAND_FR_BRAND_FK (brand_id)
FR_ITEM_BRAND_FR_ITEM_FK (item_id)
FR_BRAND
P * brand_id NUMBER (10)
brand_code VARCHAR2 (20)
brand_name VARCHAR2 (50)
brand_description VARCHAR2 (200)
top_brand CHAR (1)
cpg VARCHAR2 (50)
FR_BRAND_PK (brand_id)
FR_REWARD
P * rewards_id NUMBER (10)
reward_description VARCHAR2 (200 CHAR)
FR_REWARD_PK (rewards_id)
FR_CATEGORY
P * category_id NUMBER (10)
category_code VARCHAR2 (50)
category_description VARCHAR2 (200)
category_name VARCHAR2 (50)
FR_CATEGORY_PK (category_id)
FR_ITEM
P * item_id NUMBER (10)
item_barcode VARCHAR2 (20)
item_name VARCHAR2 (50)
item_description VARCHAR2 (200)
F * category_id NUMBER (10)
FR_ITEM_PK (item_id)
FR_ITEM_FR_CATEGORY_FK (category_id)

Figure 1(b). Relational Diagram of Fetch Rewards

Entities and Fields

Receipt (FR_RECEIPT): To store receipts of user purchases

- *Fields*:
    - *Receipt_id (Primary Key)*
    - *bonuspointsearned*
    - *bonuspoints_earnedreason*
    - *create_date*
    - *date_scanned*
    - *finished_date*
    - *modify_date*
    - *pointsawarded_date*
    - *points_earned*
    - *purchase_date*
    - *purchaseditem_count*
    - *rewardsreceiptItemlist*
    - *rewards_receiptstatus*
    - *total_spent*
    - *user_Id (Foreign Key)*

User (FR_USER): To store user details

- *Fields*:
    - *user_id (Primary Key)*
    - *user_emailid*

    - *user_phno*
    - state
    - created_date
    - last_login
    - signup_source
    - role
    - active

Receipt Item (FR_RECEIPT_ITEM): To store receipt items list

- *Fields*:
    - *receiptitem_id (Primary Key)*
    - *item_count*
    - *receipt_id (Foreign Key)*
    - *itembrand_id (Foreign Key)*

Item Brand (FR_ITEM_BRAND): to store brand items

- *Fields*:
    - *itembrand_id (Primary Key)*
    - *brand_id (Foreign Key)*
    - *item_id (Foreign Key)*

Item (FR_ITEM): to store items

- *Fields*:
    - *item_id (Primary Key)*
    - item_barcode
    - Item_name
    - item_description
    - item_price
    - category_id *(Foreign Key)*

Brand (FR_BRAND): to store brand details

- *Fields*:
    - *brand_id (Primary Key)*
    - brand_code
    - brand_name
    - brand_description
    - cpg
    - top_brand

Category (FR_CATEGORY): to store category and its details

- *Fields*:
    - *category_id (Primary Key)*
    - category_code
    - category_name
    - Category_description

User Reward (FR_USERREWARD): to store user rewards and details

- *Fields*:
    - *userreward_id (Primary Key)*
    - *rewards_id (Foreign Key)*
    - *user_id (Foreign Key)*

Reward (FR_REWARD): to store rewards available to redeem

- *Fields*:
    - *rewards_id (Primary Key)*
    - Reward_description

Above are the entities/tables and the fields in the respective entities/tables.

As per the three (receipt, user, brand) json files provided and as per my reference to the fetch rewards website, I've created few new entities/tables, separated and established relationships between them, which I felt would be optimized database schema for fetch reward.

Entity Relationship Description

- **User - Receipt Relationship:**
  One-to-Many Relationship: One user can have many receipt, and each receipt belongs to only one user.

- **Receipt - Receipt_Item Relationship:**
  Many-to-Many Relationship: One receipt may contain many items, and one item may appear in many receipts.

- **Item - Brand Relationship:**
  Many-to-Many Relationship: One item may belong to multiple brands, and one brand may have multiple items associated with it.

