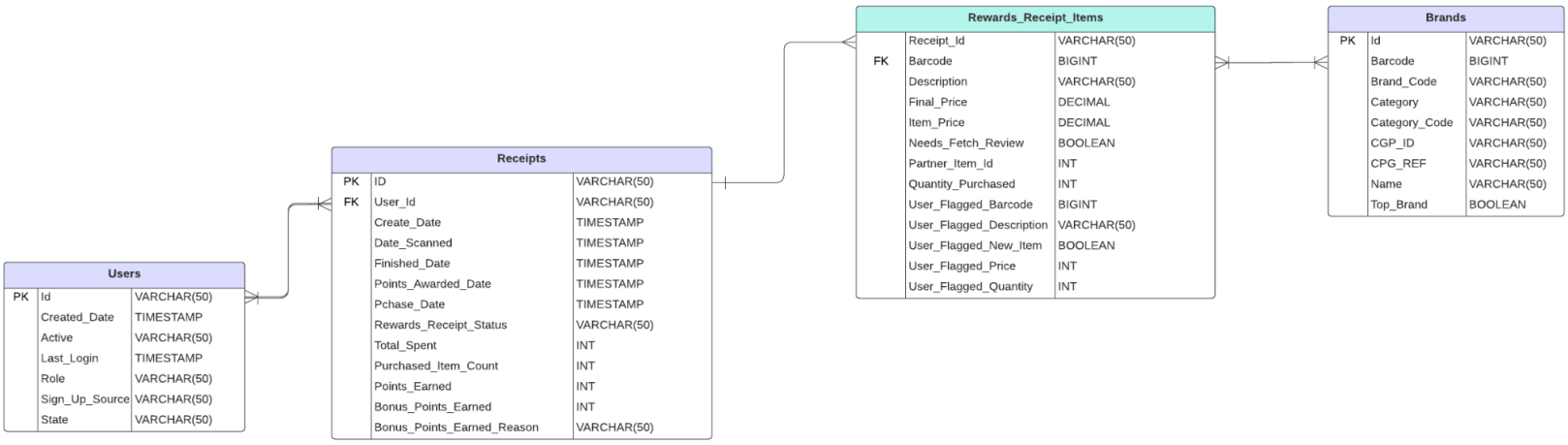
Fetch Take Home Assignment

Snowflake Credentials:

Login Link - https://huqqiiq-cp43263.snowflakecomputing.com/console/login
Username - vikram
Password - Vikram123
Database - FETCH
Schema - DATA
Role - USERADMIN
Warehouse - COMPUTE_WH
Worksheets - Data Ingestion (SQL used for Data Ingestion), SQL (SQL used to answer questions)

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



The three JSON files containing data for brands, receipts, and users were provided. To answer the SQL questions, I first ingested this data into the Snowflake data warehouse using the following steps:

- **SnowSQL Connection**: Since the data size was relatively large, I used SnowSQL to connect to Snowflake.
- **Database and Schema Creation**: I created a database named Fetch and a schema named Data.

- **Stage and Table Setup**: I created three separate stages for receipts, brands, and users. Next, I created staging tables to store the raw JSON data.
- **Table Creation**: I created four main tables: receipts, rewards_receipt_items and brands, rewards_receipt_items.
    - **Note**: I have further flattened the Items list in the receipts table.
- **File Format Configuration**: I configured a file format compatible with the JSON format to handle the data correctly.
- **Data Upload**: I uploaded the data into the respective stages using the PUT command.
- **Data Loading**: I copied the staged data into the staging tables.
- **Data Transformation and Insertion**: I inserted the data into the main tables, ensuring proper formatting and type casting of fields.

