Retail Sales Part 1: Data Cleaning

Overview

For this project we will be cleaning and analyzing our retail sales data. The aim is to provide decision makers with recommendations on sales and purchasing strategies.

TABLES

- Retail Sales (See table description provided by data author below)

Column Name	Description	Example Values
Transaction ID	A unique identifier for each transaction. Always present and unique.	TXN_1234567
Customer ID	A unique identifier for each customer. 25 unique customers.	CUST_01
Category	The category of the purchased item.	Food, Furniture
Item	The name of the purchased item. May contain missing values or None.	Item_1_FOOD, None
Price Per Unit	The static price of a single unit of the item. May contain missing or None values.	4.00, None
Quantity	The quantity of the item purchased. May contain missing or None values.	1, None
Total Spent	The total amount spent on the transaction. Calculated as Quantity * Price Per Unit.	8.00, None
Payment Method	The method of payment used. May contain missing or invalid values.	Cash, Credit Card
Location	The location where the transaction occurred. May contain missing or invalid values.	In-store, Online
Transaction Date	The date of the transaction. Always present and valid.	1/15/2023
Discount Applied	Indicates if a discount was applied to the transaction. May contain missing values.	True, False, None

- Category tables (only 2 / 9 available)
    - Each contains 25 records with corresponding codes, names, and static prices.
    - All category tables follow the same pattern (See table description below)

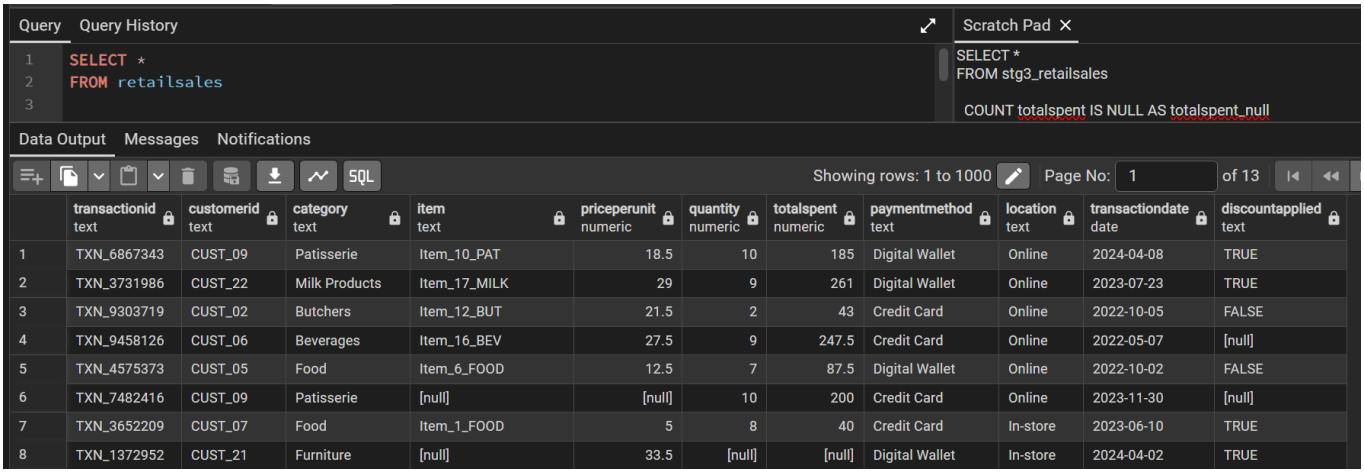
Column Name	Description	Example Values
Item Code	Corresponds to 'item' in retailsales table	Item_13_EHE
Item Name	name of the item purchased	Humidifier
Item Price	Corresponds to 'price per unit' in retailsales table. Static Prices	5.0

Actions Taken

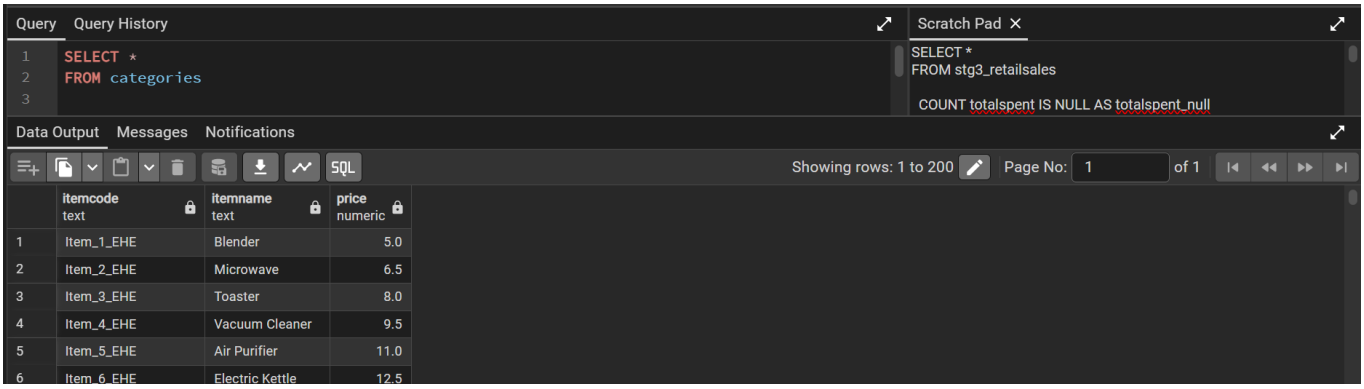
1. Loading & formatting tables
2. Entered all missing data in Categories table, except for 'itemname'. Code obtained from GEMINI.
3. Remove duplicates from both tables
4. Standardizing columns
    a. Remove trailing spaces
    b. Finding values that can be booked together
    c. Change spelling
5. Null handling
6. Relevance of ‘discountapplied’ column in RetailSales table
7. Conclusion

1. Loading & Formatting Tables

First we load the tables into pgadmin. We've loaded both category tables in the same 'categories' table. We've also formatted the columns when importing.