- **User - Reward Relationship:**
  Many-to-Many Relationship: One user can redeem many rewards, and one reward may be applicable to many users for redemption.

- **Category - Item Relationship:**
  One-to-Many Relationship: One category can have many items, and each item belongs to only one category.

- **Additional Relationships Based on Best Practices:**
  Breaking many-to-many relationships and introducing intermediate tables as one-to-many relationships adhere to database design best practices by promoting normalization, ensuring data integrity, enabling flexibility and scalability, optimizing queries, and enhancing overall clarity and understanding of the database schema. These principles contribute to the creation of robust, efficient, and maintainable databases capable of supporting evolving business requirements.

- **Item_Brand - Receipt_Item Relationship:**
  One-to-Many Relationship: One item_brand may be associated with many receipt_items, and each receipt_item belongs to only one item_brand.

- **Brand - Item_Brand Relationship:**
  One-to-Many Relationship: One brand may have many item_brands associated with it, and each item_brand belongs to only one brand.

- **Item - Item_Brand Relationship:**
  One-to-Many Relationship: One item may have many item_brands associated with it, and each item_brand belongs to only one item.

- **User - User_Rewards Relationship:**
  One-to-Many Relationship: One user may have many user_rewards for redemption, and each user_reward belongs to one specific user.

- **Rewards - User_Reward Relationship:**
  One-to-Many Relationship: One reward may apply to many user_rewards for redemption, and each user_reward belongs to one reward.

2. SQL Queries

Business Questions

1. What are the top 5 brands by receipts scanned for the most recent month?

To answer this question, we first identify the most recent month and then count the number of receipts scanned for each brand within that month. We then rank the brands based on the number of receipts scanned and select the top 5.

Query 1:

```
with most_recent_month as(
select
   recent_year,
   max(month(date_scanned)) as recent_month

from FR_RECEIPT
WHERE recent_year= (select max(year(date_scanned)) from FR_RECEIPT)
group by recent_year
),
receipts_scanned_per_brand as(
select
   brand_name,
   count(distinct receipt_id) as receipts_scanned

from FR_RECEIPT
join most_recent_month
on year(date_scanned) = recent_year
and month(date_scanned) = recent_month
join FR_RECEIPT_ITEM
on receipt_id = receipt_id
join FR_ITEM_BRAND
on itembrand_id = itembrand_id
join FR_BRAND
on brand_id = brand_id
group by brand_name
)

select
    brand_name
from(
        select
            brand_name
            ,row_number() over(order by receipts_scanned desc) as rank
        from receipts_scanned_per_brand
)
where rank <=5
```

2. How does the ranking of the top 5 brands by receipts scanned for the recent month compare to the ranking for the previous month?

To compare the rankings, we calculate the rankings for the top 5 brands by receipts scanned for both the recent month and the previous month. We then present the rankings side by side for comparison.

Query 2:

```
with most_recent_month as(
select
    recent_year,
    max(month(date_scanned)) as recent_month

from FR_RECEIPT
WHERE recent_year= (select max(year(date_scanned)) from FR_RECEIPT)
group by recent_year
),
receipts_scanned_per_brand as(
select
    brand_name,
    count(distinct receipt_id) as receipts_scanned

from FR_RECEIPT
join most_recent_month
on year(date_scanned) = recent_year
and month(date_scanned) = recent_month
join FR_RECEIPT_ITEM
on receipt_id = receipt_id
join FR_ITEM_BRAND
on itembrand_id = itembrand_id
join FR_BRAND
on brand_id = brand_id
group by brand_name
), top_5_brand_recent_month as(
    select
        brand_name
        ,rank
    from(
            select
                brand_name
                ,row_number() over(order by receipts_scanned desc) as rank
            from receipts_scanned_per_brand
    )
    where rank <=5
),
prev_month as(
    select prev_year, prev_month
    from(
            select
                year(date_scanned) as prev_year,
                month(date_scanned) as prev_month
                      row_number() over(order by year(date_scanned) desc, month(date_scanned) desc) as
rank
            from FR_RECEIPT
            group by year(date_scanned), month(date_scanned)
    )
    where rank = 2
)
receipts_scanned_per_brand_prev_month as(
    select brand_name,
            count(distinct receipt_id) as receipts_scanned

from FR_RECEIPT
join prev_month
on year(date_scanned) = prev_year
and month(date_scanned) = prev_month
join FR_RECEIPT_ITEM
on receipt_id = receipt_id
join FR_ITEM_BRAND
on itembrand_id = itembrand_id
join FR_BRAND
on brand_id = brand_id
group by brand_name
),
brand_ranks_prev_month as(
            select
                brand_name
                ,row_number() over(order by receipts_scanned desc) as rank
            from receipts_scanned_per_brand_prev_month

)
select
    brand_name
    ,recent.rank as recent_rank
    ,prev.rank as prev_rank
from top_5_brand_recent_month recent
join brand_ranks_prev_month prev
on recent.brand_name = prev.brand_name
```