SQL Ingestion Code:

```
create or replace database fetch;
create or replace schema fetch.data;

CREATE TABLE fetch.data.receipts (
    id STRING,
    bonus_points_earned INTEGER,
    bonus_points_earned_reason STRING,
    create_date TIMESTAMP,
    date_scanned TIMESTAMP,
    finished_date TIMESTAMP,
    modify_date TIMESTAMP,
    points_awarded_date TIMESTAMP,
    points_earned FLOAT,
    purchase_date TIMESTAMP,
    purchased_item_count INTEGER,
    rewards_receipt_item_list ARRAY,
    rewards_receipt_status STRING,
    total_spent FLOAT,
    user_id STRING
);

CREATE TABLE fetch.data.rewards_receipt_items (
    receipt_id STRING,
    barcode STRING,
    description STRING,
    final_price FLOAT,
    item_price FLOAT,
```

```
    needs_fetch_review BOOLEAN,
    partner_item_id STRING,
    quantity_purchased INTEGER,
    user_flagged_barcode STRING,
    user_flagged_description STRING,
    user_flagged_new_item BOOLEAN,
    user_flagged_price FLOAT,
    user_flagged_quantity INTEGER
);

CREATE TABLE fetch.data.users (
    id STRING,
    active BOOLEAN,
    created_date TIMESTAMP,
    last_login TIMESTAMP,
    role STRING,
    sign_up_source STRING,
    state STRING
);

CREATE TABLE fetch.data.brands (
    id STRING,
    barcode STRING,
    brand_code STRING,
    category STRING,
    category_code STRING,
    cpg_id STRING,
    cpg_ref STRING,
    name STRING,
    top_brand BOOLEAN
);

CREATE OR REPLACE TABLE fetch.data.receipts_staging (
    raw VARIANT
);
CREATE OR REPLACE TABLE fetch.data.users_staging (
    raw VARIANT
);
CREATE OR REPLACE TABLE fetch.data.brands_staging (
    raw VARIANT
);

CREATE OR REPLACE FILE FORMAT fetch.data.sf_tut_json_format
```

```
  TYPE = JSON;

CREATE OR REPLACE TEMPORARY STAGE fetch.data.receipts_stage FILE_FORMAT =
sf_tut_json_format;
CREATE OR REPLACE TEMPORARY STAGE fetch.data.users_stage FILE_FORMAT =
sf_tut_json_format;
CREATE OR REPLACE TEMPORARY STAGE fetch.data.brands_stage FILE_FORMAT =
sf_tut_json_format;

show stages in fetch.data;


PUT file:///Users/Desktop/fetch/receipts.json @fetch.data.receipts_stage
AUTO_COMPRESS=TRUE;
PUT file:///Users/Desktop/fetch/brands.json @fetch.data.brands_stage
AUTO_COMPRESS=TRUE;
PUT file:///Users/Desktop/fetch/users.json @fetch.data.users_stage
AUTO_COMPRESS=TRUE;

COPY INTO fetch.data.receipts_staging
FROM @fetch.data.receipts_stage
FILE_FORMAT = (FORMAT_NAME = fetch.data.sf_tut_json_format);

COPY INTO fetch.data.users_staging
FROM @fetch.data.users_stage
FILE_FORMAT = (FORMAT_NAME = fetch.data.sf_tut_json_format);

COPY INTO fetch.data.brands_staging
FROM @fetch.data.brands_stage
FILE_FORMAT = (FORMAT_NAME = fetch.data.sf_tut_json_format);

INSERT INTO fetch.data.receipts (
    id,
    bonus_points_earned,
    bonus_points_earned_reason,
    create_date,
    date_scanned,
    finished_date,
    modify_date,
    points_awarded_date,
    points_earned,
    purchase_date,
    purchased_item_count,
```

```
    rewards_receipt_item_list,
    rewards_receipt_status,
    total_spent,
    user_id
)
SELECT
    raw:"_id"."$oid"::STRING AS id,
    raw:"bonusPointsEarned"::INTEGER AS bonus_points_earned,
    raw:"bonusPointsEarnedReason"::STRING AS bonus_points_earned_reason,
    ((raw:"createDate"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
create_date,
    ((raw:"dateScanned"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
date_scanned,
    ((raw:"finishedDate"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
finished_date,
    ((raw:"modifyDate"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
modify_date,
    ((raw:"pointsAwardedDate"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
points_awarded_date,
    raw:"pointsEarned"::FLOAT AS points_earned,
    ((raw:"purchaseDate"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
purchase_date,
    raw:"purchasedItemCount"::INTEGER AS purchased_item_count,
    raw:"rewardsReceiptItemList"::ARRAY AS rewards_receipt_item_list,
    raw:"rewardsReceiptStatus"::STRING AS rewards_receipt_status,
    raw:"totalSpent"::FLOAT AS total_spent,
    raw:"userId"::STRING AS user_id
FROM fetch.data.receipts_staging;

INSERT INTO fetch.data.rewards_receipt_items (
    receipt_id,
    barcode,
    description,
    final_price,
    item_price,
    needs_fetch_review,
    partner_item_id,
    quantity_purchased,
    user_flagged_barcode,
    user_flagged_description,
    user_flagged_new_item,
    user_flagged_price,
    user_flagged_quantity
```

```
)
SELECT
    r.id AS receipt_id, -- Link to the receipts table
    item.value:"barcode"::STRING AS barcode,
    item.value:"description"::STRING AS description,
    item.value:"finalPrice"::FLOAT AS final_price,
    item.value:"itemPrice"::FLOAT AS item_price,
    item.value:"needsFetchReview"::BOOLEAN AS needs_fetch_review,
    item.value:"partnerItemId"::STRING AS partner_item_id,
    item.value:"quantityPurchased"::INTEGER AS quantity_purchased,
    item.value:"userFlaggedBarcode"::STRING AS user_flagged_barcode,
    item.value:"userFlaggedDescription"::STRING AS user_flagged_description,
    item.value:"userFlaggedNewItem"::BOOLEAN AS user_flagged_new_item,
    item.value:"userFlaggedPrice"::FLOAT AS user_flagged_price,
    item.value:"userFlaggedQuantity"::INTEGER AS user_flagged_quantity
FROM
    fetch.data.receipts AS r,
    LATERAL FLATTEN(input => r.rewards_receipt_item_list) AS item;

INSERT INTO fetch.data.users (
    id,
    active,
    created_date,
    last_login,
    role,
    sign_up_source,
    state
)
SELECT
    raw:"_id"."$oid"::STRING AS id,
    raw:active::BOOLEAN AS active,
    ((raw:"createdDate"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
created_date,
    ((raw:"lastLogin"."$date"::INTEGER / 1000) :: TIMESTAMP_NTZ) AS
last_login,
    raw:role::STRING AS role,
    raw:"signUpSource"::STRING AS sign_up_source,
    raw:state::STRING AS state
FROM fetch.data.users_staging;

INSERT INTO fetch.data.brands (
    id,
    barcode,
```

```
    brand_code,
    category,
    category_code,
    cpg_id,
    cpg_ref,
    name,
    top_brand
)
SELECT
    raw:"_id"."$oid"::STRING AS id,
    raw:barcode::STRING AS barcode,
    raw:"brandCode"::STRING AS brand_code,
    raw:category::STRING AS category,
    raw:"categoryCode"::STRING AS category_code,
    raw:cpg."$id"."$oid"::STRING AS cpg_id,
    raw:cpg."$ref"::STRING AS cpg_ref,
    raw:name::STRING AS name,
    raw:"topBrand"::BOOLEAN AS top_brand
FROM fetch.data.brands_staging;
```

