

# Customer Orders Data

## Cleaning Project

**Prepared By:** SOURABH ARYA

**Date:** 27 December 2025

**Tool Used:** MySQL

### Introduction

This project focuses on cleaning and standardizing a customer orders dataset. The goal was to ensure consistency across names, emails, mobile numbers, order details, ratings, and customer categories. A total of 15 steps were performed, and finally a cleaned view was created for analysis.

### ◆ STEP 0: Inspect Raw Data

```
SELECT *
```

```
FROM customer_orders
```

```
LIMIT 10;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

customer_id	first_name	last_name	email	mobile_number	order_id	order_date	delivery_date	order_amount
1001	Anita	SHARMA	anita1@GMAIL.COM	919110053353	ORD-2022-0001	2022-07-18	2022-07-20	3662.377
1002	KIRAN	Gupta	kiran2@GMAIL.COM	0091-9749	919110053353	2021-01-14	2021-01-21	10669.92
1003	Vikas	PATEL	vikas3@yahoo.com	0091-9664130526	ORD-2024-0003	2024-02-05	2024-02-08	23175.87
1004	POOJA	SINGH	pooja4@yahoo.com	919654049436	ORD-2022-0004	2022-07-01	2022-07-07	38255.81

customer orders 34 x

#	Time	Action	Message	Duration / Fetch
40	14:26:03	SELECT signup_date, DATEDIFF(NOW(), signup_date) AS days_with_compan...	350 row(s) returned	0.000 sec / 0.000 sec
41	14:28:01	SELECT order_amount, CASE WHEN order_amount >= 50000 THEN 'Hig...	350 row(s) returned	0.000 sec / 0.000 sec
42	14:30:10	SELECT signup_date, CASE WHEN DATEDIFF(NOW(), signup_date) <= ...	350 row(s) returned	0.000 sec / 0.000 sec
43	14:33:05	CREATE OR REPLACE VIEW customer_orders_cleaned AS SELECT customer_j...	0 row(s) affected	0.031 sec
44	14:36:02	SELECT * FROM customer_orders_cleaned LIMIT 10	10 row(s) returned	0.031 sec / 0.000 sec
45	14:38:43	SELECT * FROM customer_orders LIMIT 10	10 row(s) returned	0.000 sec / 0.000 sec

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

email	mobile_number	order_id	order_date	delivery_date	order_amount	city	signup_date	rating
a1@GMAIL.COM	919110053353	ORD-2022-0001	2022-07-18	2022-07-20	3662.377	PUNE	2021-11-22	1.882
n2@GMAIL.COM	0091-9749621470	ORD-2021-0002	2021-01-14	2021-01-21	10669.923	delhi	2018-10-02	4.892
is3@yahoo.com	0091-9664130526	ORD-2024-0003	2024-02-05	2024-02-08	23175.878	bangalore	2021-09-29	1.651
ja4@yahoo.com	919654049436	ORD-2022-0004	2022-07-01	2022-07-07	38255.816	MUMBAI	2020-07-22	4.287

customer orders 34 x

#	Time	Action	Message	Duration / Fetch
40	14:26:03	SELECT signup_date, DATEDIFF(NOW(), signup_date) AS days_with_compan...	350 row(s) returned	0.000 sec / 0.000 sec
41	14:28:01	SELECT order_amount, CASE WHEN order_amount >= 50000 THEN 'Hig...	350 row(s) returned	0.000 sec / 0.000 sec
42	14:30:10	SELECT signup_date, CASE WHEN DATEDIFF(NOW(), signup_date) <= ...	350 row(s) returned	0.000 sec / 0.000 sec
43	14:33:05	CREATE OR REPLACE VIEW customer_orders_cleaned AS SELECT customer_j...	0 row(s) affected	0.031 sec
44	14:36:02	SELECT * FROM customer_orders_cleaned LIMIT 10	10 row(s) returned	0.031 sec / 0.000 sec
45	14:38:43	SELECT * FROM customer_orders LIMIT 10	10 row(s) returned	0.000 sec / 0.000 sec

**Explanation:** This step was performed to quickly view the raw dataset before applying any cleaning. By selecting the first 10 rows, we can understand the structure of the table, the columns available (like first\_name, last\_name, email, mobile\_number, order\_date, etc.), and identify issues such as inconsistent formatting, extra spaces, mixed cases, or irregular values. This inspection helps plan the cleaning steps systematically.

## ◆ STEP 1: Clean `first_name` (Spaces + Case)