	transactionid text	customerid text	category text	item text	priceperunit numeric	quantity numeric	totalspent numeric	paymentmethod text	location text	transactiondate date	discountapplied text
1	TXN_6867343	CUST_09	Patisserie	Item_10_PAT	18.5	10	185	Digital Wallet	Online	2024-04-08	TRUE
2	TXN_3731986	CUST_22	Milk Products	Item_17_MILK	29	9	261	Digital Wallet	Online	2023-07-23	TRUE
3	TXN_9303719	CUST_02	Butchers	Item_12_BUT	21.5	2	43	Credit Card	Online	2022-10-05	FALSE
4	TXN_9458126	CUST_06	Beverages	Item_16_BEV	27.5	9	247.5	Credit Card	Online	2022-05-07	[null]
5	TXN_4575373	CUST_05	Food	Item_6_FOOD	12.5	7	87.5	Digital Wallet	Online	2022-10-02	FALSE
6	TXN_7482416	CUST_09	Patisserie	[null]	[null]	10	200	Credit Card	Online	2023-11-30	[null]
7	TXN_3652209	CUST_07	Food	Item_1_FOOD	5	8	40	Credit Card	In-store	2023-06-10	TRUE
8	TXN_1372952	CUST_21	Furniture	[null]	33.5	[null]	[null]	Digital Wallet	In-store	2024-04-02	TRUE



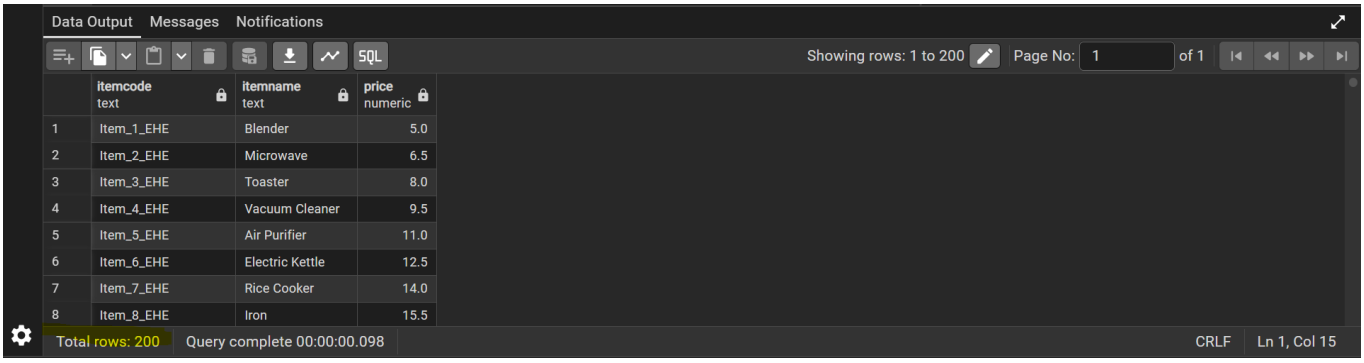
	itemcode text	itemname text	price numeric
1	Item_1_EHE	Blender	5.0
2	Item_2_EHE	Microwave	6.5
3	Item_3_EHE	Toaster	8.0
4	Item_4_EHE	Vacuum Cleaner	9.5
5	Item_5_EHE	Air Purifier	11.0
6	Item_6_EHE	Electric Kettle	12.5

2. Entering missing data in Categories table

Fill in all data excluding 'itemname'. Code from Gemini below.

```
WITH SourceData AS (
    -- Generate the item number (1 to 25) dynamically using ROW_NUMBER()
    SELECT
        price,
        ROW_NUMBER() OVER () AS item_num -- Assigns 1, 2, 3... to each price row
    FROM
        (VALUES
            (5.0), (6.5), (8.0), (9.5), (11.0),
            (12.5), (14.0), (15.5), (17.0), (18.5),
            (20.0), (21.5), (23.0), (24.5), (26.0),
            (27.5), (29.0), (30.5), (32.0), (33.5),
            (35.0), (36.5), (38.0), (39.5), (41.0)
        ) AS t(price) -- Alias 't' is required, price is the column name for the values
),
Suffixes (suffix) AS (
    VALUES
        ('CEA'), ('FOOD'), ('BEV'), ('MILK'), ('PAT'), ('BUT')
)
INSERT INTO categories (itemcode, itemname, price)
SELECT
```

```
    -- Use the calculated item_num, converting it to text
    'Item_' || sd.item_num::TEXT || '_' || s.suffix,
    NULL,
    sd.price
FROM
    SourceData sd
CROSS JOIN
    Suffixes s;
```



Data Output | Messages | Notifications

Showing rows: 1 to 200 | Page No: 1 of 1

	itemcode text	itemname text	price numeric
1	Item_1_EHE	Blender	5.0
2	Item_2_EHE	Microwave	6.5
3	Item_3_EHE	Toaster	8.0
4	Item_4_EHE	Vacuum Cleaner	9.5
5	Item_5_EHE	Air Purifier	11.0
6	Item_6_EHE	Electric Kettle	12.5
7	Item_7_EHE	Rice Cooker	14.0
8	Item_8_EHE	Iron	15.5

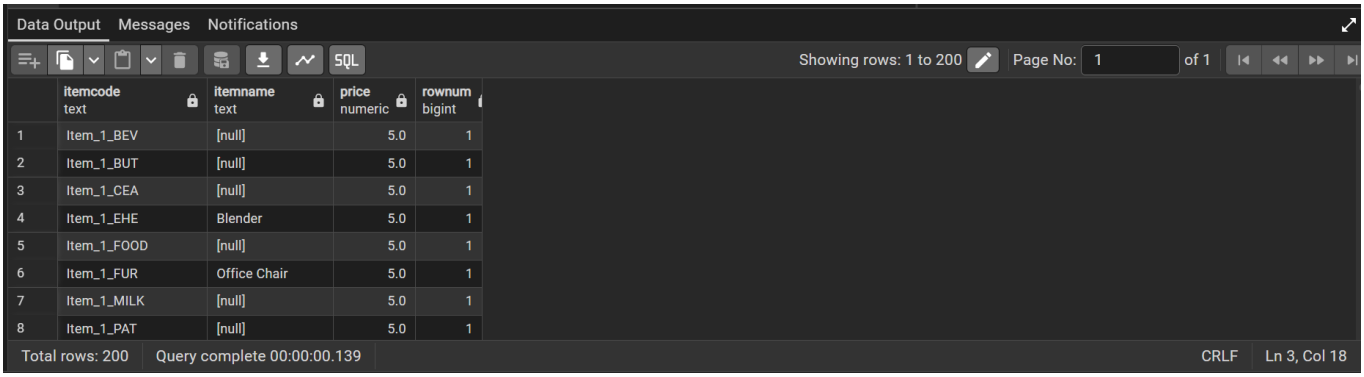
Total rows: 200 | Query complete 00:00:00.098 | CRLF | Ln 1, Col 15

