**Additional Data Quality Checks**

#### **1. Data Profiling**

Data profiling involves analyzing the distribution, patterns, and statistics of the data. This helps identify anomalies and understand the data better.

-- Numeric Column Statistics (e.g., totalSpent in Receipts)

SELECT

MIN(totalSpent) AS min\_totalSpent,

MAX(totalSpent) AS max\_totalSpent,

AVG(totalSpent) AS avg\_totalSpent,

STDDEV(totalSpent) AS stddev\_totalSpent

FROM Receipts;

-- Date Column Statistics (e.g., createdDate in Users)

SELECT

MIN(createdDate) AS earliest\_createdDate,

MAX(createdDate) AS latest\_createdDate,

DATEDIFF(day, MIN(createdDate), MAX(createdDate)) AS date\_range\_days

FROM Users;

-- Categorical Column Distribution (e.g., rewardsReceiptStatus in Receipts)

SELECT

rewardsReceiptStatus,

COUNT(\*) AS status\_count

FROM Receipts

GROUP BY rewardsReceiptStatus;

#### **2. Outlier Detection**

Outliers are values that deviate significantly from the rest of the data. Detecting them helps identify potential data entry errors or unusual patterns.

-- Outliers in totalSpent (Receipts)

SELECT

\_id AS receipt\_id,

totalSpent

FROM Receipts

WHERE totalSpent > (SELECT AVG(totalSpent) + 3 \* STDDEV(totalSpent) FROM Receipts)

OR totalSpent < (SELECT AVG(totalSpent) - 3 \* STDDEV(totalSpent) FROM Receipts);

-- Outliers in quantity (ReceiptItems)

SELECT

\_id AS item\_id,

quantity

FROM ReceiptItems

WHERE quantity > (SELECT AVG(quantity) + 3 \* STDDEV(quantity) FROM ReceiptItems)

OR quantity < (SELECT AVG(quantity) - 3 \* STDDEV(quantity) FROM ReceiptItems);

#### **3. Cross-Table Consistency Checks**

Ensure that data is consistent across related tables. For example, check if the totalSpent in the **Receipts** table matches the sum of price \* quantity in the **ReceiptItems** table.

-- Check consistency between Receipts.totalSpent and ReceiptItems (price \* quantity)

SELECT

r.\_id AS receipt\_id,

r.totalSpent AS receipt\_totalSpent,

SUM(ri.price \* ri.quantity) AS calculated\_totalSpent,

CASE

WHEN r.totalSpent = SUM(ri.price \* ri.quantity) THEN 'Consistent'

ELSE 'Inconsistent'

END AS consistency\_status

FROM Receipts r

JOIN ReceiptItems ri ON r.\_id = ri.receiptId

GROUP BY r.\_id, r.totalSpent

HAVING r.totalSpent != SUM(ri.price \* ri.quantity);

#### **4. Temporal Data Quality Checks**

Check for issues related to time-based data, such as:

* Future dates (e.g., createdDate in the future).
* Illogical date sequences (e.g., finishedDate before dateScanned).

-- Future dates in Receipts

SELECT

\_id AS receipt\_id,

dateScanned,

purchaseDate,

finishedDate

FROM Receipts

WHERE dateScanned > CURRENT\_DATE

OR purchaseDate > CURRENT\_DATE

OR finishedDate > CURRENT\_DATE;

-- Illogical date sequences in Receipts

SELECT

\_id AS receipt\_id,

dateScanned,

finishedDate

FROM Receipts

WHERE finishedDate < dateScanned;

#### **5. Business Rule Validation**

Validate data against specific business rules. For example:

* Ensure bonusPointsEarned is consistent with pointsEarned in the **Receipts** table.
* Ensure purchasedItemCount matches the count of items in rewardsReceiptItemList.

-- Validate bonusPointsEarned and pointsEarned

SELECT

\_id AS receipt\_id,

bonusPointsEarned,

pointsEarned

FROM Receipts

WHERE bonusPointsEarned != pointsEarned;

-- Validate purchasedItemCount

SELECT

r.\_id AS receipt\_id,

r.purchasedItemCount AS receipt\_item\_count,

COUNT(ri.\_id) AS actual\_item\_count

FROM Receipts r

LEFT JOIN ReceiptItems ri ON r.\_id = ri.receiptId

GROUP BY r.\_id, r.purchasedItemCount

HAVING r.purchasedItemCount != COUNT(ri.\_id);

#### **6. Data Completeness Checks**

Ensure that all expected data is present. For example:

* Check if all receipts have at least one item.
* Check if all users have at least one receipt.

-- Receipts without items

SELECT

r.\_id AS receipt\_id

FROM Receipts r

LEFT JOIN ReceiptItems ri ON r.\_id = ri.receiptId

WHERE ri.\_id IS NULL;

-- Users without receipts

SELECT

u.\_id AS user\_id

FROM Users u

LEFT JOIN Receipts r ON u.\_id = r.userId

WHERE r.\_id IS NULL;

#### **7. Data Uniqueness Checks**

Ensure that unique constraints are enforced. For example:

* Check if brandCode in the **Brands** table is unique.
* Check if barcode in the **Brands** table is unique.

-- Duplicate brandCode

SELECT

brandCode,

COUNT(\*) AS duplicate\_count

FROM Brands

GROUP BY brandCode

HAVING COUNT(\*) > 1;

-- Duplicate barcode

SELECT

barcode,

COUNT(\*) AS duplicate\_count

FROM Brands

GROUP BY barcode

HAVING COUNT(\*) > 1;

#### **8. Data Freshness Checks**

Ensure that the data is up-to-date. For example:

* Check when the last receipt was scanned.
* Check when the last user logged in.

-- Last receipt scanned

SELECT

MAX(dateScanned) AS last\_receipt\_scanned

FROM Receipts;

-- Last user login

SELECT

MAX(lastLogin) AS last\_user\_login

FROM Users;