In [1]:
```
import json
```

Question 1:Review Existing Unstructured Data and Diagram a New Structured Relational Data Model

In [2]:
```
# Adjusting the approach to handle multiple JSON objects in a file
# We will treat each line as a separate JSON object

def load_json_lines(file_path):
    data = []
    with open(file_path, 'r') as file:
        for line in file:
            data.append(json.loads(line))
    return data

brands_data = load_json_lines('brands.json')
receipts_data = load_json_lines('receipts.json')
users_data = load_json_lines('users.json')

(types := {
    "brands": type(brands_data),
    "receipts": type(receipts_data),
    "users": type(users_data)
}, brands_data[0], receipts_data[0], users_data[0])
```

Out[2]:
```
({'brands': list, 'receipts': list, 'users': list},
 {'_id': {'$oid': '601ac115be37ce2ead437551'},
  'barcode': '511111019862',
  'category': 'Baking',
  'categoryCode': 'BAKING',
  'cpg': {'$id': {'$oid': '601ac114be37ce2ead437550'}, '$ref': 'Cogs'},
  'name': 'test brand @1612366101024',
  'topBrand': False},
 {'_id': {'$oid': '5ff1e1eb0a720f0523000575'},
  'bonusPointsEarned': 500,
  'bonusPointsEarnedReason': 'Receipt number 2 completed, bonus point s
chedule DEFAULT (5cefdcacf3693e0b50e83a36)',
  'createDate': {'$date': 1609687531000},
  'dateScanned': {'$date': 1609687531000},
  'finishedDate': {'$date': 1609687531000},
  'modifyDate': {'$date': 1609687536000},
  'pointsAwardedDate': {'$date': 1609687531000},
  'pointsEarned': '500.0',
  'purchaseDate': {'$date': 1609632000000},
  'purchasedItemCount': 5,
  'rewardsReceiptItemList': [{'barcode': '4011',
    'description': 'ITEM NOT FOUND',
    'finalPrice': '26.00',
    'itemPrice': '26.00',
    'needsFetchReview': False,
    'partnerItemId': '1',
    'preventTargetGapPoints': True,
    'quantityPurchased': 5,
    'userFlaggedBarcode': '4011',
    'userFlaggedNewItem': True,
    'userFlaggedPrice': '26.00',
    'userFlaggedQuantity': 5}],
  'rewardsReceiptStatus': 'FINISHED',
  'totalSpent': '26.00',
  'userId': '5ff1e1eacfcf6c399c274ae6'},
 {'_id': {'$oid': '5ff1e194b6a9d73a3a9f1052'},
  'active': True,
  'createdDate': {'$date': 1609687444800},
  'lastLogin': {'$date': 1609687537858},
  'role': 'consumer',
  'signUpSource': 'Email',
  'state': 'WI'})
```

In [3]:
```
#Standardizing Numeric Fields in receipts.json
from datetime import datetime

# Function to normalize date fields in receipts
def normalize_receipt_dates(receipt):
    date_fields = ['createDate', 'dateScanned', 'finishedDate', 'modifyDa
    for field in date_fields:
        if field in receipt and receipt[field]:
            if isinstance(receipt[field], dict) and '$date' in receipt[f
                timestamp = receipt[field]['$date']
                receipt[field] = datetime.utcfromtimestamp(timestamp / 1(
    return receipt

# Normalize date fields for each receipt
normalized_receipts_data = [normalize_receipt_dates(receipt) for receipt

# Function to standardize numeric fields
def standardize_numeric_fields(receipt):
    numeric_fields = ['bonusPointsEarned', 'pointsEarned', 'purchasedIter
    for field in numeric_fields:
        if field in receipt:
            try:
                receipt[field] = float(receipt[field]) if field in ['poi
            except ValueError:
                receipt[field] = 0 if field in ['bonusPointsEarned', 'pu
    return receipt

# Apply the standardization to each receipt
standardized_receipts_data = [standardize_numeric_fields(receipt) for re
```

In [4]:
```
def clean_item_list(receipt):
    if 'rewardsReceiptItemList' in receipt:
        for item in receipt['rewardsReceiptItemList']:
            # Apply similar cleaning steps: standardizing numbers, handl
            # Example: Ensure 'quantityPurchased' is an integer
            if 'quantityPurchased' in item:
                try:
                    item['quantityPurchased'] = int(item['quantityPurcha
                except ValueError:
                    item['quantityPurchased'] = 0
    return receipt

# Clean the item lists for each receipt
cleaned_receipts_data = [clean_item_list(receipt) for receipt in standar
```

