Assignment Day - 9

Ratan Singh - 280013

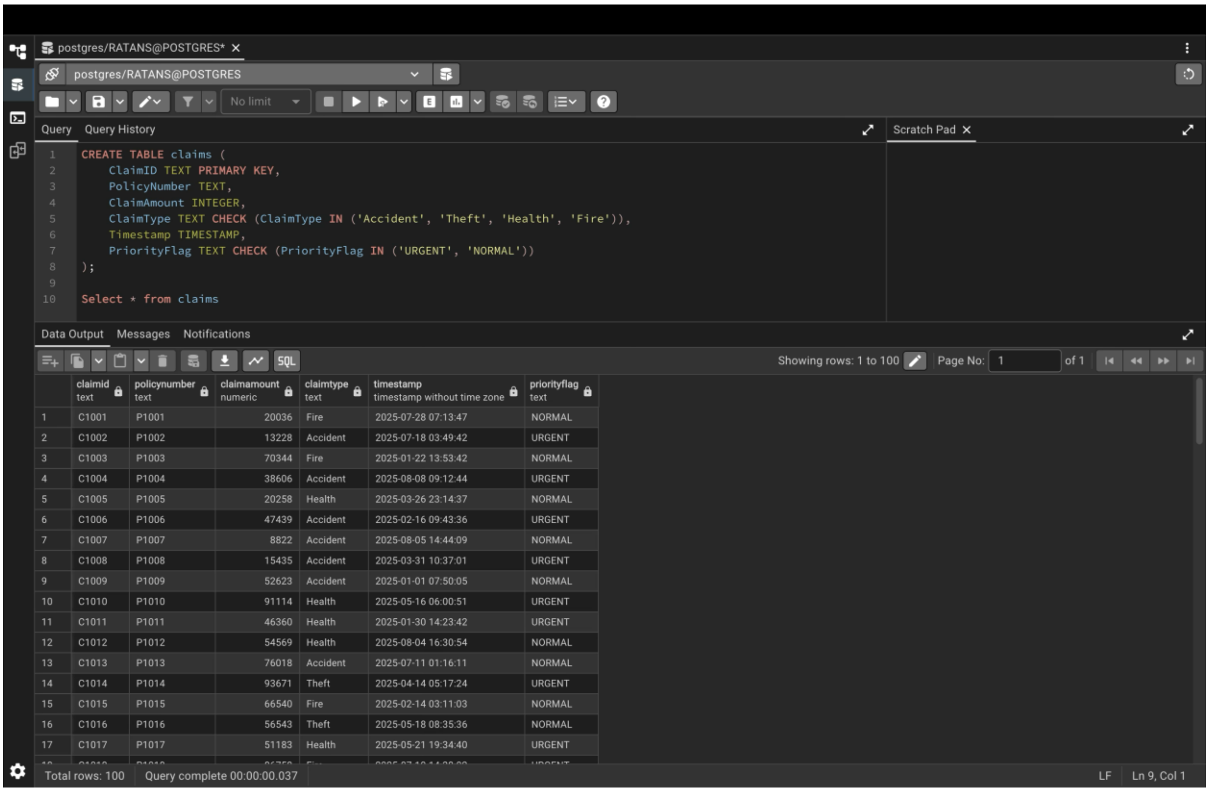
STEPS followed to create the database and table:

For database

1. Open pgAdmin.
2. Right-click on Databases → Create → Database.

For Tables

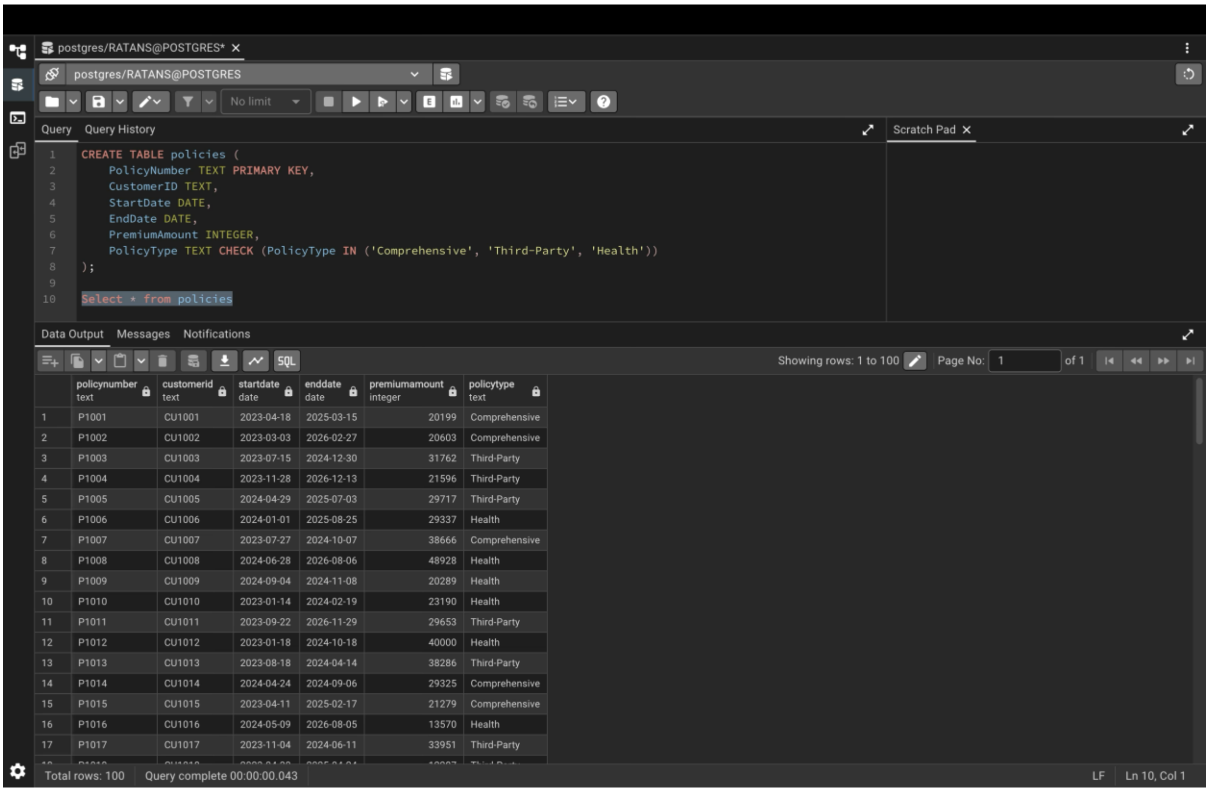
1. Create the table with a SQL CREATE TABLE statement matching the CSV structure.
2. Right-click the table you just created → Select Import/Export.
3. In the Import/Export dialog:
4. Set Filename to your CSV file path.
5. Choose Import as the option.
6. Select CSV format.
7. Check Header (if your CSV has column names in the first row).
8. Set Delimiter (usually a comma,).
9. Click OK to import the data.
10. Run a SELECT * FROM table name to confirm the import was successful.



```sql
CREATE TABLE claims (
    ClaimID TEXT PRIMARY KEY,
    PolicyNumber TEXT,
    ClaimAmount INTEGER,
    ClaimType TEXT CHECK (ClaimType IN ('Accident', 'Theft', 'Health', 'Fire')),
    Timestamp TIMESTAMP,
    PriorityFlag TEXT CHECK (PriorityFlag IN ('URGENT', 'NORMAL'))
);

Select * from claims
```

	claimid text	policynumber text	claimamount numeric	claimtype text	timestamp timestamp without time zone	priorityflag text
1	C1001	P1001	20036	Fire	2025-07-28 07:13:47	NORMAL
2	C1002	P1002	13228	Accident	2025-07-18 03:49:42	URGENT
3	C1003	P1003	70344	Fire	2025-01-22 13:53:42	NORMAL
4	C1004	P1004	38606	Accident	2025-08-08 09:12:44	URGENT
5	C1005	P1005	20258	Health	2025-03-26 23:14:37	NORMAL
6	C1006	P1006	47439	Accident	2025-02-16 09:43:36	URGENT
7	C1007	P1007	8822	Accident	2025-08-05 14:44:09	NORMAL
8	C1008	P1008	15435	Accident	2025-03-31 10:37:01	URGENT
9	C1009	P1009	52623	Accident	2025-01-01 07:50:05	NORMAL
10	C1010	P1010	91114	Health	2025-05-16 06:00:51	URGENT
11	C1011	P1011	46360	Health	2025-01-30 14:23:42	URGENT
12	C1012	P1012	54569	Health	2025-08-04 16:30:54	NORMAL
13	C1013	P1013	76018	Accident	2025-07-11 01:16:11	NORMAL
14	C1014	P1014	93671	Theft	2025-04-14 05:17:24	URGENT
15	C1015	P1015	66540	Fire	2025-02-14 03:11:03	NORMAL
16	C1016	P1016	56543	Theft	2025-05-18 08:35:36	NORMAL
17	C1017	P1017	51183	Health	2025-05-21 19:34:40	URGENT