Second: Write queries that directly answer predetermined questions from a business stakeholder

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

- The receipts with **valid barcodes** and **valid brand names** have the date_scanned only in '2021-01-01' month. So, I have considered Jan 2021 to get the top 5 brands by receipts
- I have used dense_rank to get all the brands if they are tied with same number of receipts scanned

```
with combined as(
    select distinct receipts.id as receipt_id,
           receipts.date_scanned,
           receipt_items.barcode,
           brands.category,
           brands.name
    from fetch.data.receipts receipts
    left join fetch.data.rewards_receipt_items receipt_items on receipts.id =
receipt_items.receipt_id
    left join fetch.data.brands brands on receipt_items.barcode =
brands.barcode
    where receipt_items.barcode is not null
)
```

```sql
select name as brand_name,
       count(*) as total_scans
from combined
where name is not null and
date_trunc('month', date_scanned)::date in ('2021-01-01')
group by all
qualify dense_rank() over (order by total_scans desc) <= 5;
```

	BRAND_NAME	TOTAL_SCANS
1	Tostitos	11
2	Swanson	11
3	Cracker Barrel Cheese	10
4	Diet Chris Cola	4
5	Prego	4
6	Kraft	3
7	Jell-O	3
8	Quaker	3
9	Kettle Brand	3
10	V8	2
11	Rice A Roni	2
12	Pepperidge Farm	2
13	Cheetos	2

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?**
   - The receipts with **valid barcodes** and **valid brand names** have the date_scanned only in '2021-01-01' month. So, I have considered Jan 2021 and Dec 2020 to get the top 5 brands by receipts. Dec 2020 data is not available.
   - I have used dense_rank to get all the brands if they are tied with same number of receipts scanned and partitioned based on month

```sql
with combined as(
    select distinct receipts.id as receipt_id,
           receipts.date_scanned,
           receipt_items.barcode,
           brands.category,
           brands.name,
           date_trunc('month', receipts.date_scanned)::date as month_start
    from fetch.data.receipts receipts
    left join fetch.data.rewards_receipt_items receipt_items on receipts.id =
receipt_items.receipt_id
    left join fetch.data.brands brands on receipt_items.barcode =
brands.barcode
where receipt_items.barcode is not null
)
```

```sql
select name as brand_name,
       month_start,
       count(*) as total_scans
from combined
where name is not null and
month_start in ('2021-01-01', '2020-12-01')
group by all
qualify dense_rank() over (partition by month_start order by total_scans
desc) <= 5;
```