```
SELECT first_name, TRIM(first_name) AS trimmed,  
UPPER(TRIM(first_name)) AS cleaned_first_name  
FROM customer_orders;
```

The screenshot shows the SQL Developer interface with the following components:

- Navigator:** Shows the schema `customer_orders_csv` with tables `customer_orders`, `empl`, and `emplo`.
- SQL Editor:** Contains the query:  

```
SELECT first_name,  
       TRIM(first_name) AS trimmed,  
       UPPER(TRIM(first_name)) AS cleaned_first_name  
FROM customer_orders;
```
- Result Grid:** Displays the results of the query with columns `first_name`, `trimmed`, and `cleaned_first_name`. The data rows are:  

first_name	trimmed	cleaned_first_name
Anita	Anita	ANITA
KIRAN	KIRAN	KIRAN
Vikas	Vikas	VIKAS
POOJA	POOJA	POOJA
POOJA	POOJA	POOJA
- Output:** Shows the execution log with messages such as "350 row(s) returned" and "0 row(s) affected".

Explanation: **Converted first names to uppercase and removed extra spaces.**

## ◆ STEP 2: Clean last\_name

```
SELECT last_name,
```

```
UPPER(TRIM(last_name)) AS cleaned_last_name
```

```
FROM customer_orders;
```

The screenshot shows the SQL Developer interface with the following components:

- Navigator:** Shows the schema hierarchy for 'customer\_orders\_csv', including tables like 'customer\_orders'.
- SQL Editor:** Contains the query: 

```
SELECT last_name,
       UPPER(TRIM(last_name)) AS cleaned_last_name
FROM customer_orders;
```
- Result Grid:** Displays the query results with two columns: 'last\_name' and 'cleaned\_last\_name'. The data rows are: SHARMA, GUPTA, PATEL, SINGH, and VERMA.
- Output:** A log of database actions and messages, including the execution of the query and the resulting message: '350 row(s) returned'.

Automatic context help is disabled. Use the toolbar manually get help for the current caret position or toggle automatic help.

Explanation: **Converted last names to uppercase and removed extra spaces.**

## ◆ STEP 3: Create `full_name` (CONCAT)

```
SELECT CONCAT(  
    UPPER(TRIM(first_name)),  
    '  
    UPPER(TRIM(last_name))  
    ) AS full_name  
FROM customer_orders;
```

The screenshot shows the SQL Developer interface with the following components:

- Navigator:** Shows the schema `customer_orders_csv` with tables `customer_orders` and `customer_orders_csv`.
- SQL Editor:** Contains the query:  

```
SELECT CONCAT(UPPER(TRIM(first_name)), ' ', UPPER(TRIM(last_name))) AS full_name  
FROM customer_orders;
```
- Result Grid:** Displays the results of the query, showing a single column `full_name` with five rows of data:  

full_name
ANITA SHARMA
KIRAN GUPTA
VIKAS PATEL
POOJA SINGH
POOJA VERMA
- Output:** Shows the execution log with the following actions:  

#	Time	Action	Message	Duration / Fetch
24	13:27:43	SELECT signup_date, CASE WHEN DATEDIFF(NOW(), signup_date) <= ...	350 row(s) returned	0.000 sec / 0.000 sec
25	13:28:24	CREATE OR REPLACE VIEW customer_orders_cleaned AS SELECT customer_j...	0 row(s) affected	0.031 sec
26	13:29:20	SELECT * FROM customer_orders_cleaned LIMIT 10	10 row(s) returned	0.015 sec / 0.000 sec
27	13:44:45	SELECT first_name, TRIM(first_name) AS trimmed, UPPER(TRIM(first_nam...	350 row(s) returned	0.000 sec / 0.000 sec
28	13:51:06	SELECT last_name, UPPER(TRIM(last_name)) AS cleaned_last_name FROM c...	350 row(s) returned	0.000 sec / 0.000 sec
29	14:00:23	SELECT CONCAT(UPPER(TRIM(first_name)), ' ', UPPER(TRIM(last_name))) AS full...	350 row(s) returned	0.016 sec / 0.000 sec

Explanation: **Combined first and last names into a standardized full name.**

## ◆ STEP 4: Clean **email** (Standardization)

**SELECT email,**

**LOWER(email) AS cleaned\_email**

**FROM customer\_orders;**

The screenshot shows the SQL Developer interface with a query window containing the following SQL code:

```
25  
26 SELECT email,  
27        LOWER(email) AS cleaned_email  
28 FROM customer_orders;  
29  
30  
31
```

The Results pane displays the following data:

email	cleaned_email
anrita1@GMAIL.COM	anrita1@gmail.com
kiran2@GMAIL.COM	kiran2@gmail.com
vikas3@yahoo.com	vikas3@yahoo.com
pooja4@yahoo.com	pooja4@yahoo.com
pooja5@yahoo.com	pooja5@yahoo.com

The Output pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
27	13:44:45	SELECT first_name, TRIM(first_name) AS trimmed, UPPER(TRIM(first_name)) AS cleaned_first_name FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
28	13:51:06	SELECT last_name, UPPER(TRIM(last_name)) AS cleaned_last_name FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
29	14:00:23	SELECT CONCAT(UPPER(TRIM(first_name)), ' ', UPPER(TRIM(last_name))) AS full_name FROM customer_orders LIMIT...	350 row(s) returned	0.016 sec / 0.000 sec
30	14:07:26	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
31	14:07:38	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
32	14:13:34	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: **Converted all emails to lowercase.**

## ◆ STEP 5: Clean mobile\_number (Extract last 10 digits)

```
SELECT mobile_number,
```

```
    SUBSTR(mobile_number, LENGTH(mobile_number) - 9, 10)  
AS cleaned_mobile
```

```
FROM customer_orders;
```

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

```
31  
32 SELECT mobile_number,  
33        SUBSTR(mobile_number, LENGTH(mobile_number) - 9, 10) AS cleaned_mobile  
34 FROM customer_orders;  
35  
36  
37
```

The results grid displays the following data:

mobile_number	cleaned_mobile
919110053353	9110053353
0091-9749621470	9749621470
0091-9664130526	9664130526
919654049436	9654049436
919940992571	9940992571

The output pane shows the execution log with the following messages:

#	Time	Action	Message	Duration / Fetch
28	13:51:06	SELECT last_name, UPPER(TRIM(last_name)) AS cleaned_last_name FROM c...	350 row(s) returned	0.000 sec / 0.000 sec
29	14:00:23	SELECT CONCAT(UPPER(TRIM(first_name)), ' ', UPPER(TRIM(last_name))) AS full...	350 row(s) returned	0.016 sec / 0.000 sec
30	14:07:26	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
31	14:07:38	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
32	14:13:34	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
33	14:16:22	SELECT mobile_number, SUBSTR(mobile_number, LENGTH(mobile_number) - ...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: **Extracted last 10 digits of mobile numbers.**

## ◆ STEP 6: Extract Year from `order_id`

```
SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year
```

```
FROM customer_orders;
```

The screenshot shows a SQL IDE interface with the following components:

- Navigator:** Displays a tree view of schemas. The `customer_orders_csv` schema is expanded, showing tables like `customer_orders`.
- SQL Editor:** Contains the query:

```
SELECT order_id,  
       SUBSTR(order_id, 5, 4) AS order_year  
FROM customer_orders;
```
- Result Grid:** Displays the query results with columns `order_id` and `order_year`. The results are:

order_id	order_year
ORD-2022-0001	2022
ORD-2021-0002	2021
ORD-2024-0003	2024
ORD-2022-0004	2022
ORD-2022-0005	2022
- Output:** A log of database actions. The last entry (35) shows the execution of the query to extract the year from the order ID, returning 350 rows in 0.000 seconds.

Explanation: **Extracted year from order ID.**

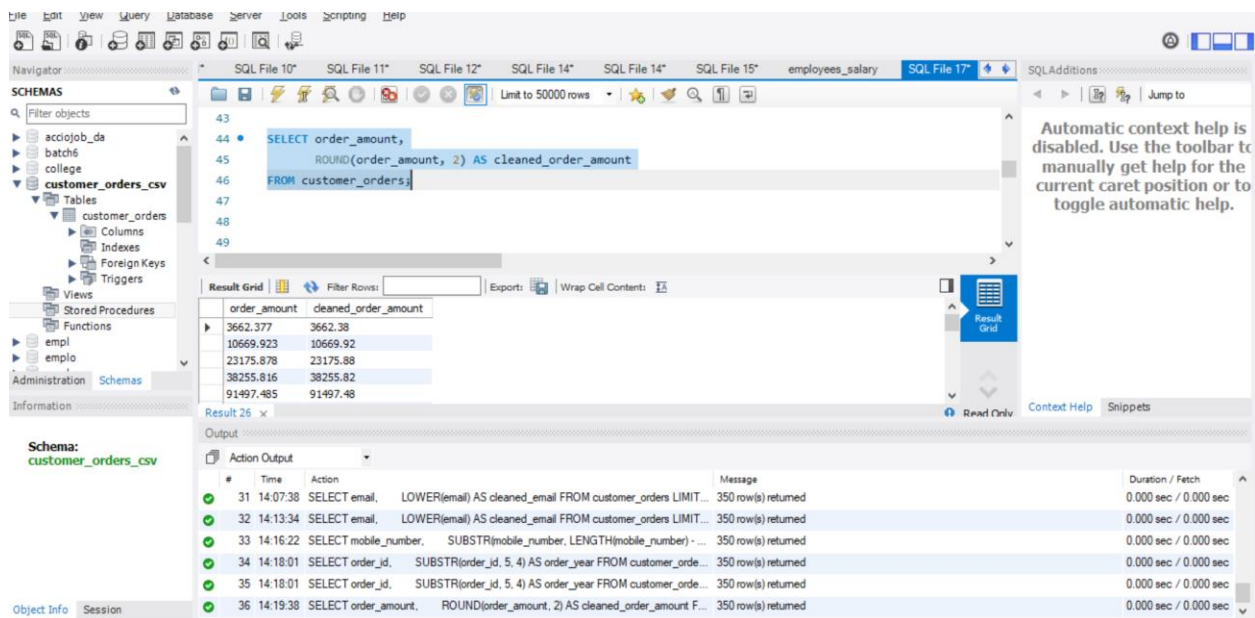


## ◆ STEP 7: Round `order_amount`

```
SELECT order_amount,
```

```
ROUND(order_amount, 2) AS cleaned_order_amount
```

```
FROM customer_orders;
```



The screenshot shows a SQL IDE interface with a query editor, a result grid, and an output pane. The query being executed is:

```
SELECT order_amount,  
       ROUND(order_amount, 2) AS cleaned_order_amount  
FROM customer_orders;
```

The result grid displays the following data:

order_amount	cleaned_order_amount
3662.377	3662.38
10669.923	10669.92
23175.878	23175.88
38255.816	38255.82
91497.485	91497.48

The output pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
31	14:07:38	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
32	14:13:34	SELECT email, LOWER(email) AS cleaned_email FROM customer_orders LIMIT...	350 row(s) returned	0.000 sec / 0.000 sec
33	14:16:22	SELECT mobile_number, SUBSTR(mobile_number, LENGTH(mobile_number) - ...	350 row(s) returned	0.000 sec / 0.000 sec
34	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_orde...	350 row(s) returned	0.000 sec / 0.000 sec
35	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_orde...	350 row(s) returned	0.000 sec / 0.000 sec
36	14:19:38	SELECT order_amount, ROUND(order_amount, 2) AS cleaned_order_amount F...	350 row(s) returned	0.000 sec / 0.000 sec

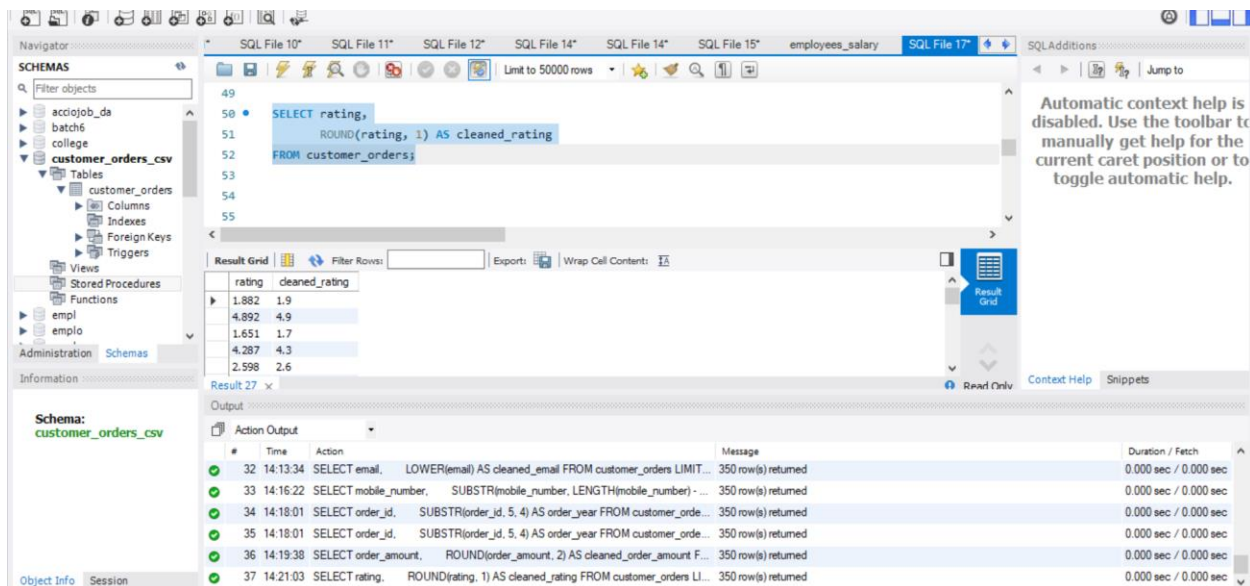
Explanation: **Rounded order amounts to 2 decimal places.**

## ◆ STEP 8: Round rating

SELECT rating,

ROUND(rating, 1) AS cleaned\_rating

FROM customer\_orders;



The screenshot shows a SQL IDE interface with a query editor, a schema navigator, and a results pane. The query editor contains the following SQL statement:

```
SELECT rating,  
       ROUND(rating, 1) AS cleaned_rating  
FROM customer_orders;
```

The results pane displays a table with two columns: **rating** and **cleaned\_rating**. The data is as follows:

rating	cleaned_rating
1.882	1.9
4.892	4.9
1.651	1.7
4.287	4.3
2.598	2.6

The bottom pane shows the execution log with the following entry:

#	Time	Action	Message	Duration / Fetch
37	14:21:03	SELECT rating, ROUND(rating, 1) AS cleaned_rating FROM customer_orders U...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: **Rounded ratings to 1 decimal place.**

## ◆ STEP 9: Standardize city

SELECT city,

UPPER(city) AS cleaned\_city

FROM customer\_orders;

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

```
55  
56 SELECT city,  
57     UPPER(city) AS cleaned_city  
58 FROM customer_orders;  
59  
60  
61
```

The results grid displays the following data:

city	cleaned_city
PUNE	PUNE
delhi	DELHI
bangalore	BANGALORE
MUMBAI	MUMBAI
hyderabad	HYDERABAD

The output pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
33	14:16:22	SELECT mobile_number, SUBSTR(mobile_number, LENGTH(mobile_number) - ...	350 row(s) returned	0.000 sec / 0.000 sec
34	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_order...	350 row(s) returned	0.000 sec / 0.000 sec
35	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_order...	350 row(s) returned	0.000 sec / 0.000 sec
36	14:19:38	SELECT order_amount, ROUND(order_amount, 2) AS cleaned_order_amount F...	350 row(s) returned	0.000 sec / 0.000 sec
37	14:21:03	SELECT rating, ROUND(rating, 1) AS cleaned_rating FROM customer_orders LI...	350 row(s) returned	0.000 sec / 0.000 sec
38	14:22:53	SELECT city, UPPER(city) AS cleaned_city FROM customer_orders LIMIT 0, 50...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: **Converted city names to uppercase.**

## ◆ STEP 10: Delivery Time Calculation (DATEDIFF)

```
SELECT order_date, delivery_date,
```

```
DATEDIFF(delivery_date, order_date) AS delivery_days
```

```
FROM customer_orders;
```

The screenshot shows a SQL IDE interface with a query editor, a result grid, and an output pane. The query editor contains the following SQL code:

```
61 SELECT order_date, delivery_date,  
62 DATEDIFF(delivery_date, order_date) AS delivery_days  
63 FROM customer_orders;  
64  
65  
66  
67
```

The result grid displays the following data:

order_date	delivery_date	delivery_days
2022-07-18	2022-07-20	2
2021-01-14	2021-01-21	7
2024-02-05	2024-02-08	3
2022-07-01	2022-07-07	6
2022-07-03	2022-07-04	1

The output pane shows the execution log with the following messages:

#	Time	Action	Message	Duration / Fetch
34	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_order...	350 row(s) returned	0.000 sec / 0.000 sec
35	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_order...	350 row(s) returned	0.000 sec / 0.000 sec
36	14:19:38	SELECT order_amount, ROUND(order_amount, 2) AS cleaned_order_amount F...	350 row(s) returned	0.000 sec / 0.000 sec
37	14:21:03	SELECT rating, ROUND(rating, 1) AS cleaned_rating FROM customer_orders LI...	350 row(s) returned	0.000 sec / 0.000 sec
38	14:22:53	SELECT city, UPPER(city) AS cleaned_city FROM customer_orders LIMIT 0, 50...	350 row(s) returned	0.000 sec / 0.000 sec
39	14:24:31	SELECT order_date, delivery_date, DATEDIFF(delivery_date, order_date) AS d...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: **Calculated delivery days between order and delivery dates.**

## ◆ STEP 11: Customer Tenure Calculation

```
SELECT signup_date,
```

```
DATEDIFF(NOW(), signup_date) AS days_with_company
```

```
FROM customer_orders;
```

The screenshot shows a SQL IDE interface with a query editor, a results grid, and an output pane. The query being executed is:

```
SELECT signup_date,  
       DATEDIFF(NOW(), signup_date) AS days_with_company  
FROM customer_orders;
```

The results grid displays the following data:

signup_date	days_with_company
2021-11-22	1496
2018-10-02	2643
2021-09-29	1550
2020-07-22	1984
2021-11-19	1499

The output pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
35	14:18:01	SELECT order_id, SUBSTR(order_id, 5, 4) AS order_year FROM customer_order...	350 row(s) returned	0.000 sec / 0.000 sec
36	14:19:38	SELECT order_amount, ROUND(order_amount, 2) AS cleaned_order_amount F...	350 row(s) returned	0.000 sec / 0.000 sec
37	14:21:03	SELECT rating, ROUND(rating, 1) AS cleaned_rating FROM customer_orders LI...	350 row(s) returned	0.000 sec / 0.000 sec
38	14:22:53	SELECT city, UPPER(city) AS cleaned_city FROM customer_orders LIMIT 0, 50...	350 row(s) returned	0.000 sec / 0.000 sec
39	14:24:31	SELECT order_date, delivery_date, DATEDIFF(delivery_date, order_date) AS d...	350 row(s) returned	0.000 sec / 0.000 sec
40	14:26:03	SELECT signup_date, DATEDIFF(NOW(), signup_date) AS days_with_compan...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: **Calculated how many days each customer has been with the company.**

## ◆ STEP 12: CASE WHEN – Order Value Categor

```
SELECT order_amount,
```

```
CASE
```

```
WHEN order_amount >= 50000 THEN 'High Value'
```

```
WHEN order_amount >= 20000 THEN 'Medium Value'
```

```
ELSE 'Low Value'
```

```
END AS order_category
```

```
FROM customer_orders;
```

The screenshot shows the SQL Developer interface with a query window containing the following SQL code:

```
77      WHEN order_amount >= 20000 THEN 'Medium Value'
78      ELSE 'Low Value'
79      END AS order_category
80  FROM customer_orders;
81
82
83
```

The Results window displays the following data:

order_amount	order_category
3662.377	Low Value
10669.923	Low Value
23175.878	Medium Value
38255.816	Medium Value
91497.485	High Value

The Action Output window shows the execution of the query and other SQL statements, including the final query that categorizes orders by value.

Explanation: **Categorized orders into High, Medium, or Low value.**

## ◆ STEP 13: CASE WHEN – Customer Type

```
SELECT signup_date,
```

```
CASE
```

```
WHEN DATEDIFF(NOW(), signup_date) <= 30 THEN 'New'
```

```
WHEN DATEDIFF(NOW(), signup_date) <= 180 THEN 'Regular'
```

```
ELSE 'Loyal'
```

```
END AS customer_type
```

```
FROM customer_orders;
```

The screenshot shows a SQL IDE interface with a query editor, a result grid, and an output pane. The query being executed is:

```
SELECT signup_date,
CASE
WHEN DATEDIFF(NOW(), signup_date) <= 30 THEN 'New'
WHEN DATEDIFF(NOW(), signup_date) <= 180 THEN 'Regular'
ELSE 'Loyal'
END AS customer_type
FROM customer_orders;
```

The result grid displays the following data:

signup_date	customer_type
2021-11-22	Loyal
2018-10-02	Loyal
2021-09-29	Loyal
2020-07-22	Loyal
2021-11-19	Loyal

The output pane shows the execution log with the following messages:

#	Time	Action	Message	Duration / Fetch
37	14:21:03	SELECT rating, ROUND(rating, 1) AS cleaned_rating FROM customer_orders LI...	350 row(s) returned	0.000 sec / 0.000 sec
38	14:22:53	SELECT city, UPPER(city) AS cleaned_city FROM customer_orders LIMIT 0, 50...	350 row(s) returned	0.000 sec / 0.000 sec
39	14:24:31	SELECT order_date, delivery_date, DATEDIFF(delivery_date, order_date) AS d...	350 row(s) returned	0.000 sec / 0.000 sec
40	14:26:03	SELECT signup_date, DATEDIFF(NOW(), signup_date) AS days_with_compan...	350 row(s) returned	0.000 sec / 0.000 sec
41	14:28:01	SELECT order_amount, CASE WHEN order_amount >= 50000 THEN 'High...	350 row(s) returned	0.000 sec / 0.000 sec
42	14:30:10	SELECT signup_date, CASE WHEN DATEDIFF(NOW(), signup_date) <= ...	350 row(s) returned	0.000 sec / 0.000 sec

Explanation: Classified customers as New, Regular, or Loyal.

## ◆ STEP 14: FINAL CLEANED VIEW (Industry Practice)

```
CREATE OR REPLACE VIEW customer_orders_cleaned AS
```

```
SELECT
```

```
customer_id,
```

```
UPPER(TRIM(first_name)) AS first_name,
```

```
UPPER(TRIM(last_name)) AS last_name,
```

```
CONCAT(UPPER(TRIM(first_name)), ' ', UPPER(TRIM(last_name))) AS full_name,
```

```
LOWER(email) AS email,
```

```
SUBSTR(mobile_number, LENGTH(mobile_number) - 9, 10) AS mobile_number,
```

```
order_id,
```

```
SUBSTR(order_id, 5, 4) AS order_year,
```

```
order_date,
```

```
delivery_date,
```

```
DATEDIFF(delivery_date, order_date) AS delivery_days,
```

```
ROUND(order_amount, 2) AS order_amount,
```

```
UPPER(city) AS city,
```

```
signup_date,
```

```
DATEDIFF(NOW(), signup_date) AS customer_tenure_days,
```



CASE

WHEN order\_amount >= 50000 THEN 'High Value'

WHEN order\_amount >= 20000 THEN 'Medium Value'

ELSE 'Low Value'

END AS order\_category,

ROUND(rating, 1) AS rating,

CASE

WHEN DATEDIFF(NOW(), signup\_date) <= 30 THEN 'New'

WHEN DATEDIFF(NOW(), signup\_date) <= 180 THEN 'Regular'

ELSE 'Loyal'

END AS customer\_type

FROM customer\_orders;

The screenshot displays the SQL Developer interface. The left sidebar shows the 'SCHEMAS' tree with 'customer\_orders\_csv' selected. The main editor window shows the following SQL code:

```
CREATE OR REPLACE VIEW customer_orders_cleaned AS
SELECT
  customer_id,
  UPPER(TRIM(first_name)) AS first_name,
  UPPER(TRIM(last_name)) AS last_name,
  CONCAT(UPPER(TRIM(first_name)), ' ', UPPER(TRIM(last_name))) AS full_name,
  LOWER(email) AS email,
  SUBSTR(mobile_number, LENGTH(mobile_number) - 9, 10) AS mobile_number,
  order_id,
  SUBSTR(order_id, 5, 4) AS order_year,
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
38	14:22:53	SELECT city, UPPER(city) AS cleaned_city FROM customer_orders LIMIT 0.50...	350 row(s) returned	0.000 sec / 0.000 sec
39	14:24:31	SELECT order_date, delivery_date, DATEDIFF(delivery_date, order_date) AS d...	350 row(s) returned	0.000 sec / 0.000 sec
40	14:26:03	SELECT signup_date, DATEDIFF(NOW(), signup_date) AS days_with_compan...	350 row(s) returned	0.000 sec / 0.000 sec
41	14:28:01	SELECT order_amount, CASE WHEN order_amount >= 50000 THEN 'Hig...	350 row(s) returned	0.000 sec / 0.000 sec
42	14:30:10	SELECT signup_date, CASE WHEN DATEDIFF(NOW(), signup_date) <= ...	350 row(s) returned	0.000 sec / 0.000 sec
43	14:33:05	CREATE OR REPLACE VIEW customer_orders_cleaned AS SELECT customer_j...	0 row(s) affected	0.031 sec

