First: Review Existing Unstructured Data and Diagram a New Structured Relational Data Model

Overview of the given unstructured datasets:

1) Receipts Data
2) Users Data
3) Brand Data.

Database Normalization procedure:

1NF:

To ensure **First Normal Form (1NF)** compliance, I identified a column within the **Receipts** dataset that violated the principles of atomicity. The column, *rewardsReceiptItemList*, contained a nested dictionary with multiple items linked to a single receipt, violating the requirement that each cell should contain only one value. Storing multiple items in a single field made it difficult to query individual items, reducing both the clarity and the efficiency of the dataset.

To address this, I extracted the *rewardsReceiptItemList* into a new table called **rewards_receipts**. Each item from the list was normalized into its own row, ensuring that the data was fully atomic, with no repeating groups within any column. I established a clear relationship between this new table and the original **Receipts** table by using *receiptId* as a foreign key, ensuring each item was linked back to the appropriate receipt. Additionally, a composite primary key was created using *partnerItemId* and *receiptId* to ensure that every item within a receipt was uniquely identifiable.

2NF:

Checking for any functional dependencies across columns in the following tables:

i) **Brands:** There is only 1 Primary Key which is *brand_id,* and it is verified that it is unique. Additionally, all the non-key attributes in the Brands table depend on the *brand_id* (As far as my understanding of the business problems at FETCH), we can conclude that Brands does not violate 2NF.

ii) **Users:** After removing the duplicates data, I can verify that *user_id* is unique and all the other non-key attributes depend on *user_id*, ensuring compliance with 2NF.

iii) **Receipts:** Similar conclusion can be drawn or **receipts** table. The table does not violate 2NF.

iv) **Rewards_Receipts:** There are so many variables in this table (around 35) and the composite primary key pair is (*receipt_id, partnerItemId*). But there are some variables which solely does not depend on the composite primary key pair like for example, description, finalPrice, itemPrice etc. Therefore the table violated the 2NF compliance and the table can be further split into two: rewards_receipts and product table where barcode could be the primary key for the product table and the foreign key in the rewards_receipts table. However, an observation is made here about the missing values in barcode:

```
percentage_of_missing_barcode = 100*rewards_receipts_df['barcode'].isna().sum()/rewards_receipts_df.shape[0]
print(percentage_of_missing_barcode)
```
✓ 0.0s

```
55.481919031839794
```

Fig 1.1 Missing value percentage of barcode in **Rewards_Receipts** table.

Given that around 55% of the barcode data is missing, to avoid complications, the table is left as is, without splitting, accepting 2NF violation.

3NF:

Even though the 2NF violation has been accepted, and a decision has been made not to split the tables due to the current data quality issues (i.e., missing barcodes), it is still worth exploring potential 3NF compliance issues. If, in the future, the missing barcode data is resolved and the table can be made compliant with 2NF, the following suggestions could help achieve 3NF.

i) Brands: We can notice a dependency between non-key entities *category* and *categoryCode*

```
### Checking if Category values have a unique pair of category code

### checking inconsistent

inconsistent = brands_df.groupby('category')['categoryCode'].nunique().reset_index()
inconsistent = inconsistent[inconsistent['categoryCode'] > 1]
inconsistent
```
✓ 0.0s

category	categoryCode

Fig 1.2 One – One relationship between *category* and *categoryCode*.

Therefore, the Brands table can be further split into 2 tables, brands and category. Where categoryCode could the foreign Key for brands table. However, the table is not split due

a) There was already an accepted violation of 2NF.
b) 55.6% of the categoryCode values are missing.

category	categoryCode

```
percentage_of_missing_categoryCode = 100*brands_df['categoryCode'].isna().sum()/brands_df.shape[0]
print(percentage_of_missing_categoryCode)
```
✓ 0.0s

```
55.69837189374464
```

Fig 1.3 Missing value percentage of *categoryCode* in **Brands** table.