3. When considering the average spend from receipts with 'rewardsReceiptStatus' of 'Accepted' or 'Rejected', which is greater?

We calculate the average total spent for receipts with 'Accepted' and 'Rejected' statuses separately and compare the averages.

Query 3:

```
select
    rewards_receiptstatus,
    avg(total_spent) as avg_total_spent
from FR_RECEIPT
group by rewards_receiptstatus
```

4. When considering the total number of items purchased from receipts with 'rewardsReceiptStatus' of 'Accepted' or 'Rejected', which is greater?

We sum up the number of purchased items for receipts with 'Accepted' and 'Rejected' statuses separately and compare the totals.

Query 4:

```
select
    rewards_receiptstatus,
    sum(purchaseitem_count) as total_purchased_items
from FR_RECEIPT
group by rewards_receiptstatus
```

5. Which brand has the most spend among users who were created within the past 6 months?

We calculate the total spend for each brand among users created within the past 6 months and rank the brands based on total spend. The brand with the highest total spend is identified.

Query 5:

```
select
    brand_name
from(
        select
```

```
            brand_id,
            brand_name,
            total_spent,
            row_number() over(order by total_spent desc) as rank
        from(
            select
                brand_id,
                brand_name,
                sum(item_count * item_price) as total_spent

            from FR_USER
            JOIN FR_RECEIPT
            on user_id = user_id
            JOIN FR_RECEIPT_ITEM
            on receipt_id = receipt_id
            JOIN FR_ITEM_BRAND
            on itembrand_id = itembrand_id
            JOIN FR_BRAND
            on brand_id = brand_id
            JOIN FR_ITEM
            on item_id = item_id
            -- created in past 6 months
            WHERE created_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
            AND created_date <= CURDATE()

            group by brand_id,brand_name
        ) as total_spent_per_brand
    ) as ranked_brands
where rank =1
```

6. Which brand has the most transactions among users who were created within the past 6 months?

Similar to the previous query, we count the number of transactions (receipts) for each brand among users created within the past 6 months. The brand with the highest number of transactions is identified.

Query 6:

```
select
    brand_name
from(
        select
            brand_id,
            brand_name,
            total_spent,
            row_number() over(order by transaction_count desc) as rank
        from(
            select
```

```
                brand_id,
                brand_name,
                count(distinct receipt_id) as transaction_count

            from FR_USER
            JOIN FR_RECEIPT
            on user_id = user_id
            JOIN FR_RECEIPT_ITEM
            on receipt_id = receipt_id
            JOIN FR_ITEM_BRAND
            on itembrand_id = itembrand_id
            JOIN FR_BRAND
            on brand_id = brand_id
            JOIN FR_ITEM
            on item_id = item_id
            -- created in past 6 months
            WHERE created_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
            AND created_date <= CURDATE()

            group by brand_id,brand_name
        ) as total_spent_per_brand
    ) as ranked_brands
where rank =1
```

3. Data Quality Assessment

Data Quality Issues

During the examination of the provided data, several instances of incomplete data were observed across multiple tables. Incomplete data can significantly impact data analysis, decision-making processes, and overall data integrity. This document highlights the identification of incomplete data issues and provides a sample SQL query to detect such issues.

