## **Section 1: Structured Relational Data Model**

### 

### **Relational Schema Diagram**

The structured relational data model consists of four key tables:

## **Objective**

The original data consists of three JSON files (**Receipts, Users, Brands**) with unstructured, nested information. To optimize for **efficient querying and analysis**, I have designed a **structured relational model** that ensures **data integrity, performance, and scalability**.

## **Relational Schema Design**

The structured relational model consists of **four key tables**, each serving a distinct role in organizing the data.

### **1. Users Table (users)**

Stores user information and account details.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| user\_id (PK) | VARCHAR | Unique identifier for the user |
| state | VARCHAR | User's state location |
| created\_date | TIMESTAMP | Timestamp of user account creation |
| last\_login | TIMESTAMP | Timestamp of the most recent login |
| role | VARCHAR | Role of the user (e.g., CONSUMER) |
| active | BOOLEAN | Indicates whether the user is active |

### **2. Receipts Table (receipts)**

Stores scanned receipts, purchase transactions, and reward statuses.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| receipt\_id (PK) | VARCHAR | Unique identifier for the receipt |
| user\_id (FK) | VARCHAR | Foreign key referencing users.user\_id |
| purchase\_date | TIMESTAMP | Date of the purchase |
| date\_scanned | TIMESTAMP | Date when the receipt was scanned |
| total\_spent | DECIMAL(10,2) | Total amount spent on the receipt |
| purchased\_item\_count | INTEGER | Number of items purchased |
| rewards\_receipt\_status | VARCHAR | Status of the receipt (e.g., ACCEPTED, REJECTED) |
| bonus\_points\_earned | INTEGER | Number of bonus points awarded |
| points\_earned | INTEGER | Points earned for the receipt |

### **3. Receipt Items Table (receipt\_items)**

Stores details of individual items purchased in a receipt.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| item\_id (PK) | VARCHAR | Unique identifier for the purchased item |
| receipt\_id (FK) | VARCHAR | Foreign key referencing receipts.receipt\_id |
| brand\_id (FK) | VARCHAR | Foreign key referencing brands.brand\_id |
| barcode | VARCHAR | Product barcode |
| quantity | INTEGER | Quantity of the item purchased |
| price | DECIMAL(10,2) | Price per unit of the item |

### **4. Brands Table (brands)**

Stores brand and product category details.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| brand\_id (PK) | VARCHAR | Unique identifier for the brand |
| name | VARCHAR | Brand name |
| barcode | VARCHAR | Barcode associated with the brand |
| brand\_code | VARCHAR | Brand code (external partner reference) |
| category | VARCHAR | Category of the brand’s products |
| top\_brand | BOOLEAN | Indicator if the brand is a 'top brand' |

## **Entity-Relationship (ER) Diagram**

To visualize the structured data model, below is an **Entity-Relationship (ER) diagram**, which shows **primary keys (PKs), foreign keys (FKs), and relationships** between tables.

**Schema Visualization:**

USERS (user\_id) ────┐

│

▼

RECEIPTS (receipt\_id, user\_id) ────┐

│

▼

RECEIPT\_ITEMS (item\_id, receipt\_id, brand\_id) ────► BRANDS (brand\_id)

### **Key Relationships:**

1. **Users → Receipts (1:M)**
   * A **user** can have multiple **receipts**, but each receipt belongs to only one **user**.
2. **Receipts → Receipt Items (1:M)**
   * A **receipt** can contain multiple **items**, but each **item** is linked to a single **receipt**.
3. **Receipt Items → Brands (M:1)**
   * Each **purchased item** is linked to a **brand**, but a **brand** can be associated with multiple **items**.

## **Performance & Scalability Considerations**

🔹 **Indexes on purchase\_date, user\_id, and brand\_id** for efficient filtering.  
🔹 **Partitioning receipts table by purchase\_date** to speed up retrieval of recent transactions.  
🔹 **Pre-aggregated summary tables for frequent reports** (e.g., top brands by month).  
🔹 **Caching common queries** (e.g., top 5 brands) to reduce computational overhead.

## **Why This Model?**

✅ **Eliminates nested JSON complexity**, making queries **faster and easier to maintain**.  
✅ **Ensures referential integrity** with **Foreign Key (FK) constraints**.  
✅ **Optimized for analytical queries** (e.g., trends, customer behavior, brand performance).

### **Final Thoughts**

This relational model transforms **unstructured JSON** into a **clean, structured, and scalable** data warehouse. Let me know if you need any refinements or enhancements!

