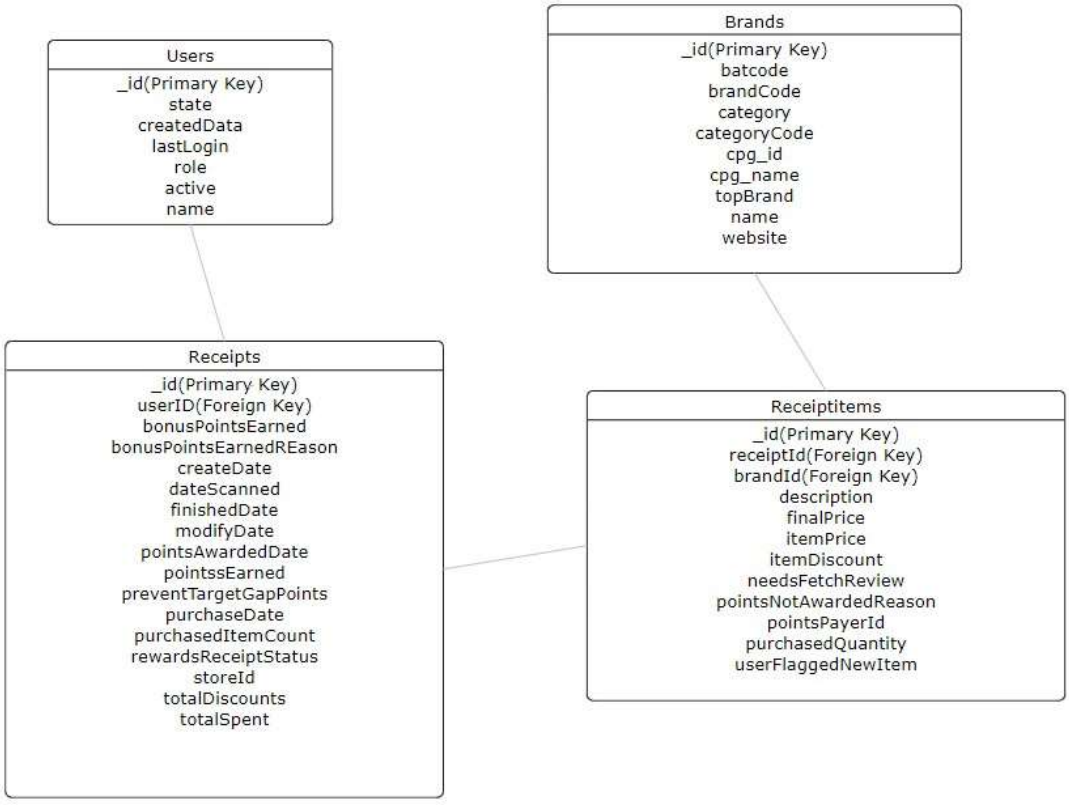
Fetch Rewards Assessment

First: Structured relational data model



Relationships Summary:

Users to Receipts: One-to-Many (Users._id -> Receipts.userId)
Receipts to ReceiptItems: One-to-Many (Receipts._id -> ReceiptItems.receiptId)
Brands to ReceiptItems: One-to-Many (Brands._id -> ReceiptItems.brandId)

Second: SQL Queries

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

```
SELECT b.name, COUNT(*) AS receipt_count
FROM Receipts r
JOIN ReceiptItems ri ON r._id = ri.receiptId
JOIN Brands b ON ri.brandId = b._id
WHERE r.dateScanned >= DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '1 month'
  AND r.dateScanned < DATE_TRUNC('month', CURRENT_DATE)
GROUP BY b.name
ORDER BY receipt_count DESC
LIMIT 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?**

```
WITH recent_month AS (
   SELECT b.name, COUNT(*) AS receipt_count
   FROM Receipts r
   JOIN ReceiptItems ri ON r._id = ri.receiptId
   JOIN Brands b ON ri.brandId = b._id
   WHERE r.dateScanned >= DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '1 month'
     AND r.dateScanned < DATE_TRUNC('month', CURRENT_DATE)
   GROUP BY b.name
),
previous_month AS (
   SELECT b.name, COUNT(*) AS receipt_count
   FROM Receipts r
   JOIN ReceiptItems ri ON r._id = ri.receiptId
   JOIN Brands b ON ri.brandId = b._id
```