SQL Queries

User Table:

```
SELECT *
FROM User
WHERE last_login IS NULL OR state IS NULL OR Signupsource IS NULL;
```

Brand Table:

```
SELECT *
FROM Brand
WHERE brand_code IS NULL;
```

```
SELECT *
FROM Brand
WHERE top_brand IS NULL;
```

Category Table:

```
SELECT *
FROM Category
WHERE category_code IS NULL;
```

```
SELECT *
FROM Category
WHERE category_name IS NULL;
```

```
SELECT *
FROM Category
WHERE category_code IS NULL OR category_name IS NULL;
```

Receipt Table:

```
SELECT *
FROM Receipt
WHERE bonuspoints_earned IS NULL
   OR bonuspoints_earnedreason IS NULL
   OR finished_date IS NULL
   OR pointsawarded_date IS NULL
   OR total_spent IS NULL
   OR purchaseditem_count IS NULL
   OR points_earned IS NULL
   OR purchase_date IS NULL;
```

These queries will help you identify rows with missing values or inconsistencies in the data, enabling you to take corrective actions to improve the overall data quality and integrity.

Explanation of SQL Queries

The SQL queries provided are designed to identify instances of missing or incomplete data within each table of the provided dataset. By executing these queries, data analysts can pinpoint specific records that require attention and remediation to enhance data quality and integrity.

For instance, queries targeting the User table isolate rows where last_login, state, or Signupsource information is absent, facilitating subsequent data cleansing efforts. Similarly, queries applied to the Brand, Category, and Receipt tables enable the identification of records with missing values in critical fields, aiding in the detection and resolution of data quality issues across different dimensions of the dataset.

These queries serve as valuable tools for data quality assessment and form part of a broader strategy to ensure the reliability, completeness, and consistency of organizational data assets. Through systematic identification and resolution of data quality issues, organizations can enhance decision-making processes and derive greater value from their data resources.

Identifying Data Quality Issues

Data quality issues can arise from various sources such as data entry errors, inconsistencies, missing values, outliers, or discrepancies between different data sources. Below are some SQL queries to help identify potential data quality issues in the Fetch Rewards database:

Detecting Duplicates:

```
SELECT receipt_id, COUNT(*)
FROM Receipt
GROUP BY receipt_id
HAVING COUNT(*) > 1;
```

This query identifies duplicate entries in the Receipt table based on the receipt_id field.

Checking for Inconsistent Data Types:

```
SELECT *
FROM Receipt
WHERE NOT ISDATE(purchase_date);
```

This query checks if the purchase_date field contains inconsistent data types, such as non-date values.

Identifying Outliers in Numeric Fields:

```
SELECT *
FROM Receipt
WHERE total_spent < 0 OR purchaseditem_count < 0 OR points_earned < 0;
```

This query identifies receipts with negative values in numeric fields like total_spent, purchaseditem_count, or points_earned.

Exploring and Evaluating Data of Questionable Provenance

When dealing with data of questionable provenance, it's crucial to take a systematic approach to explore and evaluate its quality and reliability. Here's how you can proceed:

Data Profiling:

Perform data profiling to gain insights into the data distribution, patterns, and anomalies. Use summary statistics, histograms, and frequency distributions to identify outliers, missing values, and data inconsistencies.

Data Quality Assessment:

Assess the completeness, accuracy, consistency, and integrity of the data. Verify data against predefined business rules, domain constraints, and validation criteria. Flag records with anomalies or inconsistencies for further investigation.

Data Cleaning and Standardization:

Implement data cleaning and standardization techniques to address missing values, duplicates, outliers, and inconsistencies. Use techniques like imputation, normalization, and data transformation to improve data quality and consistency.

Data Integration and Validation:

Integrate data from multiple sources and validate it against reference data or authoritative sources. Resolve discrepancies and conflicts between different data sets by establishing data reconciliation processes and resolving data conflicts through consensus or arbitration.

Data Documentation and Metadata Management:

Document data lineage, sources, transformations, and quality assessments to establish transparency and traceability. Maintain metadata repositories and data dictionaries to provide context and insights into the data's origin, structure, and semantics.