In [5]:
```
#Standardizing Numeric Fields in users.json

def normalize_user_dates(user):
    date_fields = ['createdDate', 'lastLogin']
    for field in date_fields:
        if field in user and user[field]:
            if isinstance(user[field], dict) and '$date' in user[field]:
                timestamp = user[field]['$date']
                user[field] = datetime.utcfromtimestamp(timestamp / 1000
    return user

# Normalize date fields for each user
normalized_users_data = [normalize_user_dates(user) for user in users_da

def standardize_user_fields(user):
    if 'active' in user:
        user['active'] = bool(user['active'])
    if 'state' in user and not user['state']:
        user['state'] = 'Unknown'
    return user

standardized_users_data = [standardize_user_fields(user) for user in nor
```

In [6]:
```
#Standardizing Numeric Fields in brands.json
def standardize_brands_data(brand):
    if 'topBrand' in brand:
        brand['topBrand'] = bool(brand['topBrand'])
    if 'category' in brand and not brand['category']:
        brand['category'] = 'Unknown'
    return brand

# Apply to brands data
standardized_brands_data = [standardize_brands_data(brand) for brand in
```

In [7]:
```
#  transformation for flattening rewardsReceiptItemList in receipts data
transformed_receipts = []
for receipt in standardized_receipts_data:
    if 'rewardsReceiptItemList' in receipt:
        for item in receipt['rewardsReceiptItemList']:
            # Create a new structured item record, including receipt ID a
            transformed_item = {
                'receiptId': receipt['_id']['$oid'],  # Reference to the
                'userId': receipt['userId'],          # Reference to the
                **item  # Include all original item fields
            }
            transformed_receipts.append(transformed_item)
    else:
        transformed_receipts.append(receipt)

# transformed_receipts contains a flat list of items with references to
```

In [8]:
```
# Since brands data may not need flattening like receipts, we will focus
transformed_brands = []
for brand in standardized_brands_data:
    brand_enriched = {**brand}
    brand_enriched['isTopBrand'] = brand.get('topBrand', False)
    transformed_brands.append(brand_enriched)
```

In [9]:
```
# Transform users data to enrich profiles and ensure easy linkage to acti
transformed_users = []
for user in standardized_users_data:
    user_enriched = {**user}
    user_enriched['activityMetric'] = 'ExampleMetricValue'
    transformed_users.append(user_enriched)
```

Question 3: Evaluate Data Quality Issues in the Data Provided

In [10]:
```
#user.json
import pandas as pd

df_users = pd.read_json('users.json', lines=True)

# Basic exploration
print(df_users.shape)
print(df_users.head())
print(df_users.describe(include='all'))
```

```
(495, 7)
                                        _id  active                createdDa
te  \
0  {'$oid': '5ff1e194b6a9d73a3a9f1052'}    True  {'$date': 160968744480
0}
1  {'$oid': '5ff1e194b6a9d73a3a9f1052'}    True  {'$date': 160968744480
0}
2  {'$oid': '5ff1e194b6a9d73a3a9f1052'}    True  {'$date': 160968744480
0}
3  {'$oid': '5ff1e1eacfcf6c399c274ae6'}    True  {'$date': 160968753055
4}
4  {'$oid': '5ff1e194b6a9d73a3a9f1052'}    True  {'$date': 160968744480
0}

                   lastLogin      role signUpSource state
0  {'$date': 1609687537858}  consumer        Email    WI
1  {'$date': 1609687537858}  consumer        Email    WI
2  {'$date': 1609687537858}  consumer        Email    WI
3  {'$date': 1609687530597}  consumer        Email    WI
4  {'$date': 1609687537858}  consumer        Email    WI
                                         _id active                creat
edDate  \
count                                    495    495
495
unique                                   212      2
212
top     {'$oid': '54943462e4b07e684157a532'}   True  {'$date': 14189988
82381}
freq                                      20    494
20

                         lastLogin      role signUpSource state
count                          433       495          447   439
unique                         172         2            2     8
top     {'$date': 1614963143204}  consumer        Email    WI
freq                            20       413          443   396
```

In [11]:
```
#check missing value for users.json
missing_values = df_users.isnull().sum()
print(missing_values[missing_values > 0])
```

```
lastLogin       62
signUpSource    48
state           56
dtype: int64
```