Since, already decision has been made to retain the tables as is after 1NF, it could be cumbersome to go in depth to analyse for potential violations for every non-key column pair for every table. To automate this process, the following code was written:

```
## one-one relationship check between 2 non-key columns:

df = brands_df # one of the four tables: Brands, Users, Receipts and Rewards_Receipts
pk = ['brand_id'] #primary key (can feed composite key as well)

for columni in df.columns:
    for columnj in df.columns:

        if columni in pk or columnj in pk:
            pass
        else:

            if columni != columnj:

                inconsistent = df.groupby(columni)[columnj].nunique().reset_index()
                inconsistent = inconsistent[inconsistent[columnj] > 1]
                if inconsistent.empty:
                    print(f'one-one relationship exists between {columni} and {columnj}')
```

✓ 0.0s

```
one-one relationship exists between barcode and categoryCode
one-one relationship exists between barcode and cpg_ref
one-one relationship exists between category and categoryCode
one-one relationship exists between categoryCode and category
one-one relationship exists between categoryCode and cpg_ref
one-one relationship exists between name and categoryCode
```

Fig 1.4: Automation code to check for dependencies between non-key columns in each df.

ER-Diagrams:

Relationships between entities.

Users and Receipts:

1) There is a **one-to-many** relationship between the **Users** and **Receipts** tables. The primary key of the **Users** table is *user_id*, and the primary key of the **Receipts** table is *receipt_id*. These tables are connected through a foreign key, *userId*, in the **Receipts** table, which references the *user_id* in the **Users** table. This relationship indicates that each user can generate multiple receipts, but each receipt is associated with exactly one user.

Receipts and Rewards_Receipts:

2) There is a **one-to-many** relationship between the **Receipts** and **Rewards_Receipts** tables. The primary key of the **Receipts** table is *receipt_id*, and the composite primary key of the **Rewards_Receipts** table is (*receipt_id,partnerItemId)*. These tables are connected through a foreign key, *receipt_id*, in the **Receipts** table, which references the *receipt_id* in the **Receipts** table. This relationship indicates that each receipt has multiple rewards/orders

(which is obvious as rewards_receipts is normalized from receipts), but each rewards_receipt row is associated with exactly one receipt.

Brands and Rewards_Receipts:

3) There is a **one-to-many** relationship between the **Brands** and **Rewards_Receipts** tables. The primary key of the **Brands** table is *brand_id*, and the composite primary key of the **Rewards_Receipts** table is (*receipt_id, partnerItemId*). However, due to a couple of data quality issues (which will be discussed in later sections), there is no fixed, reliable way to consistently connect the **Brands** and **Rewards_Receipts** tables. Upon analysis of the columns in the **Rewards_Receipts** table, I identified two columns that might contain information related to brands:

   a. ***Barcode***: While this field appears to be related to products or items, it is unclear whether it can be used to uniquely identify brands, as it may be too specific to individual products.
   b. ***BrandCode***: This column seems to be the better option for identifying brands, as it directly references brands, potentially making it the most viable choice.
   c. ***Other Techniques***: Additional approaches or unique identifiers that could be explored to establish a clearer link between the two tables will be discussed in later sections.

```
###Checking for missing columns to establish a foreign key between brands and rewards_receipts_df

column = 'brandCode'
column_in_rewards_receipts_df = list(rewards_receipts_df[column].unique())
column_in_brands_df = list(brands_df[column].unique())
missing_column = [x for x in column_in_rewards_receipts_df if x not in column_in_brands_df]
print(f'percentage of {column} present in rewards_receipt but not in brands table = {100*len(missing_column)/rewards_receipts_df.shape[0]}')
percentage_of_missing_column = 100*rewards_receipts_df[column].isna().sum()/rewards_receipts_df.shape[0]
print(f'percentage_of_missing_{column} in rewards_receipts is {percentage_of_missing_column}')
percentage_of_missing_column = 100*brands_df[column].isna().sum()/brands_df.shape[0]
print(f'percentage_of_missing_{column} in brands_df is {percentage_of_missing_column}')
```

✓ 0.0s

```
percentage of brandCode present in rewards_receipt but not in brands table = 2.6797291456562453
percentage_of_missing_brandCode in rewards_receipts is 62.54142054459012
percentage_of_missing_brandCode in brands_df is 23.050556983718938
```

```
###is brandcode a unique identifier?
check = brands_df.groupby('brandCode').agg({'brand_id':'count'}).reset_index().sort_values(by = 'brand_id',ascending = False)
check[check['brand_id']>1]
```

✓ 0.0s

	brandCode	brand_id
253	GOODNITES	2
276	HUGGIES	2

Fig 1.5: Percentage of missing *brandCode* in **Rewards_Receipts** and **Brands** tables, and a check for non-unique *brandCode* values.

