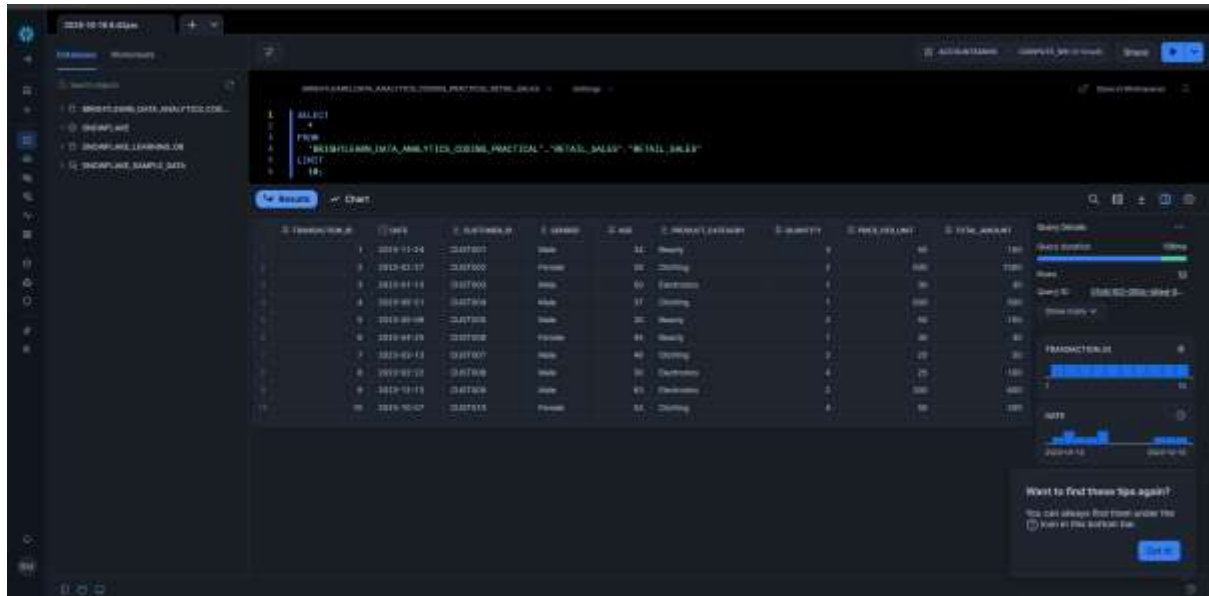


BrightLearn Data Analytics Coding Practical

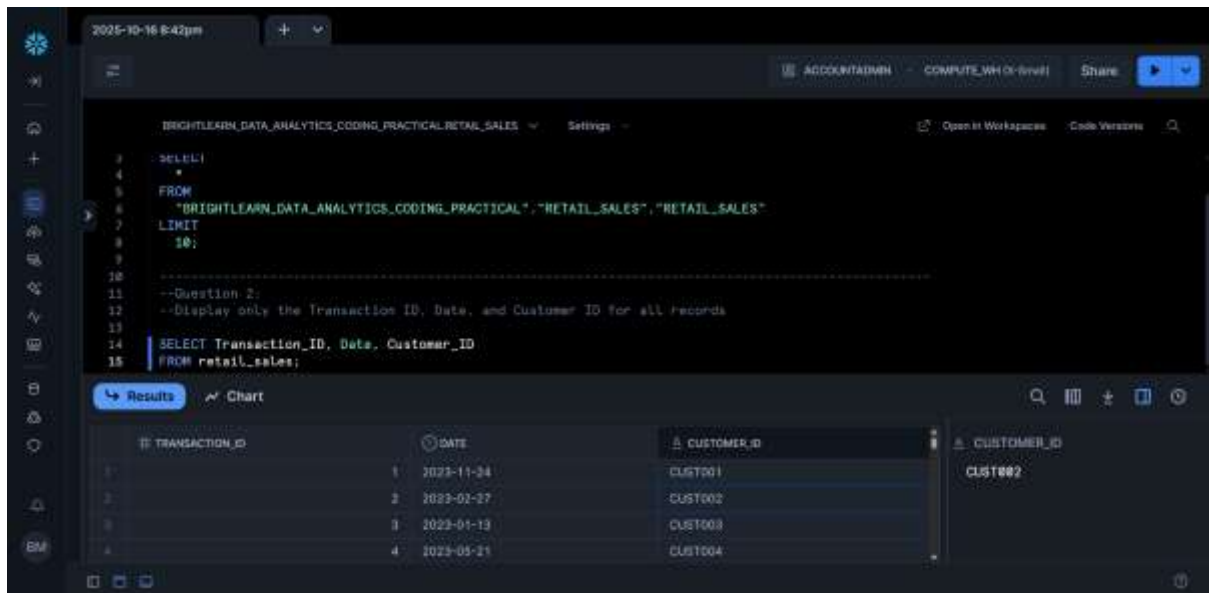
Practical 1: SQL Fundamentals (Snowflake-Basic SQL Syntax)

1. SELECT STATEMENT

Q1. Display all columns for all transactions. Expected output: All columns

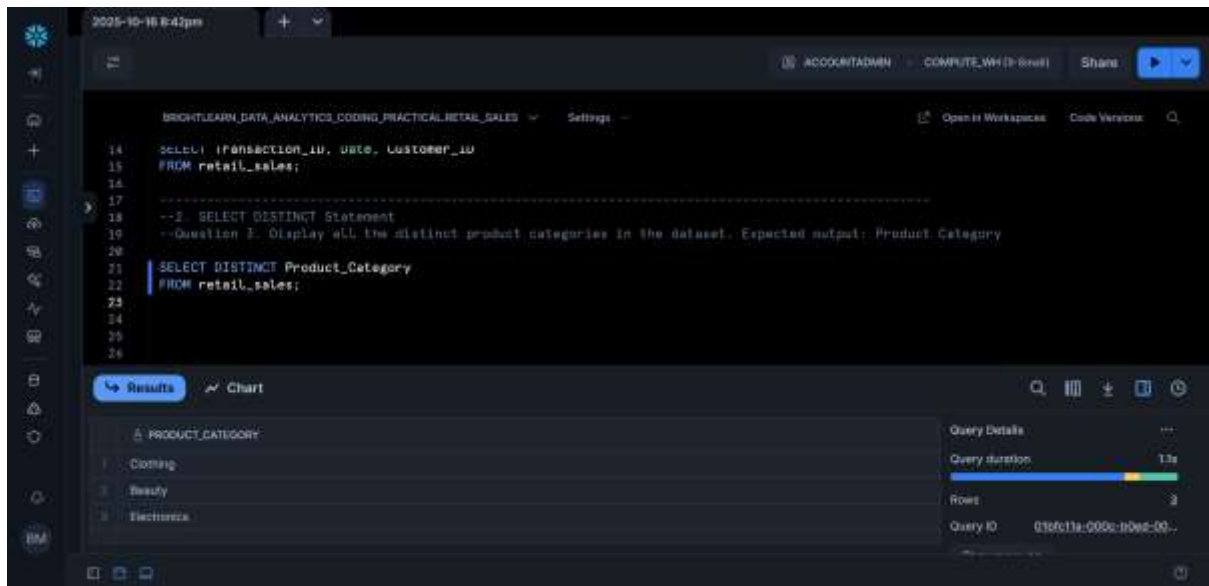


Q2. Display only the Transaction ID, Date, and Customer ID for all records. *Expected output:* Transaction ID, Date, Customer ID



2. SELECT DISTINCT STATEMENT

Q3. Display all the distinct product categories in the dataset. *Expected output:* Product Category