Explanation: **Created a view combining all transformations.**

## ◆ STEP 15: Validate Cleaned Data

**SELECT \* FROM customer\_orders\_cleaned LIMIT 10;**

The screenshot shows the SQL Developer interface with the query `SELECT * FROM customer_orders_cleaned LIMIT 10;` executed. The result grid displays the following data:

customer_id	first_name	last_name	full_name	email	mobile_number	order_id	order_year	order_date
1001	ANITA	SHARMA	ANITA SHARMA	anita1@gmail.com	9110053353	ORD-2022-0001	2022	2022-07-18
1002	KIRAN	GUPTA	KIRAN GUPTA	kirana2@gmail.com	9749621470	ORD-2021-0002	2021	2021-01-14
1003	VIKAS	PATEL	VIKAS PATEL	vikas3@yahoo.com	9664130526	ORD-2024-0003	2024	2024-02-05
1004	POOJA	SINGH	POOJA SINGH	pooja4@yahoo.com	9654049436	ORD-2022-0004	2022	2022-07-01

The Action Output pane shows the execution details for the query, including the time taken and the number of rows returned.

The screenshot shows the SQL Developer interface with the query `SELECT * FROM customer_orders_cleaned LIMIT 10;` executed. The result grid displays the following data:

order_date	delivery_date	delivery_days	order_amount	city	signup_date	customer_tenure_days	order_category	ratio
2022-07-18	2022-07-20	2	3662.38	PUNE	2021-11-22	1496	Low Value	1.9
2021-01-14	2021-01-21	7	10669.92	DELHI	2018-10-02	2643	Low Value	4.9
2024-02-05	2024-02-08	3	23175.88	BANGALORE	2021-09-29	1550	Medium Value	1.7
2022-07-01	2022-07-07	6	38255.82	MUMBAI	2020-07-22	1984	Medium Value	4.3

The Action Output pane shows the execution details for the query, including the time taken and the number of rows returned.

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Schema: customer\_orders\_csv

customer orders cleaned 33

id	delivery_date	delivery_days	order_amount	city	signup_date	customer_tenure_days	order_category	rating	customer_type
-07-20	2	3662.38	PUNE	2021-11-22	1496	Low Value	1.9	Loyal	
-01-21	7	10669.92	DELHI	2018-10-02	2643	Low Value	4.9	Loyal	
-02-08	3	23175.88	BANGALORE	2021-09-29	1550	Medium Value	1.7	Loyal	
-07-07	6	38255.82	MUMBAI	2020-07-22	1984	Medium Value	4.3	Loyal	

Output

#	Time	Action	Message	Duration / Fetch
39	14:24:31	SELECT order_date, delivery_date, DATEDIFF(delivery_date, order_date) AS d...	350 row(s) returned	0.000 sec / 0.000 sec
40	14:26:03	SELECT signup_date, DATEDIFF(NOW(), signup_date) AS days_with_compan...	350 row(s) returned	0.000 sec / 0.000 sec
41	14:28:01	SELECT order_amount, CASE WHEN order_amount >= 50000 THEN 'Hig...	350 row(s) returned	0.000 sec / 0.000 sec
42	14:30:10	SELECT signup_date, CASE WHEN DATEDIFF(NOW(), signup_date) <= ...	350 row(s) returned	0.000 sec / 0.000 sec
43	14:33:05	CREATE OR REPLACE VIEW customer_orders_cleaned AS SELECT customer_j...	0 row(s) affected	0.031 sec
44	14:36:02	SELECT * FROM customer_orders_cleaned LIMIT 10	10 row(s) returned	0.031 sec / 0.000 sec

Explanation: **Verified that all cleaned columns are correctly displayed.**