```
  WHERE r.dateScanned >= DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '2 months'
   AND r.dateScanned < DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '1 month'
  GROUP BY b.name
)
SELECT recent_month.name, recent_month.receipt_count AS recent_receipt_count,
previous_month.receipt_count AS previous_receipt_count
FROM recent_month
LEFT JOIN previous_month ON recent_month.name = previous_month.name
ORDER BY recent_receipt_count DESC
LIMIT 5;
```

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

```
SELECT rewardsReceiptStatus, AVG(totalSpent) AS avg_spent
FROM Receipts
WHERE rewardsReceiptStatus IN ('Accepted', 'Rejected')
GROUP BY rewardsReceiptStatus
ORDER BY avg_spent DESC;
```

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

```
SELECT b.name, SUM(ri.finalPrice * ri.purchasedQuantity) AS total_spend
FROM Users u
JOIN Receipts r ON u._id = r.userId
JOIN ReceiptItems ri ON r._id = ri.receiptId
JOIN Brands b ON ri.brandId = b._id
WHERE u.createdDate >= CURRENT_DATE - INTERVAL '6 months'
GROUP BY b.name
ORDER BY total_spend DESC
LIMIT 1;
```

Third: Data Quality Issues

I would love to focus on the below four key aspects when checking for data quality issues.

- Data Completeness: Checking for missing values or incomplete records.
- Data Consistency: Ensuring consistent formats and values across related fields.
- Data Accuracy: Verifying that data entries conform to expected ranges and formats.
- Data Redundancy: Identifying duplicate records or fields.

Based on my analysis, here are some data quality issues found in each table of our model:

SQL (Method 1):

We can identify the potential issues in our data by writing simple SQL queries like below to find missing fields, inconsistent data types, duplicate records, invalid data ranges and foreign key validations etc.

Users Data Quality Issues:

1. Missing values in createdAt, lastLogin and state.

```
SELECT
  COUNT(*) AS total_users,
  COUNT(userId) - COUNT(createdAt) AS missing_createdAt,
  COUNT(userId) - COUNT(lastLogin) AS missing_lastLogin,
  COUNT(userId) - COUNT(state) AS missing_state
FROM users;
```

2. Inconsistent state codes.

```
SELECT DISTINCT state

FROM users

WHERE state NOT REGEXP '^[A-Z]{2}$';
```

3. Invalid date formats.

```
SELECT userId, createdAt, lastLogin
FROM users
WHERE createdAt IS NULL OR lastLogin IS NULL;
```

Receipts Data Quality Issues:

1. Missing values in userId, dateScanned, totalSpent and *bonusPointsEarnedReason*.

```
SELECT
  COUNT(*) AS total_receipts,
  COUNT(receiptId) - COUNT(userId) AS missing_userId,
  COUNT(receiptId) - COUNT(dateScanned) AS missing_dateScanned,
  COUNT(receiptId) - COUNT(totalSpent) AS missing_totalSpent
  COUNT(receiptId) - COUNT(bonusPointsEarnedReason)  AS missing_
MissingBonusPointsEarnedReason
FROM receipts;
```

2. Inconsistent rewardsReceiptStatus, totalSpent values.

```
SELECT DISTINCT rewardsReceiptStatus

FROM receipts;


SELECT

  COUNT(*) AS InvalidTotalSpent

FROM Receipts

WHERE totalSpent != '^[0-9]+(\.[0-9]{1,2})?$';
```

3. Negative totalSpent values.

```
SELECT receiptId, totalSpent

FROM receipts

WHERE totalSpent < 0;
```

4. Check for duplicate records.

```
SELECT
  userId,
  COUNT(*) AS DuplicateCount
FROM Receipts
GROUP BY userId
HAVING COUNT(*) > 1;
```

5. Check for foreign key violations.

```
-- Check userId in Receipts
SELECT
  r.userId
FROM Receipts r
LEFT JOIN Users u ON r.userId = u._id
WHERE u._id IS NULL;
```

Brands Data Quality Issues:

1. Missing values in brand fields.

```
SELECT

  COUNT(*) AS total_brands,

  COUNT(brandId) - COUNT(name) AS missing_name

FROM brands;
```

2. Inconsistent brand names.