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
14 SELECT transaction_id, date, customer_id
15 FROM retail_sales;
16
17 -----
18 --2. SELECT DISTINCT Statement
19 --Question 3. Display all the distinct product categories in the dataset. Expected output: Product Category
20
21 SELECT DISTINCT Product_Category
22 FROM retail_sales;
23
24
25
26
```

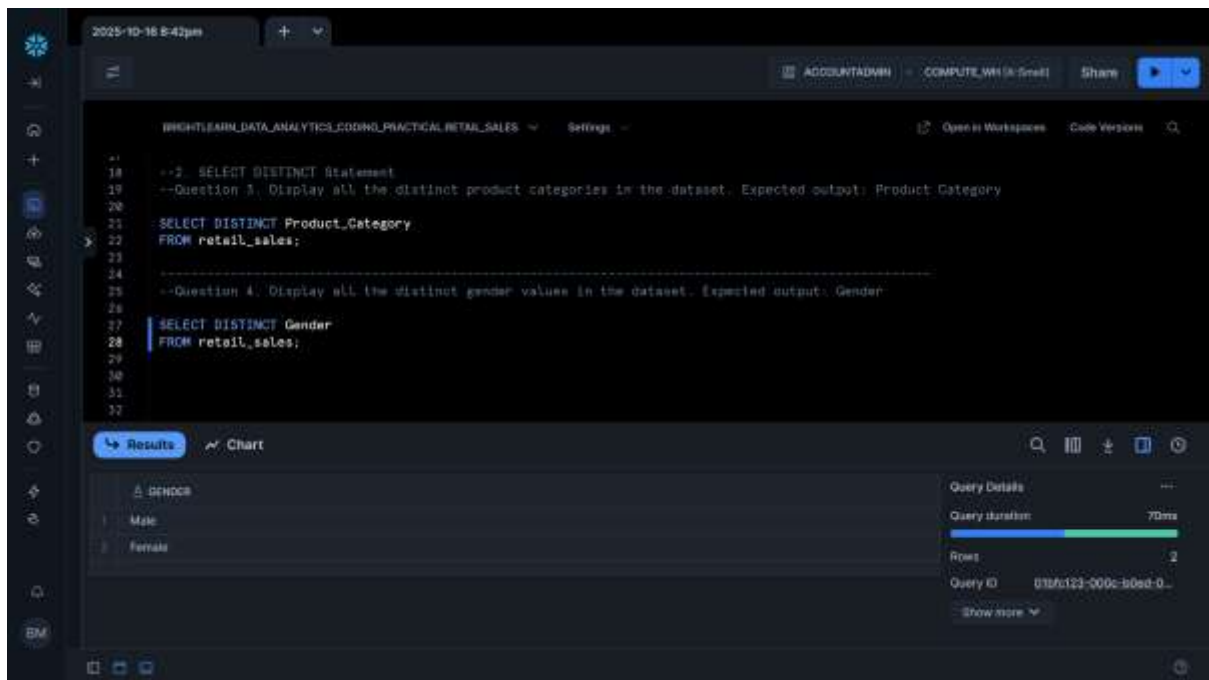
The results pane displays the output of the query, showing a table with one column, **PRODUCT_CATEGORY**, and three rows of data:

PRODUCT_CATEGORY
Clothing
Beauty
Electronics

On the right side of the results pane, the **Query Details** section shows the following information:

- Query duration: 1.3s
- Rows: 3
- Query ID: 016fc13a-000c-b06d-00...

Q4. Display all the distinct gender values in the dataset. *Expected output:* Gender



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
18 --2. SELECT DISTINCT Statement
19 --Question 3. Display all the distinct product categories in the dataset. Expected output: Product Category
20
21 SELECT DISTINCT Product_Category
22 FROM retail_sales;
23
24 -----
25 --Question 4. Display all the distinct gender values in the dataset. Expected output: Gender
26
27 SELECT DISTINCT Gender
28 FROM retail_sales;
29
30
31
32
```

The results pane displays the output of the query, showing a table with one column, **GENDER**, and two rows of data:

GENDER
Male
Female

On the right side of the results pane, the **Query Details** section shows the following information:

- Query duration: 70ms
- Rows: 2
- Query ID: 016fc123-000c-b06d-00...

3. WHERE CLAUSE

Q5. Display all transactions where the Age is greater than 40. Expected output: All columns

The screenshot shows a SQL IDE interface with a query editor and a results table. The query editor contains the following SQL code:

```
--Question 4. Display all the distinct gender values in the dataset. Expected output: Gender
SELECT DISTINCT Gender
FROM retail_sales;

-----
-- 3. WHERE Clause
-- Question 5. Display all transactions where the Age is greater than 40. Expected output: All columns
SELECT *
FROM retail_sales
WHERE Age > 40;
```

The results table displays the following data:

TRANSACTION_ID	DATE	CUSTOMER_ID	GENDER	AGE	PRODUCT_CATEGORY	QUANTITY	PRICE_PER_UNIT	TOTAL_AMOUNT
1	2023-01-13	CUST003	Male	50	Electronics	1	30	30
2	2023-04-25	CUST006	Female	45	Beauty	1	30	30
3	2023-03-13	CUST007	Male	48	Clothing	2	25	50
4	2023-12-13	CUST009	Male	63	Electronics	2	300	600
5	2023-10-07	CUST010	Female	52	Clothing	4	50	200
14	2023-07-17	CUST014	Male	64	Clothing	4	30	120
15	2023-01-16	CUST015	Female	42	Electronics	4	500	2000

Q6. Display all transactions where the Price per Unit is between 100 and 500. Expected output: All columns

The screenshot shows a SQL IDE interface with a query editor and a results table. The query editor contains the following SQL code:

```
--Question 4. Display all the distinct gender values in the dataset. Expected output: Gender
SELECT DISTINCT Gender
FROM retail_sales;

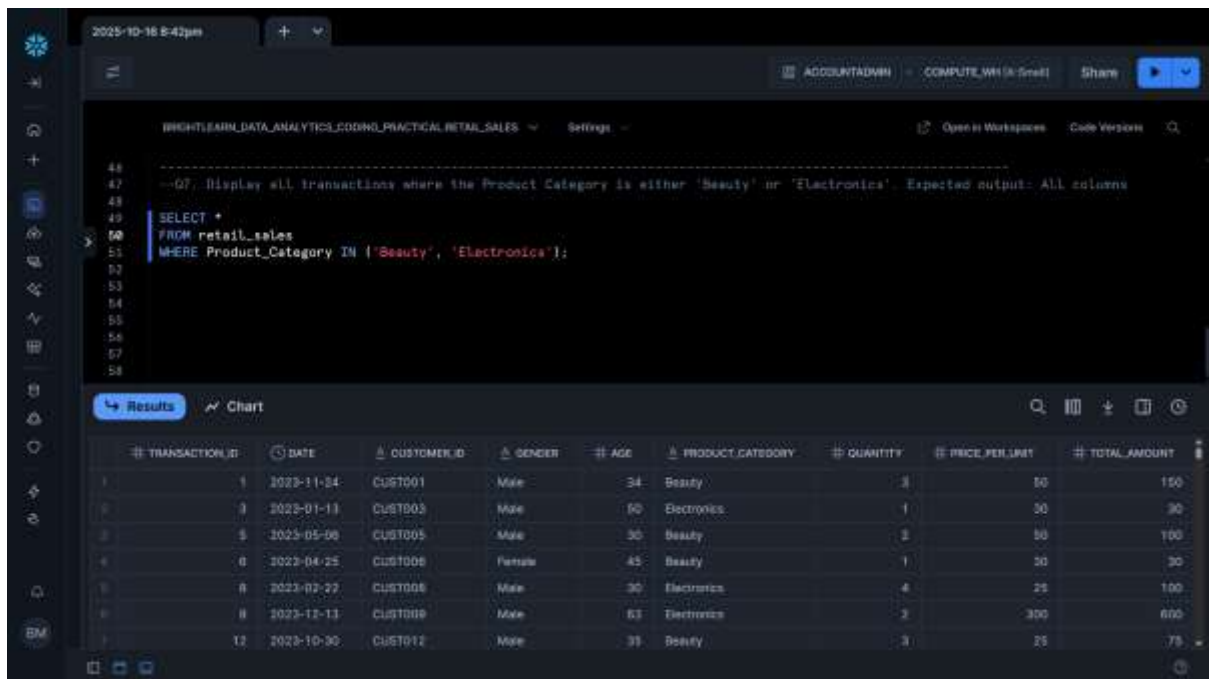
-----
-- 3. WHERE Clause
-- Question 5. Display all transactions where the Age is greater than 40. Expected output: All columns
SELECT *
FROM retail_sales
WHERE Age > 40;

-----
--Q6. Display all transactions where the Price per Unit is between 100 and 500. Expected output: All columns
SELECT *
FROM retail_sales
WHERE Price_per_Unit BETWEEN 100 AND 500;
```