## **Section 2: SQL Queries to Answer Business Questions**

### **1. Top 5 Brands by Receipts Scanned for the Most Recent Month**

sql

CopyEdit

WITH RecentMonth AS (

SELECT MAX(DATE\_TRUNC('month', purchase\_date)) AS latest\_month FROM receipts

)

SELECT b.name AS brand\_name, COUNT(r.receipt\_id) AS receipt\_count

FROM receipts r

JOIN receipt\_items ri ON r.receipt\_id = ri.receipt\_id

JOIN brands b ON ri.brand\_id = b.brand\_id

WHERE DATE\_TRUNC('month', r.purchase\_date) = (SELECT latest\_month FROM RecentMonth)

GROUP BY b.name

ORDER BY receipt\_count DESC

LIMIT 5;

### **2. Comparing the Top 5 Brands from the Recent Month vs Previous Month**

sql

CopyEdit

WITH MonthlyRankings AS (

SELECT

b.name AS brand\_name,

DATE\_TRUNC('month', r.purchase\_date) AS purchase\_month,

COUNT(r.receipt\_id) AS receipt\_count,

RANK() OVER (PARTITION BY DATE\_TRUNC('month', r.purchase\_date) ORDER BY COUNT(r.receipt\_id) DESC) AS rank

FROM receipts r

JOIN receipt\_items ri ON r.receipt\_id = ri.receipt\_id

JOIN brands b ON ri.brand\_id = b.brand\_id

WHERE DATE\_TRUNC('month', r.purchase\_date) >= DATE\_TRUNC('month', CURRENT\_DATE) - INTERVAL '1 month'

GROUP BY b.name, purchase\_month

)

SELECT

m1.brand\_name,

m1.receipt\_count AS recent\_month\_count,

m1.rank AS recent\_month\_rank,

m2.receipt\_count AS previous\_month\_count,

m2.rank AS previous\_month\_rank

FROM MonthlyRankings m1

LEFT JOIN MonthlyRankings m2

ON m1.brand\_name = m2.brand\_name

AND m1.purchase\_month = DATE\_TRUNC('month', CURRENT\_DATE)

AND m2.purchase\_month = DATE\_TRUNC('month', CURRENT\_DATE) - INTERVAL '1 month'

WHERE m1.purchase\_month = DATE\_TRUNC('month', CURRENT\_DATE);

### **3. Comparing Average Spend Between Accepted vs. Rejected Receipts**

sql

SELECT

rewards\_receipt\_status,

AVG(total\_spent) AS avg\_spend

FROM receipts

WHERE rewards\_receipt\_status IN ('Accepted', 'Rejected')

GROUP BY rewards\_receipt\_status;

**Interpretation:**

* The result will show whether **Accepted** or **Rejected** receipts have a higher average spend.

### **4. Comparing Total Items Purchased Between Accepted vs. Rejected Receipts**

sql

SELECT

rewards\_receipt\_status,

SUM(purchased\_item\_count) AS total\_items

FROM receipts

WHERE rewards\_receipt\_status IN ('Accepted', 'Rejected')

GROUP BY rewards\_receipt\_status;

**Interpretation:**

* This query determines whether **Accepted** or **Rejected** receipts have a higher total number of purchased items.

### **5. Brand with the Most Spend Among Users Created in the Last 6 Months**

sql

WITH RecentUsers AS (

SELECT user\_id FROM users

WHERE created\_date >= CURRENT\_DATE - INTERVAL '6 months'

)

SELECT b.name AS brand\_name, SUM(r.total\_spent) AS total\_spent

FROM receipts r

JOIN receipt\_items ri ON r.receipt\_id = ri.receipt\_id

JOIN brands b ON ri.brand\_id = b.brand\_id

WHERE r.user\_id IN (SELECT user\_id FROM RecentUsers)

GROUP BY b.name

ORDER BY total\_spent DESC

LIMIT 1;

**Interpretation:**

* This finds the brand with the **highest spend** among users who registered in the last 6 months.

### **6. Brand with the Most Transactions Among Users Created in the Last 6 Months**

sql

WITH RecentUsers AS (

SELECT user\_id FROM users

WHERE created\_date >= CURRENT\_DATE - INTERVAL '6 months'

)

SELECT b.name AS brand\_name, COUNT(r.receipt\_id) AS transaction\_count

FROM receipts r

JOIN receipt\_items ri ON r.receipt\_id = ri.receipt\_id

JOIN brands b ON ri.brand\_id = b.brand\_id

WHERE r.user\_id IN (SELECT user\_id FROM RecentUsers)

GROUP BY b.name

ORDER BY transaction\_count DESC

LIMIT 1;

**Interpretation:**

* This finds the brand with the **most transactions** (i.e., receipt scans) among users created in the past 6 months.