Collaborative Review and Feedback:

Engage stakeholders, subject matter experts, and data custodians in collaborative reviews and feedback sessions to validate data assumptions, interpretations, and decisions. Leverage domain knowledge and expertise to validate data quality and relevance in the context of business objectives and requirements.

By following these practices, you can systematically explore and evaluate data of questionable provenance, improve its quality and reliability, and make informed decisions based on trustworthy and actionable insights.

4. Communication with Stakeholders

Email/Slack Message

Subject: Addressing Data Quality Concerns and Optimization Strategies

Hello Stakeholders,

Hope all is well !

As we delve into our data analysis and optimization efforts, I wanted to touch base on some crucial points regarding our data quality and optimization strategies.

Questions about the Data:
Before proceeding, it's essential to address any questions you may have about the data we're working with. Are there specific metrics or insights you're looking to gain from the data analysis? Understanding your priorities will help tailor our efforts effectively.

Discovery of Data Quality Issues:
Our team has identified several data quality issues through thorough examination and analysis. These issues primarily revolve around inconsistencies, missing values, and inaccuracies within the datasets. These discrepancies could potentially skew our analyses and hinder decision-making processes.

Information Needed to Resolve Data Quality Issues:
To effectively resolve these data quality issues, we need to delve deeper into the data sources, understand the data collection processes, and collaborate with relevant teams to ensure data integrity throughout its lifecycle. Additionally, clear documentation of data sources and transformation processes would greatly aid in maintaining data quality standards.

Additional Information for Optimization:

In order to optimize the data assets we're creating, it would be beneficial to gather insights into user behaviors, market trends, and business objectives. Understanding the end goals and desired outcomes will guide us in refining our data assets to better serve the organization's needs.

Performance and Scaling Concerns:
As we progress towards production, we anticipate potential performance and scaling concerns, especially with large volumes of data. Implementing robust data processing pipelines, utilizing scalable infrastructure, and employing efficient algorithms will be key in addressing these concerns and ensuring smooth operations in production environments.

In summary, addressing data quality issues and optimizing our data assets require collaborative efforts and a clear understanding of organizational objectives. By working closely with relevant stakeholders and leveraging appropriate tools and techniques, we can ensure that our data analysis endeavors yield valuable insights to drive informed decision-making.

Please feel free to reach out if you have any further questions or if there are specific areas you'd like to focus on. Your input and guidance are invaluable as we navigate through these data-related challenges.

Thank you for your attention to this matter.

Best regards,
Naveen Mallemala

5. Resources Used

- **JSON Validator:** Used to validate the syntax and structure of JSON files. (https://jsonlint.com/)
- **JSON Fixer:** Employed to correct any syntax errors or formatting issues within JSON files. (https://codebeautify.org/json-fixer)
- **JSON Parser:** Utilized to parse and understand the structure and content of JSON data. (https://jsongrid.com/json-parser)
- **Oracle Data Modeler:** Used for designing and configuring logical and relational data models. This tool helps in visualizing the structure of databases and relationships between tables.
- **Oracle Live SQL:** Used to run Data Definition Language (DDL) scripts, build tables, and execute SQL queries. (https://livesql.oracle.com/apex/f?p=590:1000)
- **Visual Studio Code:** Utilized for coding tasks related to the assessment. It provides a robust environment for writing code and managing project files.
- **ChatGPT:** Used for seeking clarifications on concepts, framing content, and obtaining assistance with data modeling and SQL queries.
- **Chrome Browser:** Used for exploring concepts, researching information, and clarifying doubts related to the assessment requirements.
- **Git:** Employed for version control and collaboration. Used to commit code and track changes throughout the assessment process.