The results table displays the following data:

TRANSACTION_ID	DATE	CUSTOMER_ID	GENDER	AGE	PRODUCT_CATEGORY	QUANTITY	PRICE_PER_UNIT	TOTAL_AMOUNT
1	2023-02-27	CUST002	Female	26	Clothing	2	500	1000
2	2023-05-21	CUST004	Male	37	Clothing	1	500	500
3	2023-12-13	CUST009	Male	63	Electronics	2	300	600
4	2023-08-05	CUST013	Male	22	Electronics	3	500	1500
15	2023-01-16	CUST015	Female	42	Electronics	4	500	2000
16	2023-02-17	CUST016	Male	16	Clothing	3	500	1500
20	2023-11-05	CUST020	Male	22	Clothing	3	300	900

Q7. Display all transactions where the Product Category is either 'Beauty' or 'Electronics'. *Expected output:* All columns



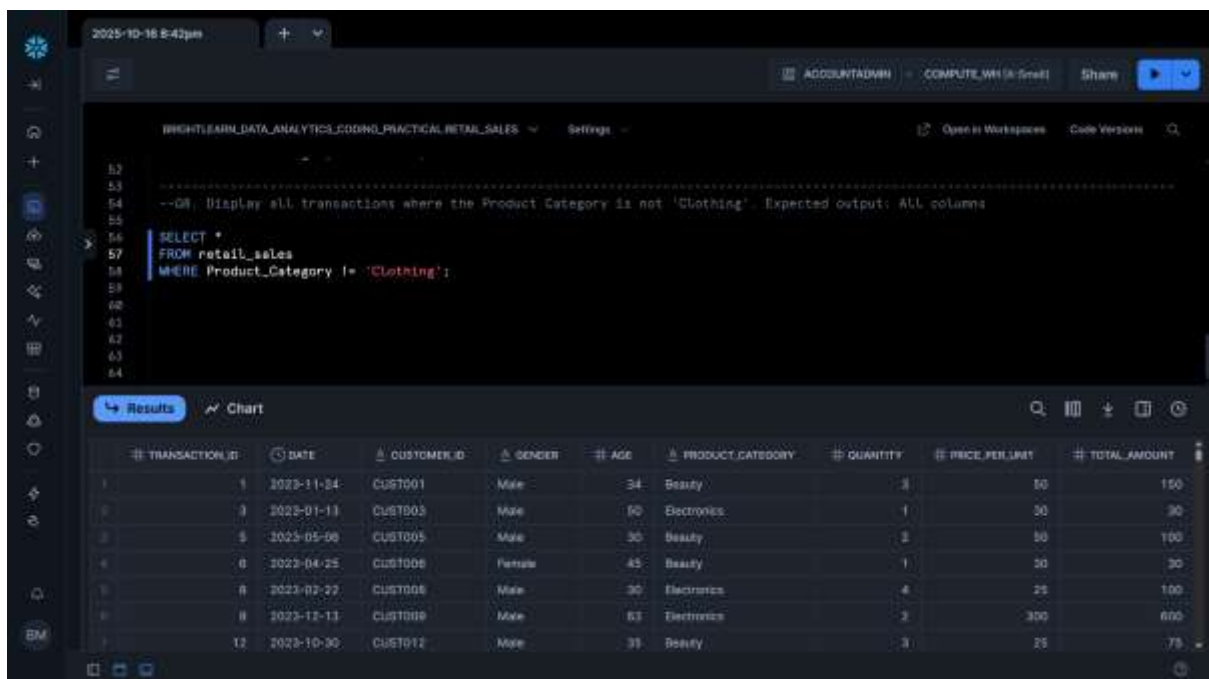
The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
--Q7, Display all transactions where the Product Category is either 'Beauty' or 'Electronics'. Expected output: All columns
SELECT *
FROM retail_sales
WHERE Product_Category IN ('Beauty', 'Electronics');
```

The results pane displays a table with the following data:

TRANSACTION_ID	DATE	CUSTOMER_ID	GENDER	AGE	PRODUCT_CATEGORY	QUANTITY	PRICE_PER_UNIT	TOTAL_AMOUNT
1	2023-11-24	CUST001	Male	34	Beauty	3	50	150
3	2023-01-13	CUST003	Male	60	Electronics	1	30	30
5	2023-05-08	CUST005	Male	30	Beauty	2	50	100
6	2023-04-25	CUST006	Female	45	Beauty	1	30	30
8	2023-02-22	CUST008	Male	30	Electronics	4	25	100
9	2023-12-13	CUST009	Male	63	Electronics	2	300	600
12	2023-10-30	CUST012	Male	39	Beauty	3	25	75

Q8. Display all transactions where the Product Category is not 'Clothing'. *Expected output:* All columns



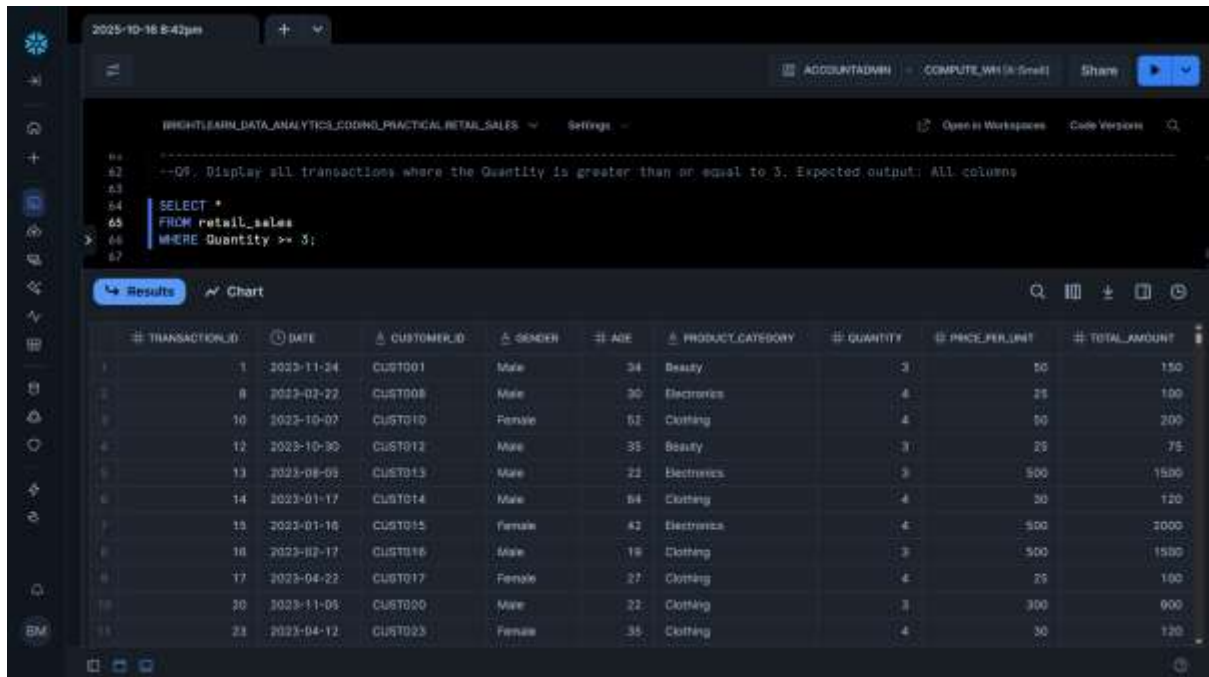
The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
--Q8, Display all transactions where the Product Category is not 'Clothing'. Expected output: All columns
SELECT *
FROM retail_sales
WHERE Product_Category != 'Clothing';
```

The results pane displays a table with the following data:

TRANSACTION_ID	DATE	CUSTOMER_ID	GENDER	AGE	PRODUCT_CATEGORY	QUANTITY	PRICE_PER_UNIT	TOTAL_AMOUNT
1	2023-11-24	CUST001	Male	34	Beauty	3	50	150
3	2023-01-13	CUST003	Male	60	Electronics	1	30	30
5	2023-05-08	CUST005	Male	30	Beauty	2	50	100
6	2023-04-25	CUST006	Female	45	Beauty	1	30	30
8	2023-02-22	CUST008	Male	30	Electronics	4	25	100
9	2023-12-13	CUST009	Male	63	Electronics	2	300	600
12	2023-10-30	CUST012	Male	39	Beauty	3	25	75