3. Removing Duplicates

We created a staging categories table as we will be deleting rows, and add a ROW_NUMBER col grouped by itemcode (has distinct values only).

```
CREATE TABLE stg_categories AS (
    SELECT *,
    ROW_NUMBER () OVER ( PARTITION BY itemcode) AS rownum
    FROM categories
    );
```

The table below is ORDER BY rownum DESC



Data Output | Messages | Notifications

Showing rows: 1 to 200 | Page No: 1 of 1

	itemcode text	itemname text	price numeric	rownum bigint
1	Item_1_BEV	[null]	5.0	1
2	Item_1_BUT	[null]	5.0	1
3	Item_1_CEA	[null]	5.0	1
4	Item_1_EHE	Blender	5.0	1
5	Item_1_FOOD	[null]	5.0	1
6	Item_1_FUR	Office Chair	5.0	1
7	Item_1_MILK	[null]	5.0	1
8	Item_1_PAT	[null]	5.0	1

Total rows: 200 | Query complete 00:00:00.139 | CRLF | Ln 3, Col 18

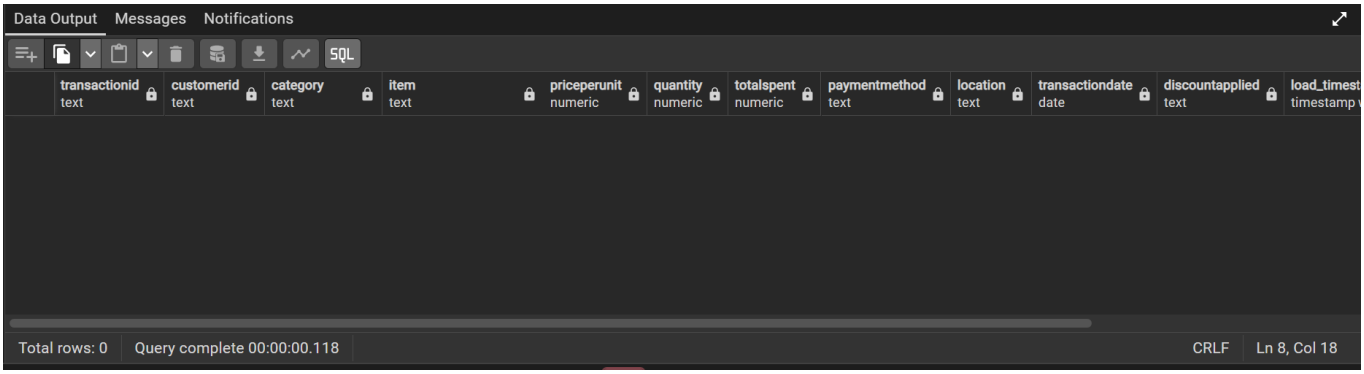
There are no duplicates, so we remove the rownum col.

```
ALTER TABLE stg_categories
DROP COLUMN rownum;
```

Now we create a staging retailsales table, and add a ROW_NUMBER col grouped by transactionid(has distinct values only) and check for duplicates. The queries were run separately

```
 CREATE TABLE stg_retailsales AS
```

```sql
SELECT *,
    NOW() AS load_timestamp,
    'NA' AS source_system_name,
    ROW_NUMBER() OVER (PARTITION BY transactionid ) AS rownum
FROM retailsales;

SELECT *
FROM stg_retailsales
WHERE rownum > 1;
```



There are no duplicates. Delete rownum.

```sql
ALTER TABLE stg_retailsales,
DROP COLUMN rownum;
```

4. Standardizing Data

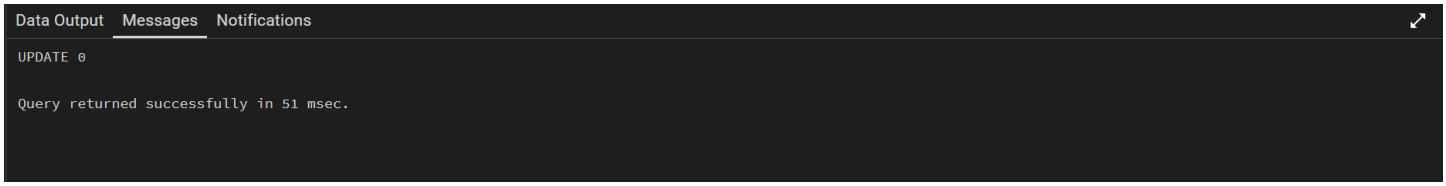
Create another staging retailsales table, as we will be making a lot of permanent changes. All columns are selected. We didn't use SELECT * to avoid copying the load_timestamp and source_systemname from the previous staging table.

```sql
CREATE TABLE stg2_retailsales AS
    SELECT
        transactionid,
        customerid,
        category,
        item,
        priceperunit,
        quantity,
        totalspent,
        paymentmethod,
        location,
        transactiondate,
        discountapplied,
        NOW() AS load_timestamp,
        'NA' AS source_system_name
    FROM stg_retailsales;
```

Now we remove all the trailing spaces from our stg2_retailsales table.

```sql
UPDATE stg2_retailsales
SET
    transactionid = TRIM(transactionid),
```