While the missing data percentages are concerning, *brandCode* remains the most viable column for identifying brands in the absence of a better alternative. After conducting a check for uniqueness, it was found that two brands are not uniquely identified by *brandCode*, which highlights a data quality issue that will need to be addressed.

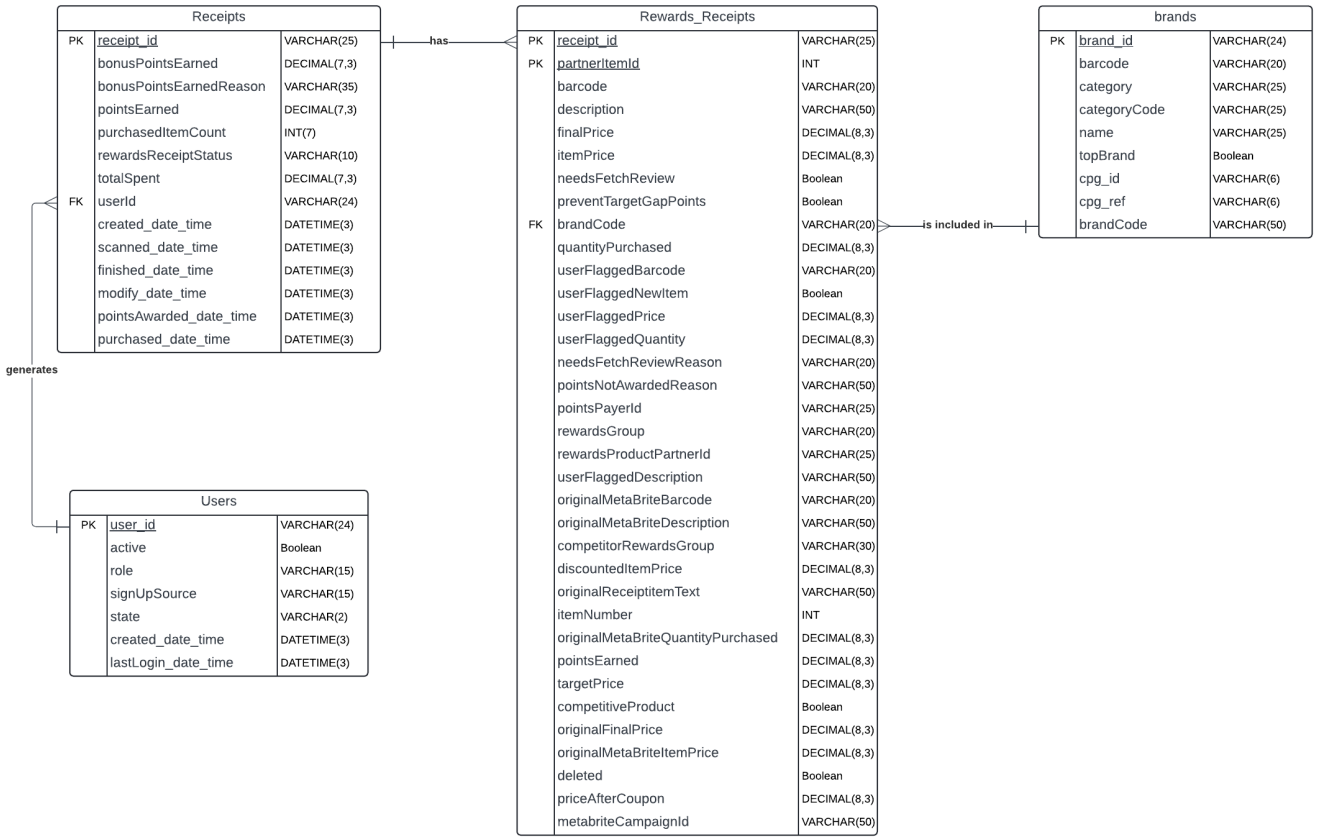

Fig 1.6: Entity Relationship Diagram