- **Stack Overflow:** An online community where developers and data professionals can ask questions, share knowledge, and seek assistance with coding and technical challenges. It's a valuable resource for troubleshooting issues, learning new concepts, and exploring best practices. (https://stackoverflow.com/)
- **DB Visualizer:** A database management and visualization tool that supports various database systems. It helps in exploring database structures, executing SQL queries, and analyzing data. (https://www.dbvis.com/)

6. DDL Code for the Fetch Reward Data Model

```
-- Generated by Oracle SQL Developer Data Modeler 23.1.0.087.0806
--   at:        2024-02-17 15:10:13 EST
--   site:      Oracle Database 11g
--   type:      Oracle Database 11g



-- predefined type, no DDL - MDSYS.SDO_GEOMETRY

-- predefined type, no DDL - XMLTYPE

CREATE TABLE fr_brand (
    brand_id          NUMBER(10) NOT NULL,
    brand_code        VARCHAR2(20),
    brand_name        VARCHAR2(50),
    brand_description VARCHAR2(200),
    top_brand         CHAR(1),
    cpg               VARCHAR2(50)
);

COMMENT ON COLUMN fr_brand.brand_id IS
    'unique identifier for brand table.';

COMMENT ON COLUMN fr_brand.brand_code IS
    'String that corresponds with the brand column in a partner product file.';

COMMENT ON COLUMN fr_brand.brand_name IS
    'this is brand name.';

COMMENT ON COLUMN fr_brand.brand_description IS
    'description for the brand and it''s respective products.';

COMMENT ON COLUMN fr_brand.top_brand IS
    'Boolean indicator for whether the brand should be featured as a ''top brand''.';
```

```
COMMENT ON COLUMN fr_brand.cpg IS
    'reference to CPG collection.
';

ALTER TABLE fr_brand ADD CONSTRAINT fr_brand_pk PRIMARY KEY ( brand_id );

CREATE TABLE fr_category (
    category_id          NUMBER(10) NOT NULL,
    category_code        VARCHAR2(50),
    category_description VARCHAR2(200),
    category_name        VARCHAR2(50)
);

COMMENT ON COLUMN fr_category.category_id IS
    'unique identifier for category.';

COMMENT ON COLUMN fr_category.category_code IS
    'code for category.';

COMMENT ON COLUMN fr_category.category_description IS
    'description for category.';

COMMENT ON COLUMN fr_category.category_name IS
    'name of the category.';

ALTER TABLE fr_category ADD CONSTRAINT fr_category_pk PRIMARY KEY ( category_id );

CREATE TABLE fr_item (
    item_id          NUMBER(10) NOT NULL,
    item_barcode     VARCHAR2(20),
    item_name        VARCHAR2(50),
    item_description VARCHAR2(200),
    category_id      NUMBER(10) NOT NULL
);

COMMENT ON COLUMN fr_item.item_id IS
    'unique identifier for the item.';

COMMENT ON COLUMN fr_item.item_barcode IS
    'the barcode on the item.';

COMMENT ON COLUMN fr_item.item_name IS
    'name of the item.';

COMMENT ON COLUMN fr_item.item_description IS
    'description of the item.';

COMMENT ON COLUMN fr_item.category_id IS
    'category id foreign key.';
```

```
ALTER TABLE fr_item ADD CONSTRAINT fr_item_pk PRIMARY KEY ( item_id );

CREATE TABLE fr_item_brand (
    itembrand_id NUMBER(10) NOT NULL,
    item_id      NUMBER(10) NOT NULL,
    brand_id     NUMBER(10) NOT NULL
);

COMMENT ON COLUMN fr_item_brand.itembrand_id IS
    'unique identifier for item brand.';

COMMENT ON COLUMN fr_item_brand.item_id IS
    'item id foreign key.';

COMMENT ON COLUMN fr_item_brand.brand_id IS
    'brand id foreign key.';

ALTER TABLE fr_item_brand ADD CONSTRAINT fr_item_brand_pk PRIMARY KEY ( itembrand_id );

CREATE TABLE fr_receipt (
    receipt_id                NUMBER(10) NOT NULL,
    bonuspointsearned         NUMBER,