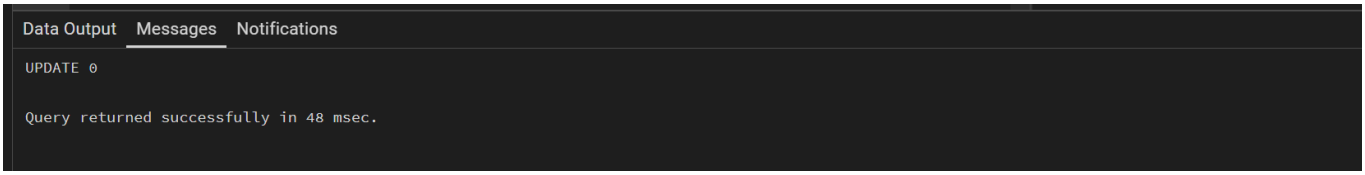
```
    customerid = TRIM(customerid),
    category = TRIM(category),
    item = TRIM(item),
    paymentmethod = TRIM(paymentmethod),
    location = TRIM(location),
    discountapplied = TRIM(discountapplied)
WHERE
    transactionid LIKE ' %' OR transactionid LIKE '% ' OR
    customerid LIKE ' %' OR customerid LIKE '% ' OR
    category LIKE ' %' OR category LIKE '% ' OR
    item LIKE ' %' OR item LIKE '% ' OR
    paymentmethod LIKE ' %' OR paymentmethod LIKE '% ' OR
    location LIKE ' %' OR location LIKE '% ' OR
    discountapplied LIKE ' %' OR discountapplied LIKE '% ';
```



The output message shows that there are no text columns with trailing spaces.

We also created another staging table for categories and trimmed the columns. Likewise, there are no trailing spaces in text columns.

```
-- create stg2_categories
CREATE TABLE stg2_categories AS
SELECT
    itemcode,
    itemname,
    price,
    NOW() AS load_timestamp,
    'NA' AS source_system_name
FROM stg_categories;
-- trim stg2_categories
UPDATE stg2_categories
SET
    itemcode = TRIM(itemcode),
    itemname = TRIM(itemname)
WHERE
    itemcode LIKE '% ' OR itemcode LIKE ' %' OR
    itemname LIKE '% ' OR itemname LIKE ' %';
```

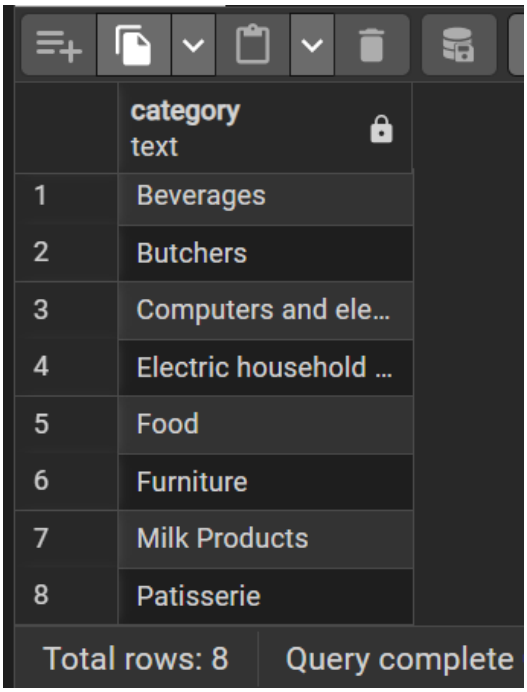


We also looked for records that can be grouped together, ie ewallet and GooglePay. We ran each of the queries below separately.

```
-- category column
```

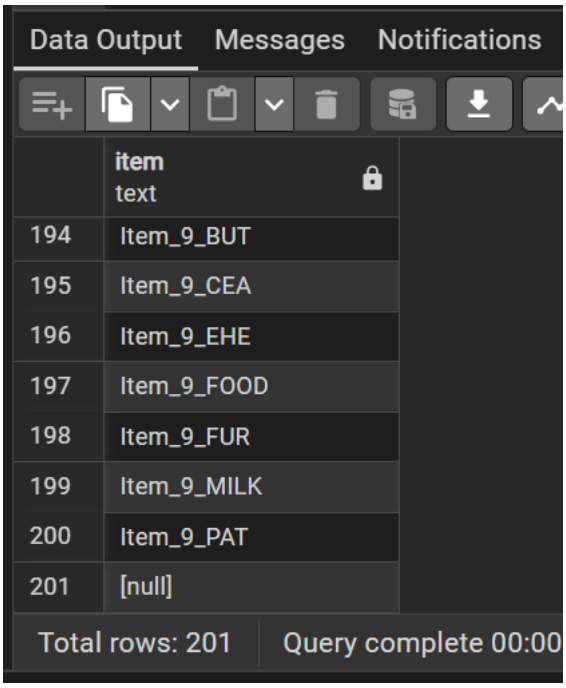
```sql
SELECT DISTINCT(category)
FROM stg2_retailsales
ORDER BY category;
-- paymentmethod
SELECT DISTINCT(paymentmethod)
FROM stg2_retailsales;
-- location
SELECT DISTINCT(location)
FROM stg2_retailsales;
-- item
SELECT DISTINCT (item)
FROM stg2_retailsales
ORDER BY item;
-- discountapplied
SELECT DISTINCT (discountapplied)
FROM stg2_retailsales;
```

Example Outputs are as shown below. We've notice some spelling errors.



	category text
1	Beverages
2	Butchers
3	Computers and ele...
4	Electric household ...
5	Food
6	Furniture
7	Milk Products
8	Patisserie

Total rows: 8 Query complete



Data Output Messages Notifications

	item text
194	Item_9_BUT
195	Item_9_CEA
196	Item_9_EHE
197	Item_9_FOOD
198	Item_9_FUR
199	Item_9_MILK
200	Item_9_PAT
201	[null]