Q9. Display all transactions where the Quantity is greater than or equal to 3. Expected output: All columns

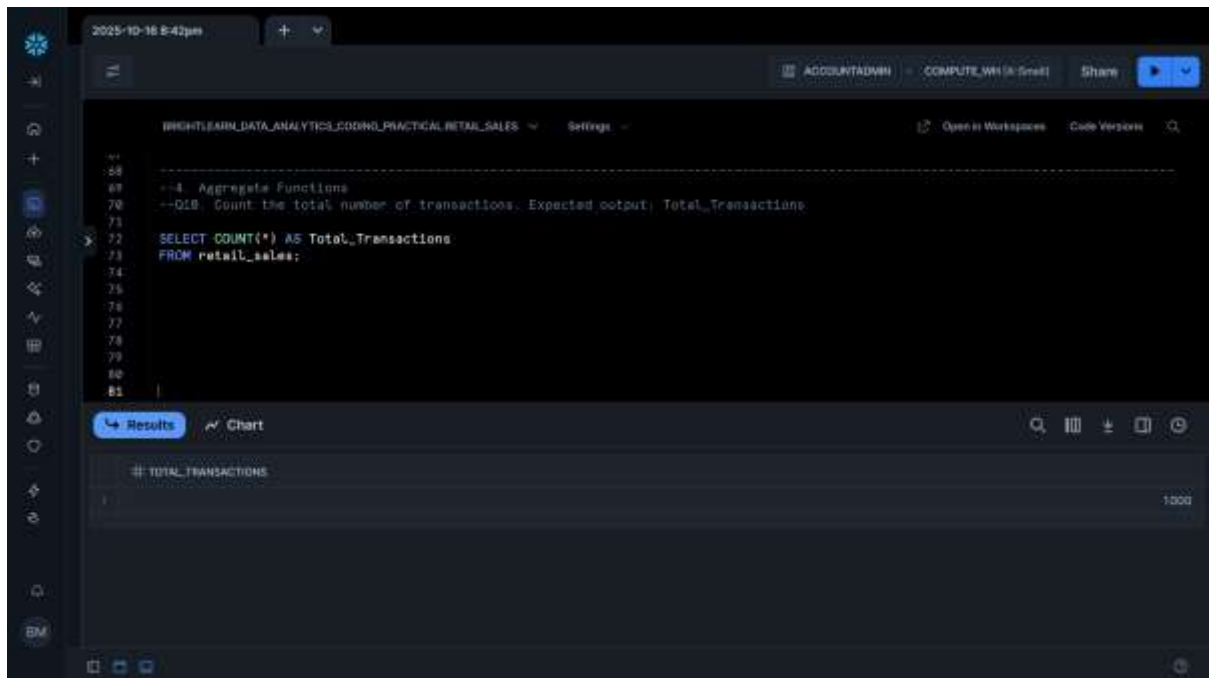


The screenshot shows a SQL IDE interface with a dark theme. The top bar displays the date and time '2025-10-16 8:42pm', a user 'ACCOUNTADMIN', and a database 'COMPUTE_WH (A-Snow)'. The main editor area contains a SQL query: `--Q9. Display all transactions where the Quantity is greater than or equal to 3. Expected output: All columns`
`SELECT *`
`FROM retail_sales`
`WHERE Quantity >= 3;`
Below the editor, the 'Results' tab is active, showing a table with 11 rows and 9 columns. The columns are: TRANSACTION_ID, DATE, CUSTOMER_ID, GENDER, AGE, PRODUCT_CATEGORY, QUANTITY, PRICE_PER_UNIT, and TOTAL_AMOUNT. The data rows are as follows:

TRANSACTION_ID	DATE	CUSTOMER_ID	GENDER	AGE	PRODUCT_CATEGORY	QUANTITY	PRICE_PER_UNIT	TOTAL_AMOUNT
1	2023-11-24	CUST001	Male	34	Beauty	3	50	150
8	2023-02-22	CUST008	Male	30	Electronics	4	25	100
10	2023-10-07	CUST010	Female	61	Clothing	4	50	200
12	2023-10-30	CUST012	Male	35	Beauty	3	25	75
13	2023-08-09	CUST013	Male	23	Electronics	3	500	1500
14	2023-01-17	CUST014	Male	64	Clothing	4	30	120
15	2023-01-16	CUST015	Female	42	Electronics	4	500	2000
16	2023-02-17	CUST016	Male	19	Clothing	3	500	1500
17	2023-04-23	CUST017	Female	27	Clothing	4	25	100
20	2023-11-05	CUST020	Male	22	Clothing	3	300	900
23	2023-04-12	CUST023	Female	35	Clothing	4	30	120

4. AGGREGATE FUNCTIONS

Q10. Count the total number of transactions. *Expected output: Total_Transactions*

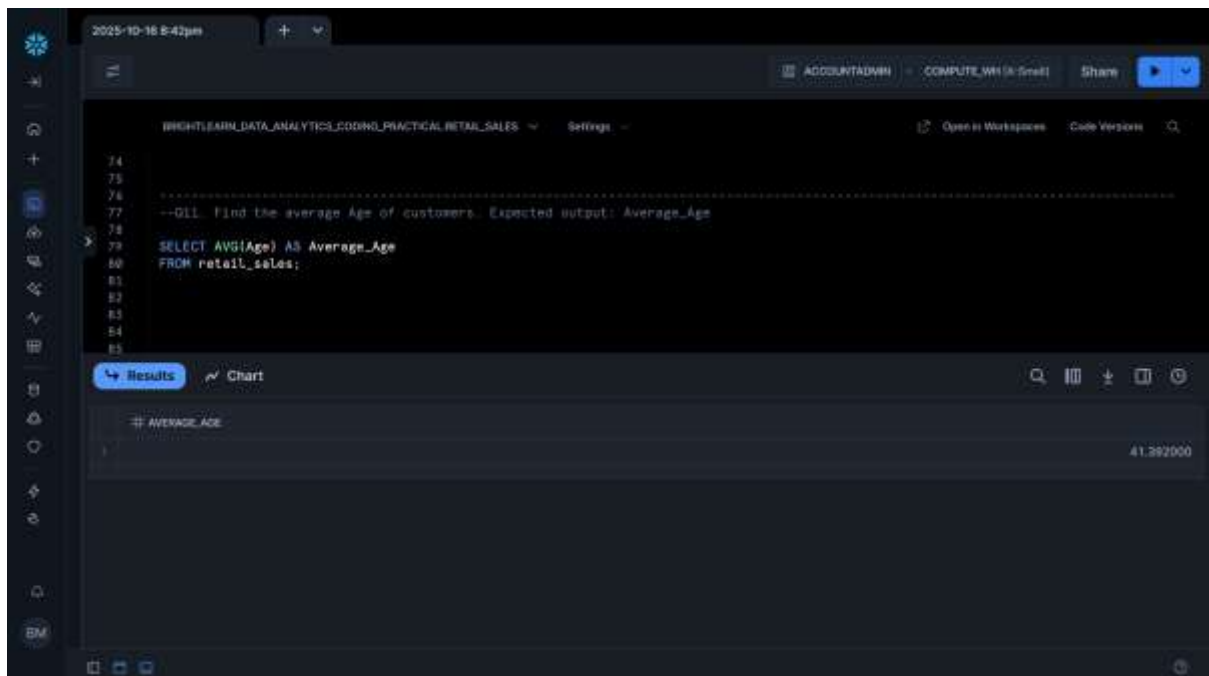


The screenshot shows a SQL IDE interface with a dark theme. The top bar displays the date and time '2025-10-16 5:42pm' and the user 'ACCOUNTADMIN'. The main editor area contains the following SQL code:

```
--4. Aggregate Functions
--Q10. Count the total number of transactions. Expected output: Total_Transactions
SELECT COUNT(*) AS Total_Transactions
FROM retail_sales;
```

The 'Results' tab is active, showing a single row with the column 'TOTAL_TRANSACTIONS' and the value '1000'.