```
SELECT DISTINCT name
FROM brands
WHERE name IS NULL OR name = '';
```

ReceiptItems Data Quality Issues:

1. Missing values in priceAfterCoupon and priceAfterDiscount.

```
SELECT
  COUNT(*) AS total_items,
  COUNT(itemId) - COUNT(priceAfterCoupon) AS
missing_priceAfterCoupon,
  COUNT(itemId) - COUNT(priceAfterDiscount) AS
missing_priceAfterDiscount
FROM receipt_items;

SELECT
  COUNT(*) AS MissingPointsNotAwardedReason
FROM ReceiptItems
WHERE pointsNotAwardedReason IS NULL;
```

2. Negative values in price fields.

```
SELECT itemId, priceAfterCoupon, priceAfterDiscount
FROM receipt_items
WHERE priceAfterCoupon < 0 OR priceAfterDiscount < 0;
```

3. Check for foreign key violations.

```
-- Check brandId in ReceiptItems

SELECT

  ri.brandId

FROM ReceiptItems ri

LEFT JOIN Brands b ON ri.brandId = b._id

WHERE b._id IS NULL;
```

Python Script (Method 2):

Another alternative approach is to write a Python script for common problems such as missing values, duplicates, and inconsistent data types. We will be using pandas library for this method. Here is a sample Python script to perform some of these checks:

```
import pandas as pd

import json


# Load JSON data

with open('receipts.json') as f:

  receipts = json.load(f)

with open('users.json') as f:

  users = json.load(f)

with open('brands.json') as f:

  brands = json.load(f)
```

```
# Convert JSON to DataFrames
df_receipts = pd.json_normalize(receipts)
df_users = pd.json_normalize(users)
df_brands = pd.json_normalize(brands)

# Check for missing values
missing_values = {
   'receipts': df_receipts.isnull().sum(),
   'users': df_users.isnull().sum(),
   'brands': df_brands.isnull().sum()
}

# Check for duplicates
duplicates = {
   'receipts': df_receipts.duplicated().sum(),
   'users': df_users.duplicated().sum(),
   'brands': df_brands.duplicated().sum()
}

# Check for inconsistent data types
data_types = {
   'receipts': df_receipts.dtypes,
   'users': df_users.dtypes,
   'brands': df_brands.dtypes
}
# Print results
print("Missing Values:\n", missing_values)
print("\nDuplicates:\n", duplicates)
print("\nData Types:\n", data_types)
```

Fourth: Communication with stakeholders

Subject: Clarification Needed on Data Quality and Optimization Strategy

Dear [Recipient Name],

I hope this message finds you well. I am writing to discuss some findings from our recent data review and to seek clarification on several points to ensure we can optimize our data assets effectively.

Key Questions and Findings:

1. **Questions about the Data:**
    - Are there any specific data validation rules we should be aware of for fields like **state**, **totalSpent**, and **rewardsReceiptStatus**?
    - Are there constraints on acceptable values for brand names and receipt statuses?
2. **Discovery of Data Quality Issues:**
    - We identified missing values, inconsistent state codes, invalid date formats, negative amounts in **totalSpent**, and invalid brand names. These issues were discovered through a series of SQL and Python checks on the datasets.
3. **Resolution of Data Quality Issues:**
    - To address these issues, we need to know the business rules governing acceptable values. For example, should all **state** codes follow the standard two-letter format? What are the acceptable values for **rewardsReceiptStatus**?
4. **Additional Information Needed:**
    - We require a comprehensive data dictionary outlining the intended use and constraints for each field. This will help us understand the business context and ensure data consistency.
    - Any historical changes in data validation rules would also be useful to understand deviations in the data.
5. **Performance and Scaling Concerns:**
    - In production, we anticipate challenges with data processing speed and storage scalability, especially as data volume grows. To address these, we plan to implement efficient indexing strategies, utilize data partitioning, and possibly employ distributed processing systems like Apache Spark for handling large datasets.

Next Steps:

To move forward, we recommend a meeting to discuss these questions and obtain the necessary details. This will help us ensure data integrity and optimize our data models for better business insights.

Please let us know your availability for a meeting next week.

Best regards,

Chanikya