	BRAND_NAME	MONTH_START	TOTAL_SCANS
1	Tostitos	2021-01-01	11
2	Swanson	2021-01-01	11
3	Cracker Barrel Cheese	2021-01-01	10
4	Prego	2021-01-01	4
5	Diet Chris Cola	2021-01-01	4
6	Quaker	2021-01-01	3
7	Jell-O	2021-01-01	3
8	Kraft	2021-01-01	3
9	Kettle Brand	2021-01-01	3
10	Cheetos	2021-01-01	2
11	Pepperidge Farm	2021-01-01	2
12	V8	2021-01-01	2
13	Rice A Roni	2021-01-01	2

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

The **Accepted** rewardsReceiptStatus has higher average spend compared to the Rejected

```sql
select distinct rewards_receipt_status,
       avg(total_spent) over (partition by rewards_receipt_status) as
avg_spent
from fetch.data.receipts
where rewards_receipt_status in ('FINISHED', 'REJECTED')
order by avg_spent desc;
```

	REWARDS_RECEIPT_STATUS	AVG_SPENT
1	FINISHED	80.854305019
2	REJECTED	23.326056338

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

The **Accepted** rewardsReceiptStatus has higher total number of items purchased compared to the Rejected

```
select distinct rewards_receipt_status,
       coalesce(sum(purchased_item_count) over (partition by
rewards_receipt_status), 0) as total_items
from fetch.data.receipts
where rewards_receipt_status in ('FINISHED', 'REJECTED')
order by total_items desc;
```

	REWARDS_RECEIPT_STATUS	TOTAL_ITEMS
1	FINISHED	8184
2	REJECTED	173

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

- The last receipt scanned date is '2021-03-01', I have considered this date and looked for the users who were created past six months
- **Tostitos** has the most spend among the users who were created within the past 6 months

```
with recent_users as (
    select distinct id as user_id
    from fetch.data.users
    where created_date >= dateadd(month, -6, '2021-03-01')
    ),
user_receipts as (
    select r.id as receipt_id,
           r.total_spent,
           i.barcode as item_barcode,
           r.user_id
    from fetch.data.receipts as r
    join fetch.data.rewards_receipt_items as i on r.id = i.receipt_id
    join recent_users as u
    on r.user_id = u.user_id
),
brand_spend as (
```

```
    select b.name as brand_name,
           sum(ur.total_spent) as total_spend
    from user_receipts as ur
    join fetch.data.brands as b
    on ur.item_barcode = b.barcode
    group by b.name
)
select brand_name,
       total_spend
from brand_spend
order by total_spend desc;
```

	BRAND_NAME	TOTAL_SPEND
1	Tostitos	15799.37
2	Pepperidge Farm	14165.85
3	V8	9443.9
4	Prego	9443.9
5	Diet Chris Cola	9443.9
6	Swanson	7187.14
7	Cracker Barrel Cheese	4885.89
8	Jell-O	4754.37
9	Cheetos	4721.95
10	Kettle Brand	2400.91
11	Grey Poupon	743.79
12	Quaker	32.42

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

- The last receipt scanned date is '2021-03-01', I have considered this date and looked for the users who were created past six months
- **Swanson** has the most transactions among the users who were created within the past 6 months.

```
with recent_users as (
    select distinct id as user_id
    from fetch.data.users
    where created_date >= dateadd(month, -6, '2021-03-01')
    -- since select max(create_date) from fetch.data.receipts = 2021-03-01
    --where created_date >= dateadd(month, -6, current_date()) -- Users
created in the past 6 months
)
,
user_receipts as (
    select r.id as receipt_id,
           r.total_spent,
```

```
            i.barcode as item_barcode,
            r.user_id
    from fetch.data.receipts as r
    join fetch.data.rewards_receipt_items as i on r.id = i.receipt_id
    join recent_users as u
    on r.user_id = u.user_id
    --where r.item_barcode is not null
),
brand_transactions as (
    select b.name as brand_name,
           count(distinct ur.receipt_id) as transaction_count
    from user_receipts as ur
    join fetch.data.brands as b on ur.item_barcode = b.barcode
    group by b.name
)
select brand_name,
       transaction_count
from brand_transactions
order by transaction_count desc;
```

	BRAND_NAME	TRANSACTION_COUNT
1	Swanson	11
2	Tostitos	11
3	Kettle Brand	3
4	Jell-O	2
5	Cracker Barrel Cheese	2
6	V8	1
7	Prego	1
8	Diet Chris Cola	1
9	Grey Poupon	1
10	Cheetos	1
11	Quaker	1
12	Pepperidge Farm	1

Third: Evaluate Data Quality Issues in the Data Provided

- The ID column in the three main tables (receipts, brands, users) is expected to act as a primary key, meaning it should not contain duplicates or null values. While the receipts and brands tables have unique primary keys, the users table contains duplicates in the ID column.