Q11. Find the average Age of customers. *Expected output: Average_Age*

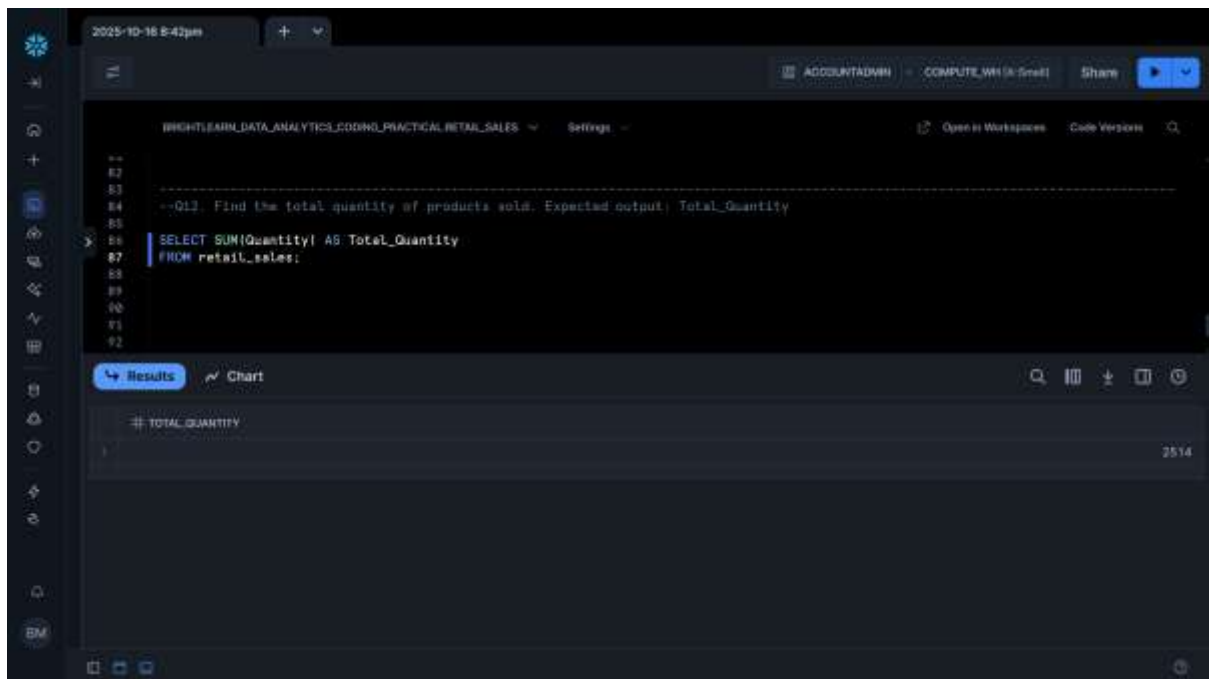


The screenshot shows the same SQL IDE interface. The main editor area contains the following SQL code:

```
--Q11. Find the average Age of customers. Expected output: Average_Age
SELECT AVG(Age) AS Average_Age
FROM retail_sales;
```

The 'Results' tab is active, showing a single row with the column 'AVERAGE_AGE' and the value '41.392000'.

Q12. Find the total quantity of products sold. *Expected output:* Total_Quantity

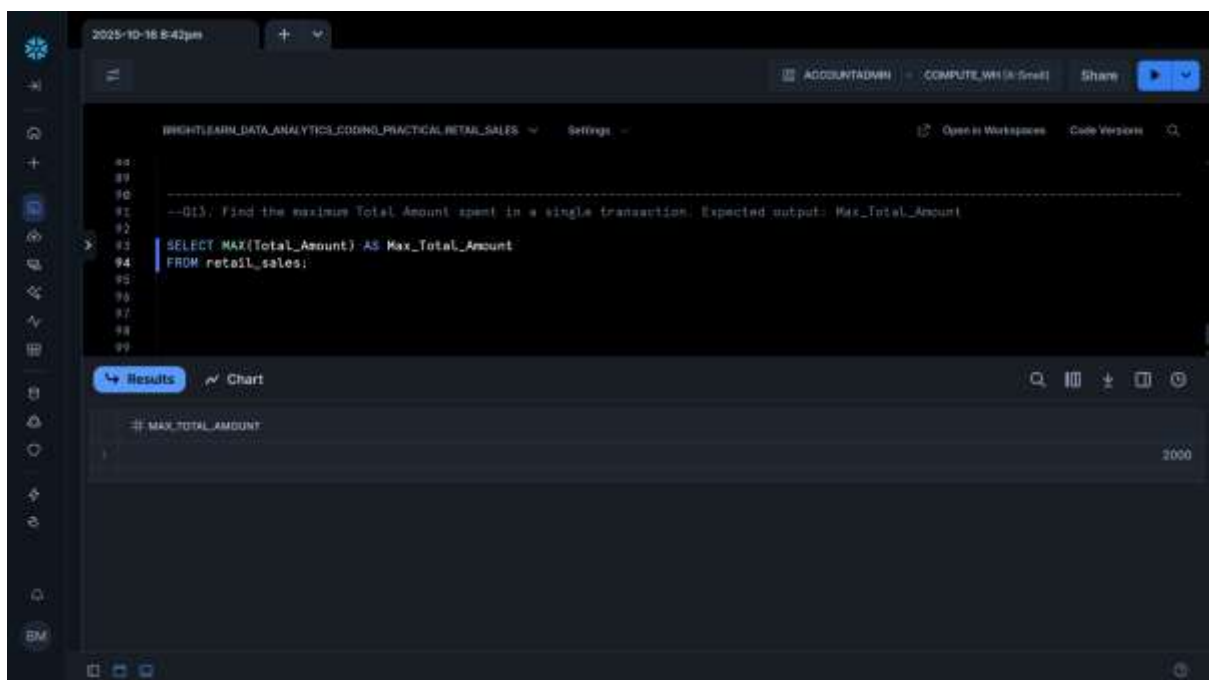


The screenshot shows a SQL IDE interface with a dark theme. The top bar displays the date and time '2025-10-16 8:42pm' and the user 'ACCOUNTADMIN'. The main editor area contains a SQL query: `SELECT SUM(Quantity) AS Total_Quantity FROM retail_sales;`. The query is highlighted in blue. Below the editor, the 'Results' tab is active, showing a single row with the column 'TOTAL_QUANTITY' and the value '2514'.

```
--Q12. Find the total quantity of products sold. Expected output: Total_Quantity
SELECT SUM(Quantity) AS Total_Quantity
FROM retail_sales;
```

TOTAL_QUANTITY
2514

Q13. Find the maximum Total Amount spent in a single transaction. *Expected output:* Max_Total_Amount



The screenshot shows a SQL IDE interface with a dark theme. The top bar displays the date and time '2025-10-16 8:42pm' and the user 'ACCOUNTADMIN'. The main editor area contains a SQL query: `SELECT MAX(Total_Amount) AS Max_Total_Amount FROM retail_sales;`. The query is highlighted in blue. Below the editor, the 'Results' tab is active, showing a single row with the column 'MAX_TOTAL_AMOUNT' and the value '2000'.

```
--Q13. Find the maximum Total Amount spent in a single transaction. Expected output: Max_Total_Amount
SELECT MAX(Total_Amount) AS Max_Total_Amount
FROM retail_sales;
```

MAX_TOTAL_AMOUNT
2000

Q14. Find the minimum Price per Unit in the dataset. Expected output: Min_Price_per_Unit

The screenshot shows a SQL IDE interface with a dark theme. The top bar displays the date and time '2025-10-16 8:42pm' and the user 'ACCOUNTADMIN'. The main editor area contains a SQL query with line numbers 96 to 106. The query is as follows:

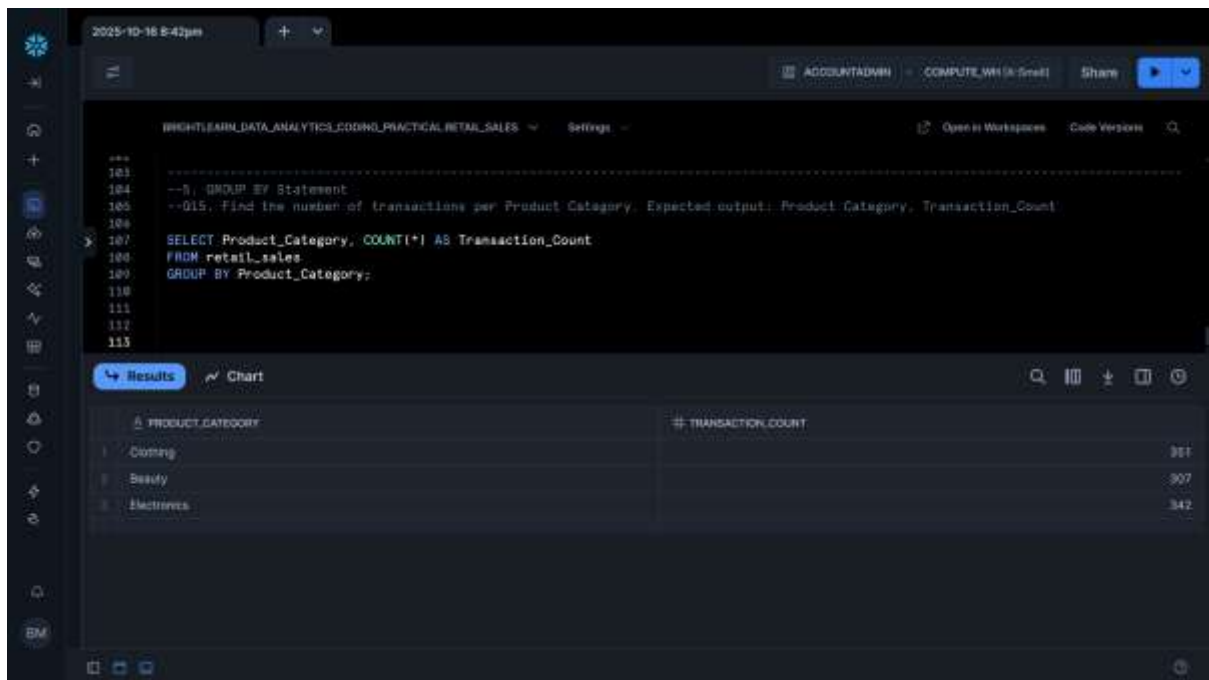
```
96  
97  
98  
99 --Q14. Find the minimum Price per Unit in the dataset. Expected output: Min_Price_per_Unit  
100  
101 SELECT MIN(Price_per_Unit) AS Min_Price_per_Unit  
102 FROM retail_sales;  
103  
104  
105  
106
```

Below the editor, the 'Results' tab is active, showing a table with one column, 'MIN_PRICE_PER_UNIT', and one row with the value '26'.

MIN_PRICE_PER_UNIT
26

5. GROUP BY STATEMENT

Q15. Find the number of transactions per Product Category. *Expected output:* Product Category, Transaction_Count



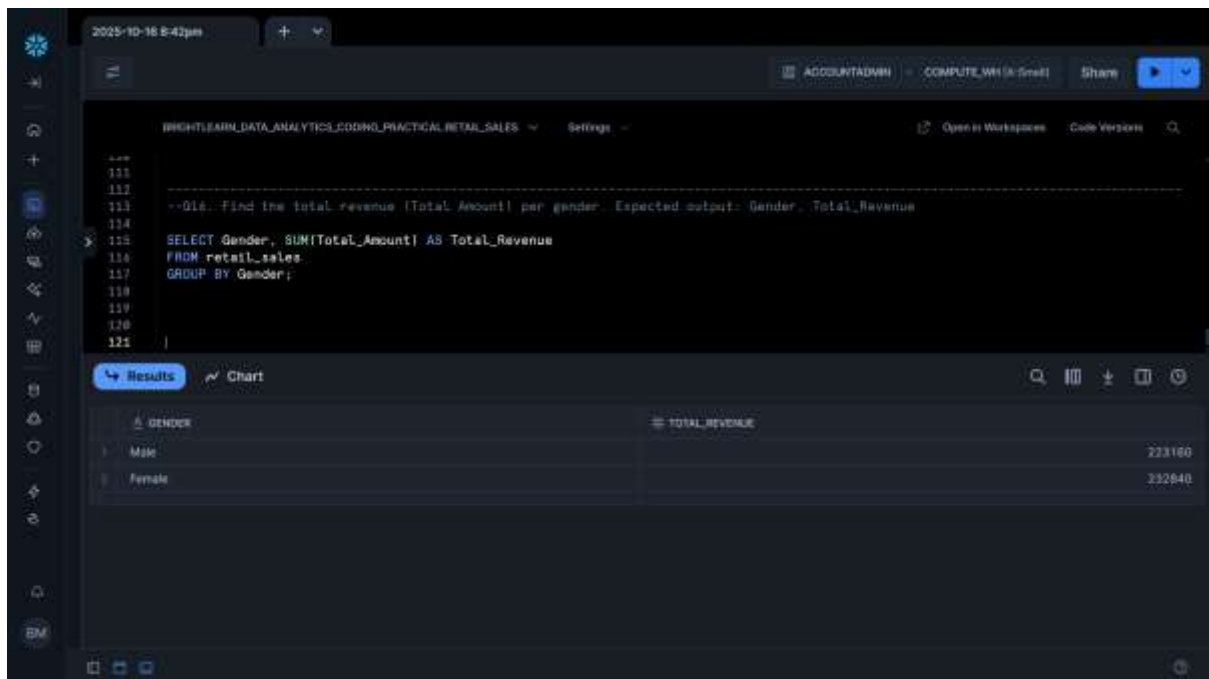
The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
---
101
102 -----
103 --5. GROUP BY Statement
104 --Q15. Find the number of transactions per Product Category. Expected output: Product Category, Transaction_Count
105
106
107 SELECT Product_Category, COUNT(*) AS Transaction_Count
108 FROM retail_sales
109 GROUP BY Product_Category;
110
111
112
113
```

The results pane displays the output of the query:

PRODUCT_CATEGORY	TRANSACTION_COUNT
Clothing	351
Beauty	307
Electronics	342

Q16. Find the total revenue (Total Amount) per gender. *Expected output:* Gender, Total_Revenue



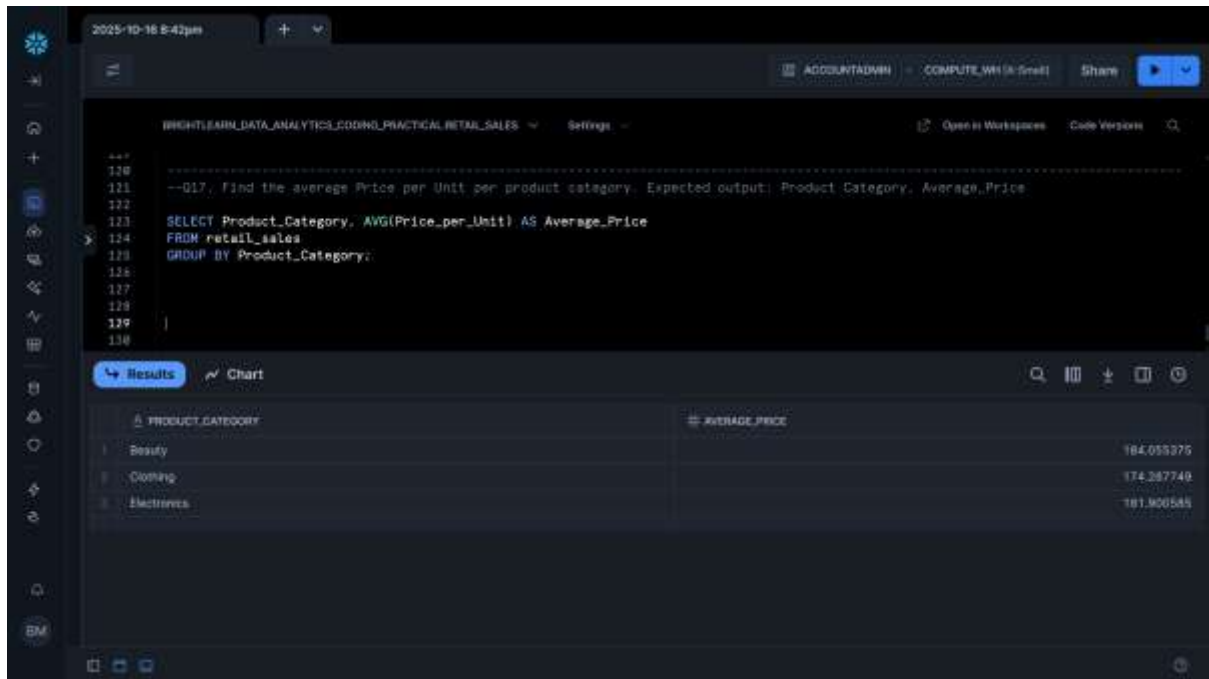
The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
---
111
112 -----
113 --Q16. Find the total revenue (Total Amount) per gender. Expected output: Gender, Total_Revenue
114
115
116 SELECT Gender, SUM(Total_Amount) AS Total_Revenue
117 FROM retail_sales
118 GROUP BY Gender;
119
120
121
```