    bonuspoints_earnedreason VARCHAR2(200),
    create_date               DATE,
    date_scanned              DATE,
    finished_date             DATE,
    modify_date               DATE,
    pointsawarded_date        DATE,
    points_earned             NUMBER,
    purchase_date             DATE,
    rewards_receiptstatus     VARCHAR2(20),
    total_spent               NUMBER,
    purchaseitem_count        NUMBER,
    user_id                   NUMBER(10) NOT NULL
);

COMMENT ON COLUMN fr_receipt.receipt_id IS
    'Unique id for receipts table.';

COMMENT ON COLUMN fr_receipt.bonuspointsearned IS
    'bonus points earned for this purchase.';

COMMENT ON COLUMN fr_receipt.bonuspoints_earnedreason IS
    'reason for earning bonus points.';

COMMENT ON COLUMN fr_receipt.create_date IS
    'Date on which this event is created.';
```

```
COMMENT ON COLUMN fr_receipt.date_scanned IS
    'Date that the user scanned this receipt.';

COMMENT ON COLUMN fr_receipt.finished_date IS
    'Date on which receipt finished processing.';

COMMENT ON COLUMN fr_receipt.modify_date IS
    'The date the event was modified.';

COMMENT ON COLUMN fr_receipt.pointsawarded_date IS
    'The date we awarded points for the transaction.';

COMMENT ON COLUMN fr_receipt.points_earned IS
    'The number of points earned for the receipt.';

COMMENT ON COLUMN fr_receipt.purchase_date IS
    'the date of the purchase.';

COMMENT ON COLUMN fr_receipt.rewards_receiptstatus IS
    'status of the receipt through receipt validation and processing.';

COMMENT ON COLUMN fr_receipt.total_spent IS
    ' The total amount on the receipt.';

COMMENT ON COLUMN fr_receipt.purchaseitem_count IS
    'Count of number of items on the receipt.';

COMMENT ON COLUMN fr_receipt.user_id IS
    'user_id foreign key.';

ALTER TABLE fr_receipt ADD CONSTRAINT fr_receipt_pk PRIMARY KEY ( receipt_id );

CREATE TABLE fr_receipt_item (
    receiptitem_id NUMBER(10) NOT NULL,
    receipt_id     NUMBER(10) NOT NULL,
    itembrand_id   NUMBER(10) NOT NULL
);

COMMENT ON COLUMN fr_receipt_item.receiptitem_id IS
    'unique identifier for receipt item.';

COMMENT ON COLUMN fr_receipt_item.receipt_id IS
    'receipt_id foreign key.';

COMMENT ON COLUMN fr_receipt_item.itembrand_id IS
    'item brand foreign key.';

ALTER TABLE fr_receipt_item ADD CONSTRAINT fr_receipt_item_pk PRIMARY KEY ( receiptitem_id,
                                                                            receipt_id );
```

```
CREATE TABLE fr_reward (
    rewards_id         NUMBER(10) NOT NULL,
    reward_description VARCHAR2(200 CHAR)
);

COMMENT ON COLUMN fr_reward.rewards_id IS
    'unique identifier for rewards.';

COMMENT ON COLUMN fr_reward.reward_description IS
    'description of the reward.';

ALTER TABLE fr_reward ADD CONSTRAINT fr_reward_pk PRIMARY KEY ( rewards_id );

CREATE TABLE fr_user (
    user_id      NUMBER(10) NOT NULL,
    user_emailid VARCHAR2(50),
    user_phno    NUMBER(10),
    state        VARCHAR2(2 CHAR),
    created_date DATE,
    last_login   TIMESTAMP,
    role         VARCHAR2(8 CHAR),
    active       VARCHAR2(20)
);

COMMENT ON COLUMN fr_user.user_id IS
    'This is unique id for user table.';

COMMENT ON COLUMN fr_user.user_emailid IS
    'email id of user.';

COMMENT ON COLUMN fr_user.user_phno IS
    'phone number of the user.';

COMMENT ON COLUMN fr_user.state IS
    'This is state column.';

COMMENT ON COLUMN fr_user.created_date IS
    'This is user account created date.';

COMMENT ON COLUMN fr_user.last_login IS
    'last date and time user loggedin to account.';

COMMENT ON COLUMN fr_user.role IS