## **Section 3:**

To ensure the reliability of the **Receipts**, **Users**, and **Brands** data, I conducted a thorough data quality analysis using Python and SQL. Below are the key findings:

### **1. Missing Data Issues**

* **Receipts:** 4,600 missing values across various fields, mainly in purchaseDate and totalSpent.
* **Users:** 166 missing values, affecting createdDate and lastLogin.
* **Brands:** 1,651 missing values, primarily in brandCode and category.

🔹 **Impact:** Missing values can lead to incomplete reporting and inaccurate financial analysis.  
🔹 **Solution:** Decide whether to impute missing values, exclude incomplete records, or flag them for review.

### **2. Duplicate Records**

* **Receipts:** 0 duplicate entries.
* **Users:** 282 duplicate records, likely caused by redundant user registrations.
* **Brands:** 74 duplicate entries after excluding unhashable fields.

🔹 **Impact:** Duplicate users can lead to inflated metrics and incorrect user behavior analysis.  
🔹 **Solution:** Identify criteria to merge duplicate users and standardize brand names.

### **3. Future-Dated Entries**

* **Receipts:** 98 records where purchaseDate is in the future.
* **Users:** 23 accounts with createdDate in the future.

🔹 **Impact:** Future dates suggest data ingestion or timestamp errors, affecting trend analysis.  
🔹 **Solution:** Implement validation rules to reject or correct future timestamps.

### **4. Negative Spend Values**

* 87 receipts contain negative totalSpent values.

🔹 **Impact:** Incorrect totalSpent values could distort financial reports and revenue calculations.  
🔹 **Solution:** Investigate if negative values represent refunds, data entry errors, or system bugs.

### **5. Orphaned Foreign Key Issues**

* **Orphaned Users in Receipts:** 312 receipts reference userId values that do not exist in the Users table.
* **Orphaned Brands in Items:** 521 items reference brandId values that do not exist in the Brands table.

🔹 **Impact:** Orphaned foreign keys indicate data integrity issues and could lead to incomplete analysis.  
🔹 **Solution:** Validate data joins and check if missing users/brands should be backfilled or flagged.

### **6. Final Summary of Data Quality Issues**

|  |  |
| --- | --- |
| **Category** | **Issue Count** |
| Missing Values (Receipts) | 4,600 |
| Missing Values (Users) | 166 |
| Missing Values (Brands) | 1,651 |
| Duplicate Rows (Receipts) | 0 |
| Duplicate Rows (Users) | 282 |
| Duplicate Rows (Brands) | 74 |
| Future Dates (Receipts) | 98 |
| Future Dates (Users) | 23 |
| Negative Spend Values | 87 |
| Orphaned Users in Receipts | 312 |
| Orphaned Brands in Items | 521 |

### **Next Steps & Recommendations**

✅ **Missing Data:** Define handling rules (drop, impute, or flag missing values).  
✅ **Duplicate Records:** Merge duplicate users and brands where necessary.  
✅ **Future-Dated Entries:** Enforce data validation rules to prevent incorrect timestamps.  
✅ **Negative Spend:** Investigate whether negative values indicate refunds or system errors.  
✅ **Orphaned Data:** Implement checks to ensure all userId and brandId values exist in their respective tables.

### **Code Used for Analysis**

I used Python and SQL queries to identify these issues. Below is the Python code used for detecting missing values, duplicates, future dates, negative spend, and orphaned foreign keys.

python:

import pandas as pd

# Checking for missing values in each dataset

missing\_values\_receipts = df\_receipts\_clean.isnull().sum().sum()

missing\_values\_users = df\_users\_clean.isnull().sum().sum()

missing\_values\_brands = df\_brands\_clean.isnull().sum().sum()

# Checking for duplicate rows

duplicates\_receipts = df\_receipts\_clean.duplicated().sum()

duplicates\_users = df\_users\_clean.duplicated().sum()

duplicates\_brands = df\_brands\_clean.drop(columns=['cpg'], errors='ignore').duplicated().sum() # Dropping unhashable field

# Checking for future dates

future\_dates\_receipts = (df\_receipts\_clean['purchaseDate'] > pd.Timestamp.now()).sum()

future\_dates\_users = (df\_users\_clean['createdDate'] > pd.Timestamp.now()).sum()

# Converting `totalSpent` to numeric and checking for negative spend values

df\_receipts\_clean['totalSpent'] = pd.to\_numeric(df\_receipts\_clean['totalSpent'], errors='coerce')

negative\_spend = (df\_receipts\_clean['totalSpent'] < 0).sum()