The results pane displays the output of the query:

GENDER	TOTAL_REVENUE
Male	223160
Female	232840

Q17. Find the average Price per Unit per product category. *Expected output:* Product Category, Average_Price



The screenshot shows a SQL IDE interface with a dark theme. The top bar displays the date and time '2025-10-16 8:42pm', a '+' icon, and the user 'ACCOUNTADMIN' with a 'Share' button. The main editor area contains a SQL query with line numbers 120 through 130. The query is as follows:

```
120 -----
121 --Q17. Find the average Price per Unit per product category. Expected output: Product Category, Average_Price
122
123 SELECT Product_Category, AVG(Price_per_Unit) AS Average_Price
124 FROM retail_sales
125 GROUP BY Product_Category;
126
127
128
129
130
```

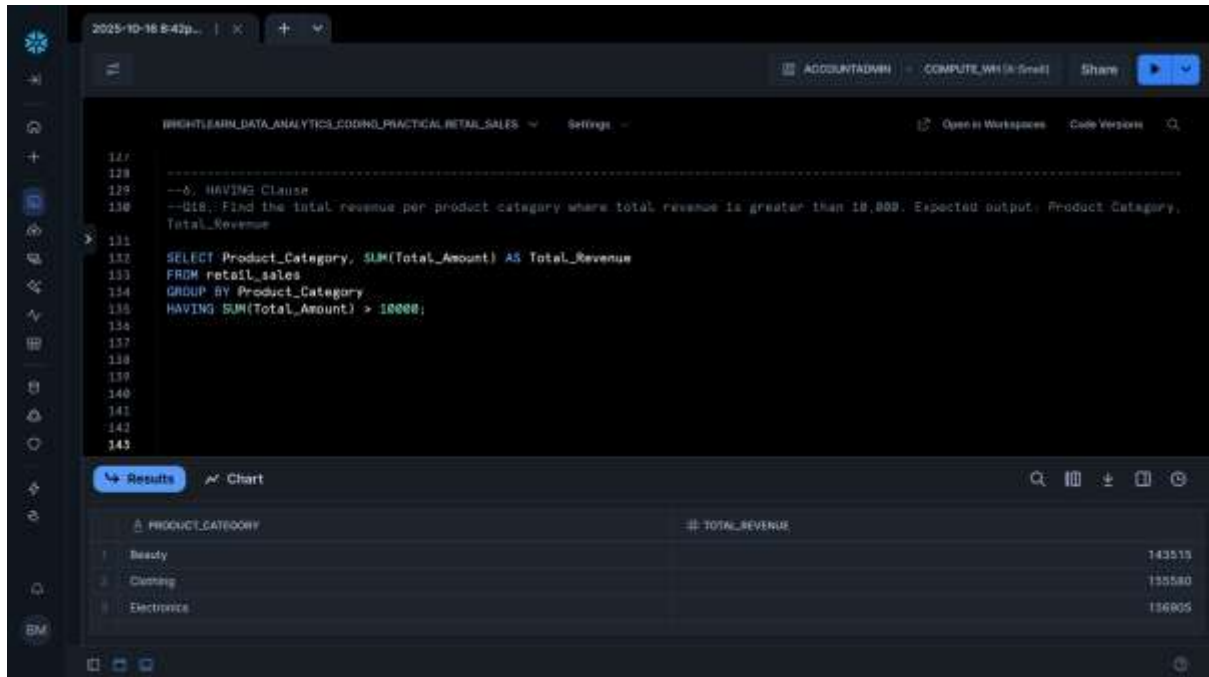
Below the editor, the 'Results' tab is active, showing a table with two columns: 'PRODUCT_CATEGORY' and 'AVERAGE_PRICE'. The table contains three rows of data:

PRODUCT_CATEGORY	AVERAGE_PRICE
Beauty	184.055375
Clothing	174.387749
Electronics	181.900585

6. HAVING CLAUSE

Q18. Find the total revenue per product category where total revenue is greater than 10,000.