    'This is role of user, which is constant "CONSUMER"';

COMMENT ON COLUMN fr_user.active IS
    'This column is to indicate if a user is active or not.';
```

```
ALTER TABLE fr_user ADD CONSTRAINT fr_user_pk PRIMARY KEY ( user_id );

CREATE TABLE fr_user_reward (
    userrewards_id NUMBER(10) NOT NULL,
    user_id        NUMBER(10) NOT NULL,
    rewards_id     NUMBER(10) NOT NULL
);

COMMENT ON COLUMN fr_user_reward.userrewards_id IS
    'unique identifier for user rewards.';

COMMENT ON COLUMN fr_user_reward.user_id IS
    'user_id foreign key.';

COMMENT ON COLUMN fr_user_reward.rewards_id IS
    'reward_id foreign key.';

ALTER TABLE fr_user_reward ADD CONSTRAINT fr_user_reward_pk PRIMARY KEY ( userrewards_id );

ALTER TABLE fr_item_brand
    ADD CONSTRAINT fr_item_brand_fr_brand_fk FOREIGN KEY ( brand_id )
        REFERENCES fr_brand ( brand_id );

ALTER TABLE fr_item_brand
    ADD CONSTRAINT fr_item_brand_fr_item_fk FOREIGN KEY ( item_id )
        REFERENCES fr_item ( item_id );

ALTER TABLE fr_item
    ADD CONSTRAINT fr_item_fr_category_fk FOREIGN KEY ( category_id )
        REFERENCES fr_category ( category_id );

ALTER TABLE fr_receipt
    ADD CONSTRAINT fr_receipt_fr_user_fk FOREIGN KEY ( user_id )
        REFERENCES fr_user ( user_id );

ALTER TABLE fr_receipt_item
    ADD CONSTRAINT fr_receipt_item_brand_fk FOREIGN KEY ( itembrand_id )
        REFERENCES fr_item_brand ( itembrand_id );

ALTER TABLE fr_receipt_item
    ADD CONSTRAINT fr_receipt_item_fr_receipt_fk FOREIGN KEY ( receipt_id )
        REFERENCES fr_receipt ( receipt_id );

ALTER TABLE fr_user_reward
    ADD CONSTRAINT fr_user_reward_fr_reward_fk FOREIGN KEY ( rewards_id )
        REFERENCES fr_reward ( rewards_id );

ALTER TABLE fr_user_reward
    ADD CONSTRAINT fr_user_reward_fr_user_fk FOREIGN KEY ( user_id )
```

```
        REFERENCES fr_user ( user_id );



-- Oracle SQL Developer Data Modeler Summary Report: 
-- 
-- CREATE TABLE                             9
-- CREATE INDEX                             0
-- ALTER TABLE                             17
-- CREATE VIEW                              0
-- ALTER VIEW                               0
-- CREATE PACKAGE                           0
-- CREATE PACKAGE BODY                      0
-- CREATE PROCEDURE                         0
-- CREATE FUNCTION                          0
-- CREATE TRIGGER                           0
-- ALTER TRIGGER                            0
-- CREATE COLLECTION TYPE                   0
-- CREATE STRUCTURED TYPE                   0
-- CREATE STRUCTURED TYPE BODY              0
-- CREATE CLUSTER                           0
-- CREATE CONTEXT                           0
-- CREATE DATABASE                          0
-- CREATE DIMENSION                         0
-- CREATE DIRECTORY                         0
-- CREATE DISK GROUP                        0
-- CREATE ROLE                              0
-- CREATE ROLLBACK SEGMENT                  0
-- CREATE SEQUENCE                          0
-- CREATE MATERIALIZED VIEW                 0
-- CREATE MATERIALIZED VIEW LOG             0
-- CREATE SYNONYM                           0
-- CREATE TABLESPACE                        0
-- CREATE USER                              0
-- 
-- DROP TABLESPACE                          0
-- DROP DATABASE                            0
-- 
-- REDACTION POLICY                         0
-- 
-- ORDS DROP SCHEMA                         0
-- ORDS ENABLE SCHEMA                       0
-- ORDS ENABLE OBJECT                       0
-- 
-- ERRORS                                   0
-- WARNINGS                                 0
```