Total rows: 201 Query complete 00:00

We made a spelling change as shown below

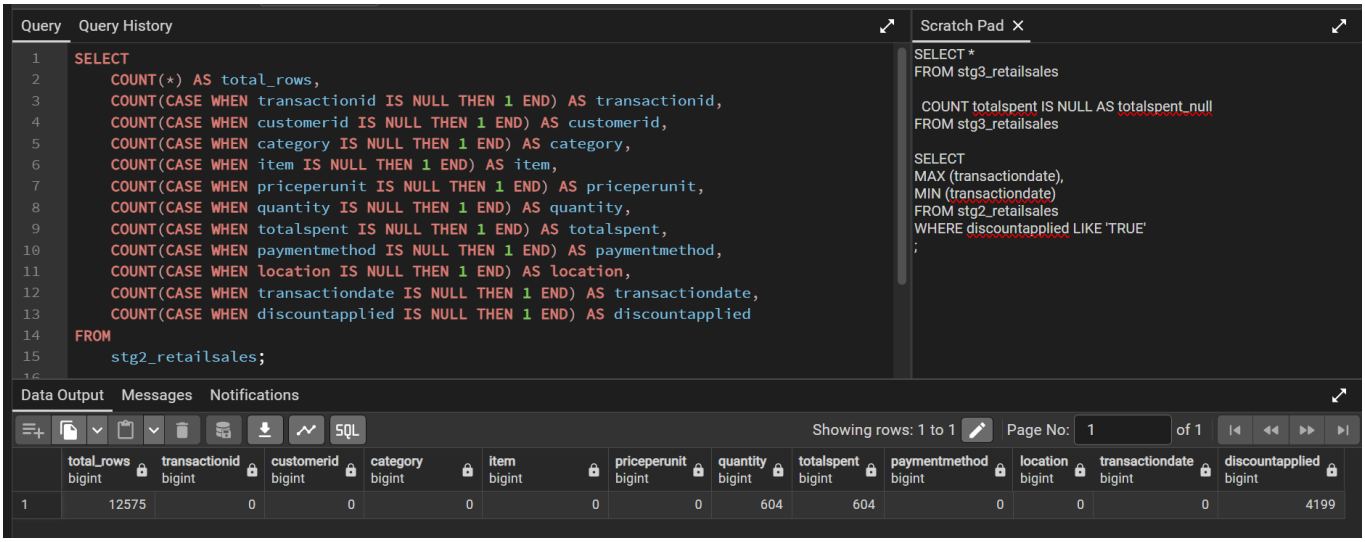
```sql
UPDATE stg2_retailsales
SET category = 'Milk products'
WHERE category = 'Milk Products';
```

We also changed the data type for quantity from numerical to integer

```sql
ALTER TABLE stg2_retailsales
ALTER COLUMN quantity TYPE INTEGER
```

5. Null Handling

Next we found which columns in stg2_retailsales & stg2_categories had nulls, by counting the number or null records.



```
SELECT
    COUNT(*) AS total_rows,
    COUNT(CASE WHEN transactionid IS NULL THEN 1 END) AS transactionid,
    COUNT(CASE WHEN customerid IS NULL THEN 1 END) AS customerid,
    COUNT(CASE WHEN category IS NULL THEN 1 END) AS category,
    COUNT(CASE WHEN item IS NULL THEN 1 END) AS item,
    COUNT(CASE WHEN priceperunit IS NULL THEN 1 END) AS priceperunit,
    COUNT(CASE WHEN quantity IS NULL THEN 1 END) AS quantity,
    COUNT(CASE WHEN totalspent IS NULL THEN 1 END) AS totalspent,
    COUNT(CASE WHEN paymentmethod IS NULL THEN 1 END) AS paymentmethod,
    COUNT(CASE WHEN location IS NULL THEN 1 END) AS location,
    COUNT(CASE WHEN transactiondate IS NULL THEN 1 END) AS transactiondate,
    COUNT(CASE WHEN discountapplied IS NULL THEN 1 END) AS discountapplied
FROM
    stg2_retailsales;
```

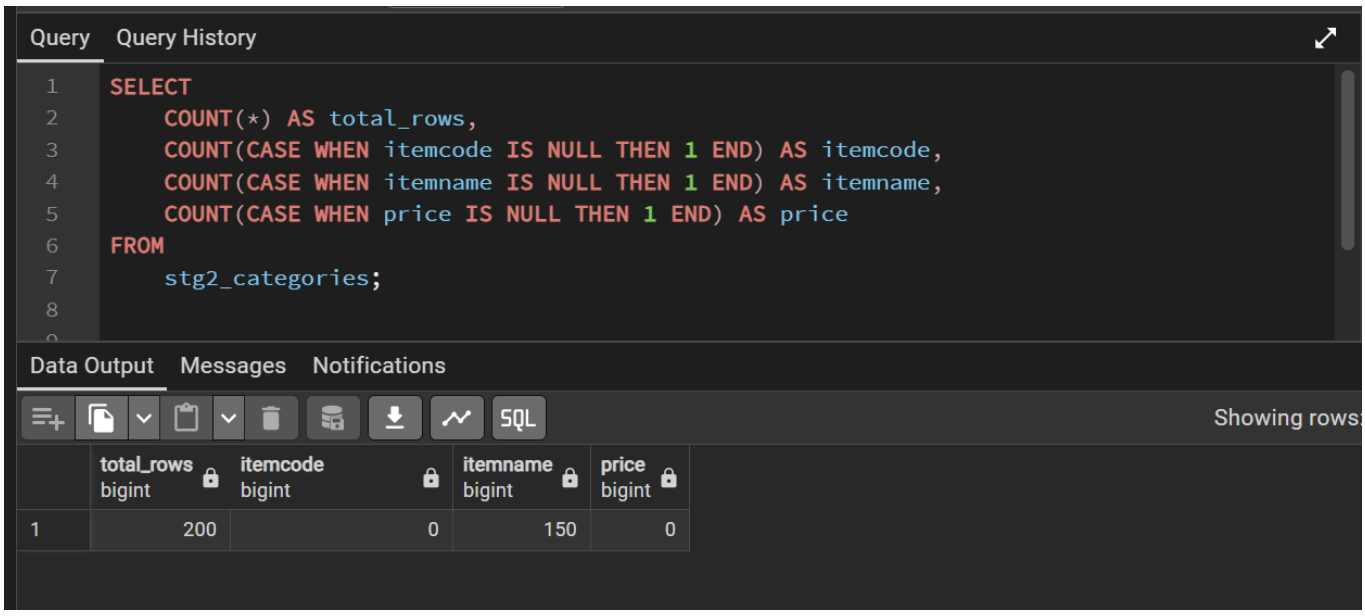
Scratch Pad

```
SELECT *
FROM stg3_retailsales

 COUNT totalspent IS NULL AS totalspent_null
FROM stg3_retailsales

SELECT
MAX (transactiondate),
MIN (transactiondate)
FROM stg2_retailsales
WHERE discountapplied LIKE 'TRUE'
;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	total_rows bigint	transactionid bigint	customerid bigint	category bigint	item bigint	priceperunit bigint	quantity bigint	totalspent bigint	paymentmethod bigint	location bigint	transactiondate bigint	discountapplied bigint
1	12575	0	0	0	0	0	604	604	0	0	0	4199