- To identify the duplicate user_id values, the following query can be used:

```
select id,
       count(*)
from fetch.data.users
```

```
group by id
having count(*) > 1
order by 2 desc;
```

- Another issue is observed in the receipts table, where **purchase_date** is greater than **date_scanned**. This indicates that some receipts were scanned before the purchase was made, which is logically inconsistent.
    - To identify such problematic records:
      ```
      select *
      from fetch.data.receipts
      where date_scanned < purchase_date;
      ```
- In the rewards_receipt_items table, two significant data quality issues were identified:
    - Duplicate Barcodes with Conflicting Partner IDs: There are cases where two items have the same barcode, but the partner_id field differs.
    - Missing Barcodes: Many items lack a barcode even though final_price and item_price are present. This creates a gap in the analysis of purchase categories, as missing barcodes prevent proper classification and analysis.

Fourth: Communicate with Stakeholders

What questions do you have about the data?

- Is the app used exclusively in the United States? The users table includes a state field but lacks a country field, which would be critical for supporting international usage.
- What time zone is used for the receipts and source data? What time zone should the final reporting data models align with to ensure consistency and accuracy?
- The receipts data with non-null barcode values is only available for January 2021 and does not include previous months. Is this expected or a limitation in the data capture?
- If the data is generated using OCR technology from receipts, how accurate is the extraction process? Are there known issues or inconsistencies in capturing the data?
- Are there fields missing from the current schema that would be useful for additional analysis? For example, fields to track promotional offers or item categories.

- Are we tracking user data using Type-2 Slowly Changing Dimensions (SCD)? Type-2 SCD would help in understanding user behavior over time, including tracking changes in user activity and acquisition channels.
- Are data quality checks in place? Are there processes to validate data quality, such as preventing null values and duplicates during ingestion?
- What is the relationship between create_date and date_scanned in receipts table? Are these fields expected to differ? For example, should date_scanned always occur after create_date?
- How is the rewards_receipt_status processed? What is the order of execution for this field? Is there a lifecycle for receipts that progresses from one status to another?

How did you discover the data quality issues?

- These data quality issues were identified while writing SQL queries to analyze the data and answer the above questions. For instance:
    - Duplicate results were observed when joining the users table due to multiple records with the same user_id.
    - Some items in the rewards_receipt_items table were missing in the final output because their barcode was null, causing issues when joining with the brands table.

What do you need to know to resolve the data quality issues?

- **Primary Key Rules:** Which columns are intended to be unique? Should they serve as primary keys? For instance: Should the id column in the users, receipts and brands tables always be unique and non-null?
- **Field Relationships:** Should purchase_date always be before or on the same day as date_scanned in the receipts table? Are there rules governing the relationship between partner_item_id and barcode in teh rewards_receipt_items table?
- **Null Constraints:** Which fields must not contain NULL values (eg: barcode, user_id, total_spent)
- **Data Integrity:** Should every item have a valid barcode that matches an entry in the brands table?
- By understanding these requirements will help establish clear validation rules, enforce consistency, and maintain the reliability of the data for the analysis.

What other information would you need to help you optimize the data assets you're trying to create?

- Information on data refresh frequency, data lineage, and ownership to ensure accuracy and accountability.
- Insights into which fields and tables are most used by downstream systems to optimize data modeling and storage.
- Clarity on how much historical data is needed for analysis, which can help in designing incremental strategies.

What performance and scaling concerns do you anticipate in production and how do you plan to address them?

- Since the receipts and receipt_items data grows exponentially, it becomes hard to maintain because any complex logic/aggregation is built using this table. Implement incremental data processing to only insert or update the latest records from the source streams. This reduces both computation and execution time.
- This can be handled via implementing an incremental data processing approach to handle updates and inserts efficiently. This involves processing on the latest records from source streams, rather than reprocessing the entire dataset.
- By using query optimizations such as proper indexing, clustering and partitioning to ensure that queries on large tables execute efficiently
- Assigning appropriate warehouse sizes based on model execution needs:
    - Use an XL warehouse for heavy operations like processing the receipts table.
    - Use Medium or Large warehouses for models that consume this data.
    - This approach balances performance and cost, ensuring scalability as the data grows.