# Checking for orphaned foreign keys

orphaned\_users\_in\_receipts = (~df\_receipts\_clean['userId'].isin(df\_users\_clean['\_id'])).sum()

orphaned\_brands\_in\_items = (~df\_receipts\_clean.explode('rewardsReceiptItemList')['rewardsReceiptItemList']

.apply(lambda x: x.get('barcode') if isinstance(x, dict) else None)

.isin(df\_brands\_clean['barcode'])).sum()

# Creating a summary table of data quality issues

data\_quality\_issues = pd.DataFrame({

"Category": [

"Missing Values (Receipts)",

"Missing Values (Users)",

"Missing Values (Brands)",

"Duplicate Rows (Receipts)",

"Duplicate Rows (Users)",

"Duplicate Rows (Brands)",

"Future Dates (Receipts)",

"Future Dates (Users)",

"Negative Spend Values",

"Orphaned Users in Receipts",

"Orphaned Brands in Items"

],

"Count": [

missing\_values\_receipts,

missing\_values\_users,

missing\_values\_brands,

duplicates\_receipts,

duplicates\_users,

duplicates\_brands,

future\_dates\_receipts,

future\_dates\_users,

negative\_spend,

orphaned\_users\_in\_receipts,

orphaned\_brands\_in\_items

]

})

# Displaying the results

import ace\_tools as tools

tools.display\_dataframe\_to\_user(name="Final Data Quality Issues", dataframe=data\_quality\_issues)

### **Final Thoughts**

This analysis ensures: ✔ **Data Integrity:** Identifying missing, duplicate, and incorrect data.  
✔ **Scalability:** Preparing for large-scale reporting by handling orphaned foreign keys.  
✔ **Actionability:** Offering clear next steps to improve data quality.

## **Section 4: Stakeholder Communication (Email/Slack Message)**

### **Stakeholder Communication – Email/Slack Message**

**Subject:** Data Analysis Insights & Next Steps for Fetch Rewards

Hi,

I’ve completed an initial review of the Fetch Rewards data, identifying key trends and areas that may require attention. Below is a summary of findings, along with a few questions to ensure we’re aligned on data expectations and next steps.

### **1. Key Insights**

* **Top 5 Brands Analysis:** We identified the most popular brands based on receipts scanned in the last month and compared their rankings to the previous month.
* **User Behavior:** Users who joined within the past 6 months show distinct spending patterns, with a clear leader in both total spend and transaction volume.
* **Receipt Status Impact:** We analyzed how **Accepted vs. Rejected** receipts differ in terms of total spend and item purchases, uncovering differences that may inform product or marketing strategies.

### **2. Data Quality Observations**

During the review, I discovered some **data inconsistencies** that could impact reporting accuracy:  
**Missing Values** – Some receipts lack purchaseDate and totalSpent, which affects trend analysis.

**Data Type Issues** – Dates are stored as text, requiring conversion, and totalSpent is sometimes formatted as a string instead of numeric.  
 **Foreign Key Issues** – Some receipts reference userId values that don’t exist in the Users table, leading to orphaned records.  
 **Unusual Data Patterns** – We found negative spend amounts and future-dated purchases, which may indicate data ingestion issues.

### **3. Open Questions**

To ensure accurate analysis, I’d appreciate clarity on a few points:  
 Are missing purchaseDate and totalSpent values expected behavior, or should they be excluded from reporting?  
 Should **Rejected receipts** be included in spend and transaction analysis, or should we focus only on Accepted ones?  
How should we handle **orphaned user and brand records**—should they be flagged for manual review or ignored?  
 What data refresh cadence should we expect? This will help us determine the best way to optimize performance.

### **4. Performance & Scaling Considerations**

Looking ahead, we need to ensure our approach scales efficiently. Some key concerns:  
**Data Volume Growth** – As transactions grow, queries on receipts and receipt\_items tables will become slower.  
 **Indexing for Performance** – We propose indexing **date-based fields** (purchaseDate) and **foreign keys** (userId, brandId) to improve query speed.  
**Pre-Aggregated Tables** – Creating summary tables for **frequent queries** (e.g., monthly brand rankings) can reduce processing time.

**Data Retention Strategy** – Should we archive older receipts to keep queries efficient?

### **5. Next Steps**

* Awaiting confirmation on how to handle missing and orphaned data.
* Implementing initial optimizations to enhance query performance.
* Planning a discussion on data refresh frequency and retention strategies.

Let me know if you’d like to discuss any of these points further!

**Best Regards,**

**Mohan**