```
SELECT
    COUNT(*) AS total_rows,
    COUNT(CASE WHEN itemcode IS NULL THEN 1 END) AS itemcode,
    COUNT(CASE WHEN itemname IS NULL THEN 1 END) AS itemname,
    COUNT(CASE WHEN price IS NULL THEN 1 END) AS price
FROM
    stg2_categories;
```

Data Output | Messages | Notifications

Showing rows:

	total_rows bigint	itemcode bigint	itemname bigint	price bigint
1	200	0	150	0

We added a ‘itemcategory’ column to stg2_categories. This will help us fetch the other 2 columns to update the stg2_retailsales table

```
-- addcol stg2_categories.itemcategory
ALTER TABLE stg2_categories
ADD COLUMN itemcategory text;
-- update stg2_categories.itemcategory
UPDATE stg2_categories
SET itemcategory = stg2_retailsales.category
FROM stg2_retailsales
WHERE itemcode = stg2_retailsales.item;
-- view table
SELECT *
FROM stg2_categories
```

Data Output | Messages | Notifications

Showing rows: 1 to 200

	itemcode text	itemname text	price numeric	load_timestamp timestamp with time zone	source_system_name text	itemcategory text
1	Item_10_PAT	[null]	18.5	2025-11-11 11:31:23.840706+...	NA	Patisserie
2	Item_12_BUT	[null]	21.5	2025-11-11 11:31:23.840706+...	NA	Butchers
3	Item_16_BEV	[null]	27.5	2025-11-11 11:31:23.840706+...	NA	Beverages
4	Item_6_FOOD	[null]	12.5	2025-11-11 11:31:23.840706+...	NA	Food
5	Item_1_FOOD	[null]	5.0	2025-11-11 11:31:23.840706+...	NA	Food
6	Item_16_FUR	Shoe Rack	27.5	2025-11-11 11:31:23.840706+...	NA	Furniture
7	Item_22_BUT	[null]	36.5	2025-11-11 11:31:23.840706+...	NA	Butchers
8	Item_3_BUT	[null]	8.0	2025-11-11 11:31:23.840706+...	NA	Butchers

Total rows: 200 | Query complete 00:00:00.099

We also checked for nulls in the newly updated itemcategory column

Query | Query History

```
1  SELECT
2      COUNT(CASE WHEN itemcategory IS NULL THEN 1 END) AS itemcategory_nullcount,
3      COUNT(CASE WHEN itemcode IS NULL THEN 1 END)AS itemcode_nullcount,
4      COUNT (CASE WHEN price IS NULL THEN 1 END) AS price_nullcount
5  FROM stg2_categories
6
```

Data Output | Messages | Notifications

Showing rows: 1

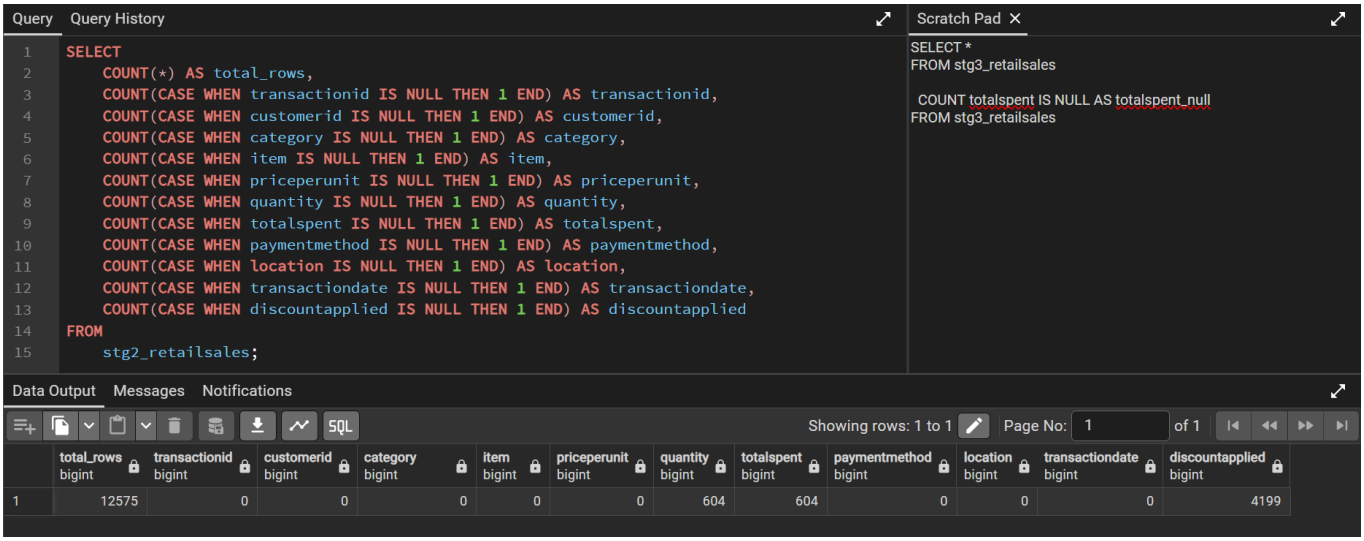
	itemcategory_nullcount bigint	itemcode_nullcount bigint	price_nullcount bigint
1	0	0	0

Since there are no nulls in itemcategories, we proceeded to use the stg2_categories table to fill up the stg2_retailsales table. We ran these queries separately.