Expected output: Product_Category, Total_Revenue



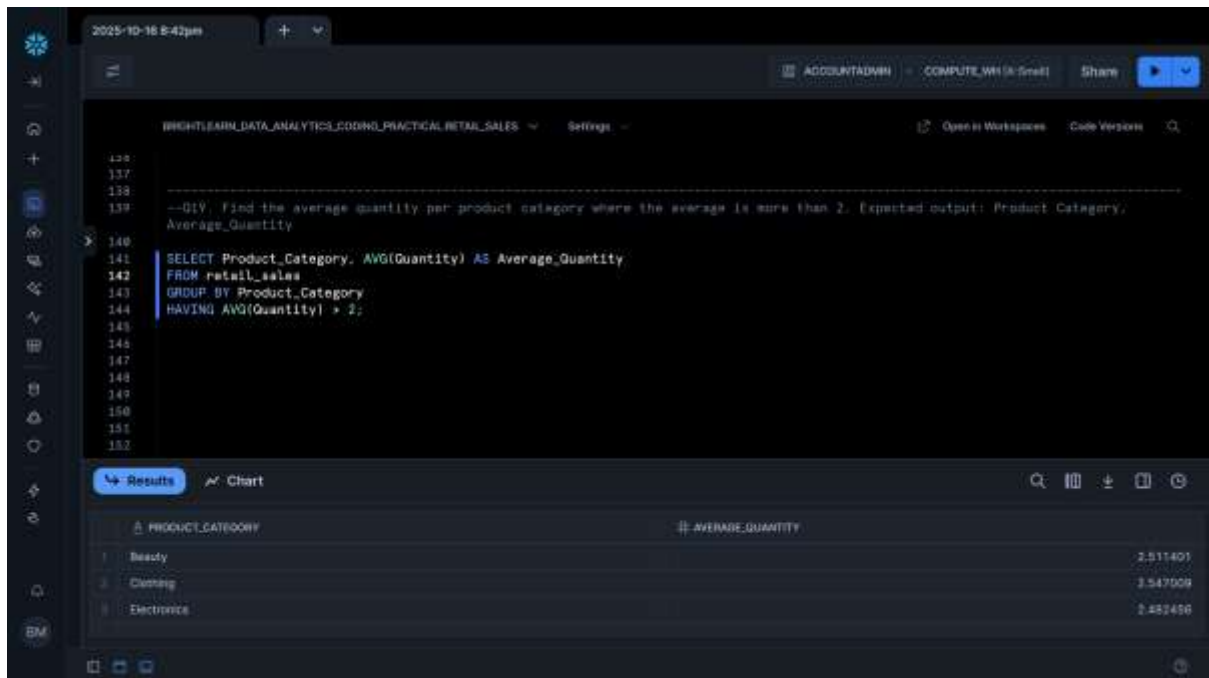
The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
117
118
119 --Q18. Find the total revenue per product category where total revenue is greater than 10,000. Expected output: Product_Category,
120 Total_Revenue
121
122 SELECT Product_Category, SUM(Total_Amount) AS Total_Revenue
123 FROM retail_sales
124 GROUP BY Product_Category
125 HAVING SUM(Total_Amount) > 10000;
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
```

The results pane shows the following data:

PRODUCT_CATEGORY	TOTAL_REVENUE
Beauty	143515
Clothing	135580
Electronics	136905

Q19. Find the average quantity per product category where the average is more than 2. *Expected output: Product_Category, Average_Quantity*



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

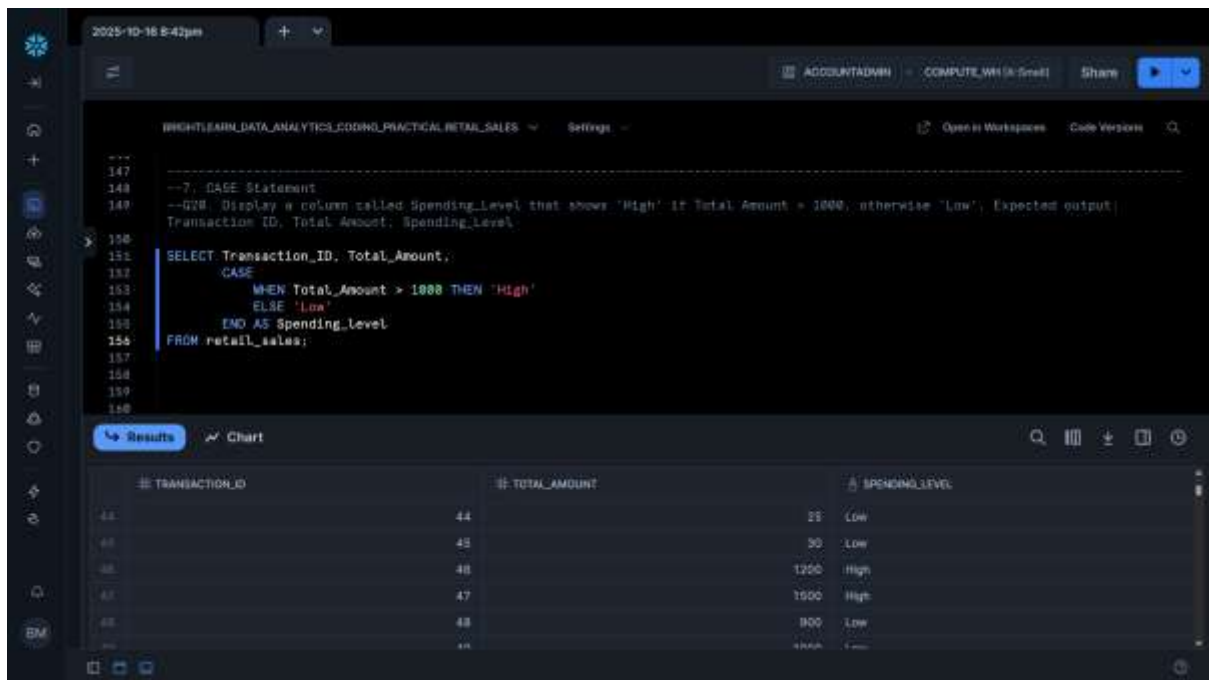
```
136
137
138
139 --Q19. Find the average quantity per product category where the average is more than 2. Expected output: Product_Category,
140 Average_Quantity
141
142 SELECT Product_Category, AVG(Quantity) AS Average_Quantity
143 FROM retail_sales
144 GROUP BY Product_Category
145 HAVING AVG(Quantity) > 2;
146
147
148
149
150
151
152
```

The results pane shows the following data:

PRODUCT_CATEGORY	AVERAGE_QUANTITY
Beauty	2.511401
Clothing	2.547009
Electronics	2.452456

7. CASE STATEMENT

Q20. Display a column called Spending_Level that shows 'High' if Total Amount > 1000, otherwise 'Low'. Expected output: Transaction ID, Total Amount, Spending_Level



The screenshot shows a SQL IDE interface with a dark theme. The top bar indicates the date and time as 2025-10-16 8:42pm. The main editor displays a SQL query using a CASE statement to categorize transactions based on their total amount. The query is as follows:

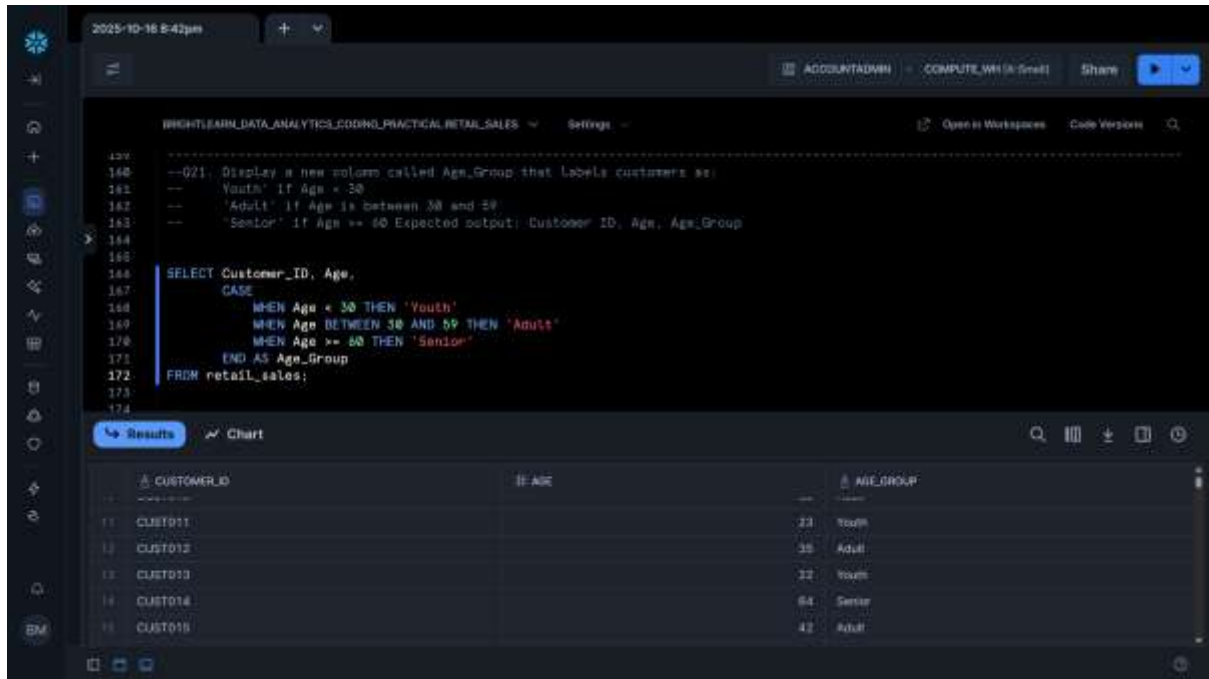
```
---  
147  
148 --7. CASE Statement  
149 --Q20. Display a column called Spending_Level that shows 'High' if Total Amount > 1000, otherwise 'Low'. Expected output:  
150 Transaction ID, Total Amount, Spending_Level  
151  
152 SELECT Transaction_ID, Total_Amount,  
153        CASE  
154            WHEN Total_Amount > 1000 THEN 'High'  
155            ELSE 'Low'  
156        END AS Spending_Level  
157 FROM retail_sales;  
158  
159  
160
```

Below the editor, the 'Results' tab is active, showing a table with three columns: TRANSACTION_ID, TOTAL_AMOUNT, and SPENDING_LEVEL. The table contains six rows of data:

TRANSACTION_ID	TOTAL_AMOUNT	SPENDING_LEVEL
44	25	Low
45	50	Low
46	1200	High
47	1500	High
48	900	Low
49	1000	Low

Q21. Display a new column called Age_Group that labels customers as:

- Youth' if Age < 30
- 'Adult' if Age is between 30 and 59
- 'Senior' if Age >= 60 Expected output: Customer ID, Age, Age_Group



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains a SQL statement that uses a CASE statement to categorize customers into 'Youth', 'Adult', or 'Senior' based on their age. The results pane displays the output of the query, showing columns for Customer ID, Age, and Age_Group.

```
--Q21. Display a new column called Age_Group that labels customers as:  
-- Youth' if Age < 30  
-- 'Adult' if Age is between 30 and 59  
-- 'Senior' if Age >= 60 Expected output: Customer ID, Age, Age_Group  
  
SELECT Customer_ID, Age,  
       CASE  
         WHEN Age < 30 THEN 'Youth'  
         WHEN Age BETWEEN 30 AND 59 THEN 'Adult'  
         WHEN Age >= 60 THEN 'Senior'  
       END AS Age_Group  
FROM retail_sales;
```

CUSTOMERID	AGE	AGE_GROUP
CUST011	23	Youth
CUST012	35	Adult
CUST013	22	Youth
CUST014	64	Senior
CUST015	42	Adult