In [12]:
```
#identify duplicate for users.json
# Convert dictionary columns to string
for col in df_users.columns:
    if isinstance(df_users[col].iloc[0], dict):
        df_users[col] = df_users[col].astype(str)

duplicate_rows = df_users.duplicated().sum()
print(f"Total duplicate rows: {duplicate_rows}")
```

```
Total duplicate rows: 283
```

In [13]:
```
#Validate Categorical Data for users.json

# Check unique values in 'role'
unique_roles = df_users['role'].unique()
print("Unique Roles:", unique_roles)

# Check unique values in 'signUpSource'
unique_sign_up_sources = df_users['signUpSource'].unique()
print("Unique SignUpSources:", unique_sign_up_sources)

# Check unique values in 'state'
unique_states = df_users['state'].unique()
print("Unique States:", unique_states)
```

```
Unique Roles: ['consumer' 'fetch-staff']
Unique SignUpSources: ['Email' 'Google' nan]
Unique States: ['WI' 'KY' 'AL' 'CO' 'IL' nan 'OH' 'SC' 'NH']
```

In [14]:
```
#brands.json
df_brands = pd.read_json('brands.json', lines=True)

# Basic exploration
print(df_brands.shape)
print(df_brands.head())
print(df_brands.describe(include='all'))
```

```
(1167, 8)
                                      _id       barcode        category
\
0  {'$oid': '601ac115be37ce2ead437551'}  511111019862          Baking
1  {'$oid': '601c5460be37ce2ead43755f'}  511111519928       Beverages
2  {'$oid': '601ac142be37ce2ead43755d'}  511111819905          Baking
3  {'$oid': '601ac142be37ce2ead43755a'}  511111519874          Baking
4  {'$oid': '601ac142be37ce2ead43755e'}  511111319917  Candy & Sweets

       categoryCode                                                cpg
\
0            BAKING  {'$id': {'$oid': '601ac114be37ce2ead437550'}, ...
1         BEVERAGES  {'$id': {'$oid': '5332f5fbe4b03c9a25efd0ba'}, ...
2            BAKING  {'$id': {'$oid': '601ac142be37ce2ead437559'}, ...
3            BAKING  {'$id': {'$oid': '601ac142be37ce2ead437559'}, ...
4  CANDY_AND_SWEETS  {'$id': {'$oid': '5332fa12e4b03c9a25efd1e7'}, ...

                          name  topBrand                    brandCode
0  test brand @1612366101024       0.0                          NaN
1                  Starbucks       0.0                     STARBUCKS
2  test brand @1612366146176       0.0  TEST BRANDCODE @1612366146176
3  test brand @1612366146051       0.0  TEST BRANDCODE @1612366146051
4  test brand @1612366146827       0.0  TEST BRANDCODE @1612366146827
                                       _id       barcode category  \
count                                  1167  1.167000e+03     1012
unique                                 1167           NaN       23
top     {'$oid': '601ac115be37ce2ead437551'}           NaN   Baking
freq                                      1           NaN      369
mean                                    NaN  5.111115e+11      NaN
std                                     NaN  2.874497e+05      NaN
min                                     NaN  5.111110e+11      NaN
25%                                     NaN  5.111112e+11      NaN
50%                                     NaN  5.111114e+11      NaN
75%                                     NaN  5.111117e+11      NaN
max                                     NaN  5.111119e+11      NaN

       categoryCode                                                cpg
\
count           517                                                1167
unique           14                                                 204
top          BAKING  {'$ref': 'Cogs', '$id': {'$oid': '559c2234e4b0...
freq            359                                                  98
mean            NaN                                                 NaN
std             NaN                                                 NaN
min             NaN                                                 NaN
25%             NaN                                                 NaN
50%             NaN                                                 NaN
75%             NaN                                                 NaN
max             NaN                                                 NaN

           name    topBrand brandCode
count      1167  555.000000       933
unique     1156         NaN       897
top     Huggies         NaN
freq          2         NaN        35
mean        NaN    0.055856       NaN
std         NaN    0.229850       NaN
```

```
min         NaN    0.000000       NaN
25%         NaN    0.000000       NaN
50%         NaN    0.000000       NaN
75%         NaN    0.000000       NaN
max         NaN    1.000000       NaN
```

In [15]:
```
#check missing value for brands.json
missing_values = df_brands.isnull().sum()
print(missing_values[missing_values > 0])
```

```
category        155
categoryCode    650
topBrand        612
brandCode       234
dtype: int64
```