```sql
--update stg2_retailsales.quantity
UPDATE stg2_retailsales
SET quantity = totalspent / priceperunit
WHERE quantity IS NULL;
--update stg2_retailsales.totalspent
UPDATE stg2_retailsales
SET totalspent = priceperunit * quantity
WHERE totalspent IS NULL;
-- update stg2_retailsales.priceperunit
UPDATE stg_retailsales
SET priceperunit = totalspent/quantity
WHERE priceperunit IS NULL
--update stg2_retailsales.item
UPDATE stg2_retailsales AS r
SET item = c.itemcode
FROM stg2_categories AS c
WHERE r.priceperunit = c.price
AND r.category = c.itemcategory
```

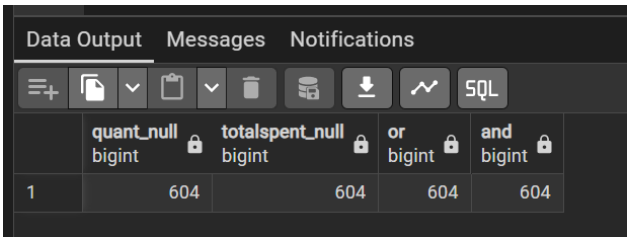
```
AND r.item IS NULL
```

After that, we checked the stg2_retailsales table for nulls again.



Quantity, totalspent, discountapplied has nulls. We checked to see if there were rows with both quantity & totalspent as null.

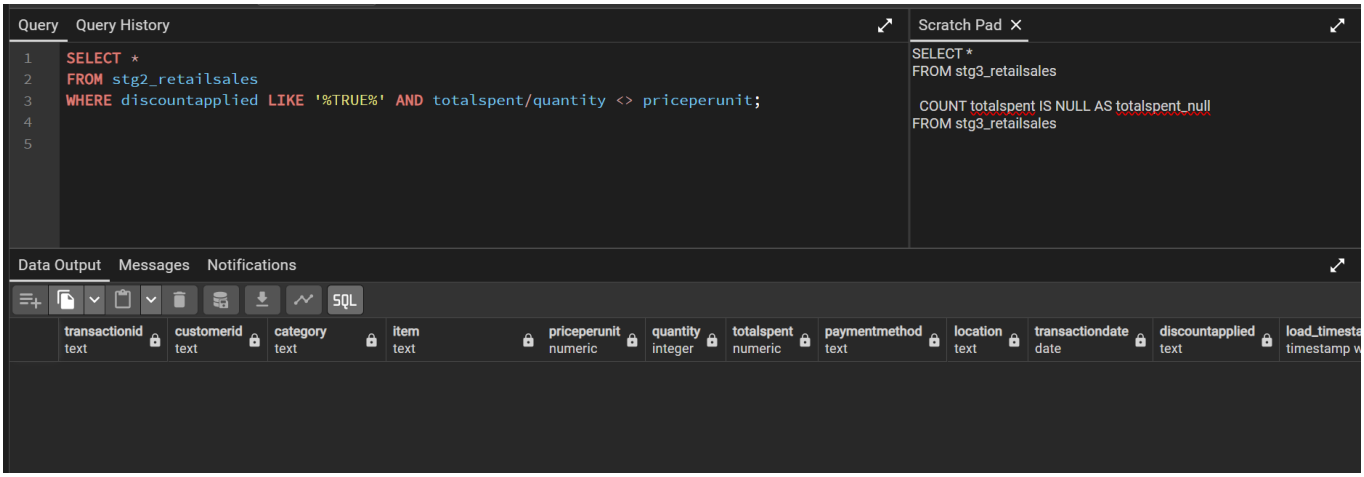
```
SELECT
    COUNT (CASE WHEN quantity IS NULL THEN 1 END) AS quant_null,
    COUNT (CASE WHEN totalspent IS NULL THEN 1 END) AS totalspent_null,
    COUNT (CASE WHEN quantity IS NULL or totalspent is null then 1 end) as OR,
    COUNT (CASE WHEN quantity IS NULL AND totalspent IS NULL then 1 end) AS AND
FROM stg2_retailsales
```



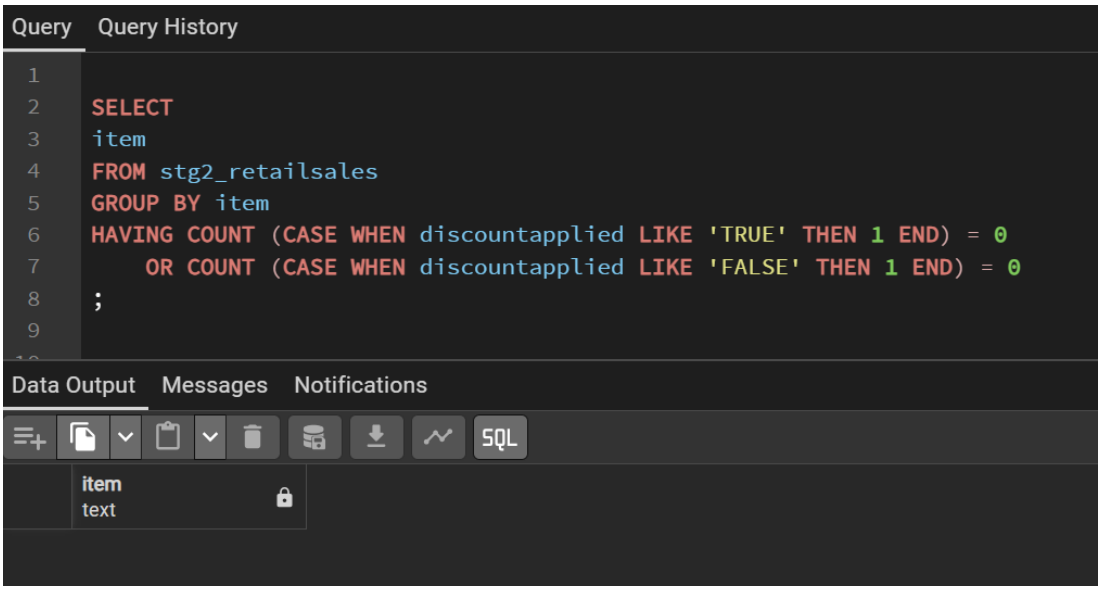
There are nulls for both columns in the AND section of the query.

6. Relevance of ‘discountapplied’ column in RetailSales table

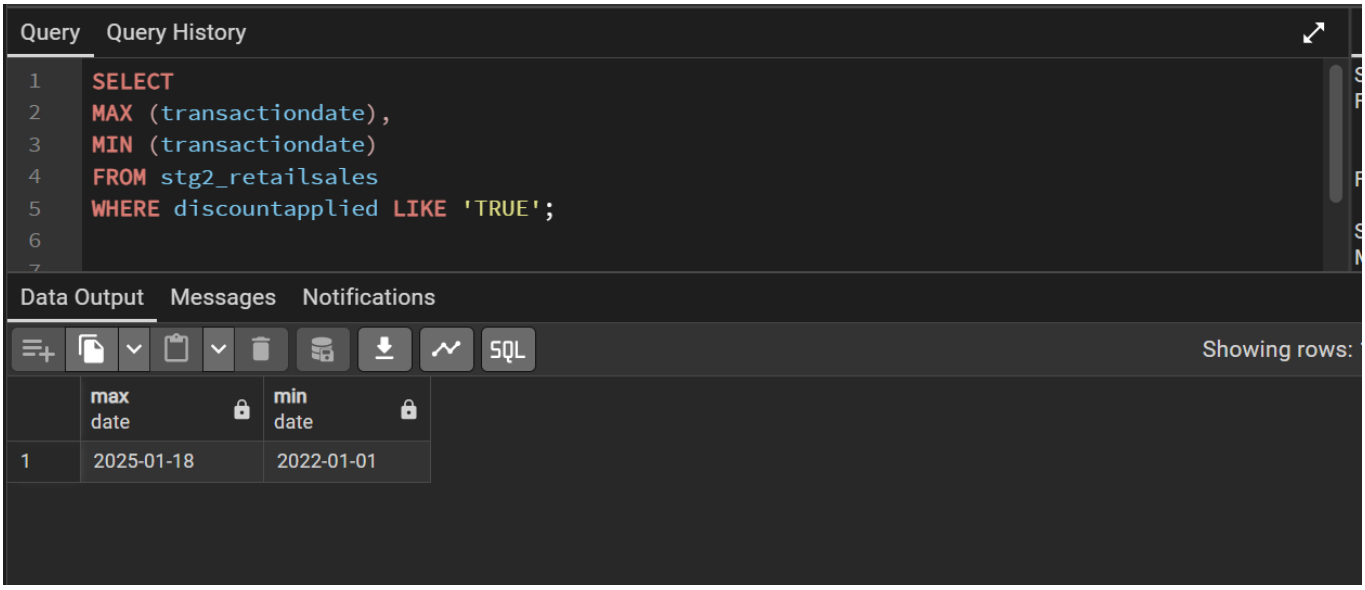
We checked if we can use totalspent/quantity <> priceperunit can help us determine which customers used discounts.



The output shows that totalspent and price per unit does not indicate that the discount is used. It is possible that this table just does not list discounted prices. Next, we tried testing if discounts are only applicable for certain items.



We also checked to see it discountapplied is affected by transaction date range