In [16]:
```
#identify duplicate for brands.json

for col in df_brands.columns:
    if isinstance(df_brands[col].iloc[0], dict):
        df_brands[col] = df_brands[col].astype(str)

duplicate_rows = df_brands.duplicated().sum()
print(f"Total duplicate rows: {duplicate_rows}")
```

```
Total duplicate rows: 0
```

In [17]:
```


# Validate 'category' column for unique values and missing data
unique_categories = df_brands['category'].unique()
print("Unique Categories:", unique_categories)

# Validate 'categoryCode' column for unique values and missing data
unique_category_codes = df_brands['categoryCode'].unique()
print("Unique Category Codes:", unique_category_codes)

# Validate 'topBrand' column; checking if it's treated as boolean or con
unique_top_brand = df_brands['topBrand'].unique()
print("Unique TopBrand Values:", unique_top_brand)
```

```
Unique Categories: ['Baking' 'Beverages' 'Candy & Sweets' 'Condiments &
Sauces'
 'Canned Goods & Soups' nan 'Magazines' 'Breakfast & Cereal'
 'Beer Wine Spirits' 'Health & Wellness' 'Beauty' 'Baby' 'Frozen'
 'Grocery' 'Snacks' 'Household' 'Personal Care' 'Dairy'
 'Cleaning & Home Improvement' 'Deli' 'Beauty & Personal Care'
 'Bread & Bakery' 'Outdoor' 'Dairy & Refrigerated']
Unique Category Codes: ['BAKING' 'BEVERAGES' 'CANDY_AND_SWEETS' nan 'HE
ALTHY_AND_WELLNESS'
 'GROCERY' 'PERSONAL_CARE' 'CLEANING_AND_HOME_IMPROVEMENT'
 'BEER_WINE_SPIRITS' 'BABY' 'BREAD_AND_BAKERY' 'OUTDOOR'
 'DAIRY_AND_REFRIGERATED' 'MAGAZINES' 'FROZEN']
Unique TopBrand Values: [ 0. nan  1.]
```

In [18]:
```
#receipts.json
df_receipts = pd.read_json('receipts.json', lines=True)

# Basic exploration
print(df_receipts.shape)
print(df_receipts.head())
print(df_receipts.describe(include='all'))
```

```
4  {'$oid': '5ff1e1d20a7214ada1000561'}               5.0

                             bonusPointsEarnedReason  \
0  Receipt number 2 completed, bonus point schedu...
1  Receipt number 5 completed, bonus point schedu...
2                        All-receipts receipt bonus
3                        All-receipts receipt bonus
4                        All-receipts receipt bonus

                  createDate               dateScanned  \
0  {'$date': 1609687531000}  {'$date': 1609687531000}
1  {'$date': 1609687483000}  {'$date': 1609687483000}
2  {'$date': 1609687537000}  {'$date': 1609687537000}
3  {'$date': 1609687534000}  {'$date': 1609687534000}
4  {'$date': 1609687506000}  {'$date': 1609687506000}

                finishedDate                modifyDate  \
0  {'$date': 1609687531000}  {'$date': 1609687536000}
1  {'$date': 1609687483000}  {'$date': 1609687488000}
2                       NaN  {'$date': 1609687542000}
```

In [19]:
```
#check missing value for receipts.json
missing_values = df_receipts.isnull().sum()
print(missing_values[missing_values > 0])
```

```
bonusPointsEarned          575
bonusPointsEarnedReason    575
finishedDate               551
pointsAwardedDate          582
pointsEarned               510
purchaseDate               448
purchasedItemCount         484
rewardsReceiptItemList     440
totalSpent                 435
dtype: int64
```

In [20]:
```
#identify duplicate for receipts.json
for col in df_receipts.columns:
    if isinstance(df_receipts[col].iloc[0], dict) or isinstance(df_recei
        df_receipts[col] = df_receipts[col].apply(lambda x: str(x))


duplicate_rows = df_receipts.duplicated().sum()
print(f"Total duplicate rows: {duplicate_rows}")
```

```
Total duplicate rows: 0
```

In [21]:
```
# Validate Categorical Data for receipts.json

#Check unique values in 'rewardsReceiptStatus' to validate categorical da
unique_receipt_statuses = df_receipts['rewardsReceiptStatus'].unique()
print("Unique Receipt Statuses:", unique_receipt_statuses)
```

```
Unique Receipt Statuses: ['FINISHED' 'REJECTED' 'FLAGGED' 'SUBMITTED'
'PENDING']
```

In [ ]:
```

```