```sql
SELECT
MAX (transactiondate),
MIN (transactiondate)
FROM stg2_retailsales
WHERE discountapplied LIKE 'FALSE';
```

	max date	min date
1	2025-01-18	2022-01-01

Discountapplied is not affected by transactiondate.

Discountapplied is not affected by paymentmethod either, as shown below

```sql
SELECT paymentmethod,
discountapplied
FROM stg2_retailsales;
```

	paymentmethod text	discountapplied text
1	Digital Wallet	TRUE
2	Credit Card	FALSE
3	Credit Card	[null]
4	Digital Wallet	FALSE
5	Credit Card	TRUE
6	Credit Card	FALSE
7	Cash	FALSE
8	Cash	TRUE
9	Cash	TRUE

7. Conclusion

```sql
SELECT
    COUNT(*) AS total_rows,
    COUNT(CASE WHEN transactionid IS NULL THEN 1 END) AS transactionid,
    COUNT(CASE WHEN customerid IS NULL THEN 1 END) AS customerid,
    COUNT(CASE WHEN category IS NULL THEN 1 END) AS category,
    COUNT(CASE WHEN item IS NULL THEN 1 END) AS item,
    COUNT(CASE WHEN priceperunit IS NULL THEN 1 END) AS priceperunit,
    COUNT(CASE WHEN quantity IS NULL THEN 1 END) AS quantity,
    COUNT(CASE WHEN totalspent IS NULL THEN 1 END) AS totalspent,
    COUNT(CASE WHEN paymentmethod IS NULL THEN 1 END) AS paymentmethod,
    COUNT(CASE WHEN location IS NULL THEN 1 END) AS location,
    COUNT(CASE WHEN transactiondate IS NULL THEN 1 END) AS transactiondate,
    COUNT(CASE WHEN discountapplied IS NULL THEN 1 END) AS discountapplied
FROM
    stg2_retailsales;
```

```sql
SELECT *
FROM stg3_retailsales

 COUNT totalspent IS NULL AS totalspent_null
FROM stg3_retailsales
```

	total_rows bigint	transactionid bigint	customerid bigint	category bigint	item bigint	priceperunit bigint	quantity bigint	totalspent bigint	paymentmethod bigint	location bigint	transactiondate bigint	discountapplied bigint
1	12575	0	0	0	0	0	604	604	0	0	0	4199

There are still null values in the quantity, totalspent and discounts columns in the retailsales table.

1. Records with nulls in the quantity column also have nulls in the totalspent column. It is not possible to derive the missing values anywhere else.
2. It is possible that discounts have been applied, but this table does not list discounted prices.