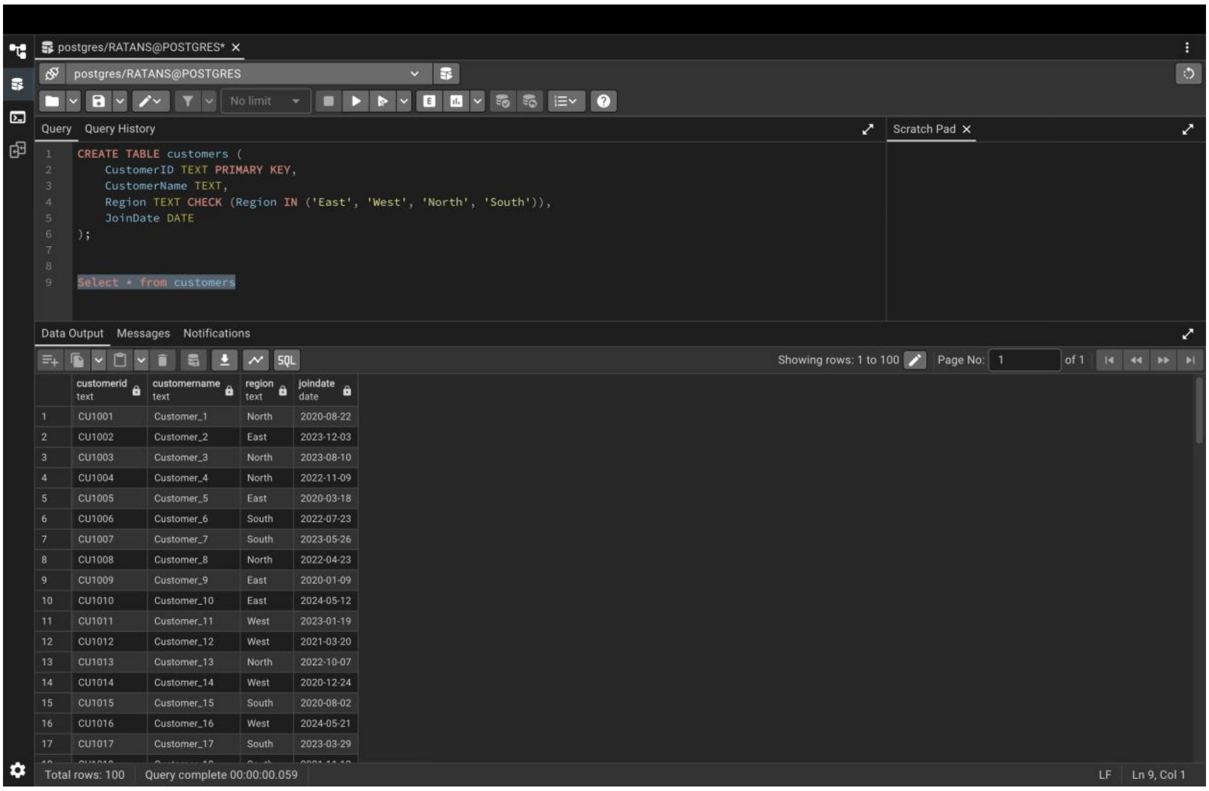
Total rows: 100 Query complete 00:00:00.037



```sql
CREATE TABLE policies (
    PolicyNumber TEXT PRIMARY KEY,
    CustomerID TEXT,
    StartDate DATE,
    EndDate DATE,
    PremiumAmount INTEGER,
    PolicyType TEXT CHECK (PolicyType IN ('Comprehensive', 'Third-Party', 'Health'))
);


Select * from policies
```

	policynumber text	customerid text	startdate date	enddate date	premiumamount integer	policytype text
1	P1001	CU1001	2023-04-18	2025-03-15	20199	Comprehensive
2	P1002	CU1002	2023-03-03	2026-02-27	20603	Comprehensive
3	P1003	CU1003	2023-07-15	2024-12-30	31762	Third-Party
4	P1004	CU1004	2023-11-28	2026-12-13	21596	Third-Party
5	P1005	CU1005	2024-04-29	2025-07-03	29717	Third-Party
6	P1006	CU1006	2024-01-01	2025-08-25	29337	Health
7	P1007	CU1007	2023-07-27	2024-10-07	38666	Comprehensive
8	P1008	CU1008	2024-06-28	2026-08-06	48928	Health
9	P1009	CU1009	2024-09-04	2024-11-08	20289	Health
10	P1010	CU1010	2023-01-14	2024-02-19	23190	Health
11	P1011	CU1011	2023-09-22	2026-11-29	29653	Third-Party
12	P1012	CU1012	2023-01-18	2024-10-18	40000	Health
13	P1013	CU1013	2023-08-18	2024-04-14	38286	Third-Party
14	P1014	CU1014	2024-04-24	2024-09-06	29325	Comprehensive
15	P1015	CU1015	2023-04-11	2025-02-17	21279	Comprehensive
16	P1016	CU1016	2024-05-09	2026-08-05	13570	Health
17	P1017	CU1017	2023-11-04	2024-06-11	33951	Third-Party

Total rows: 100 Query complete 00:00:00.043



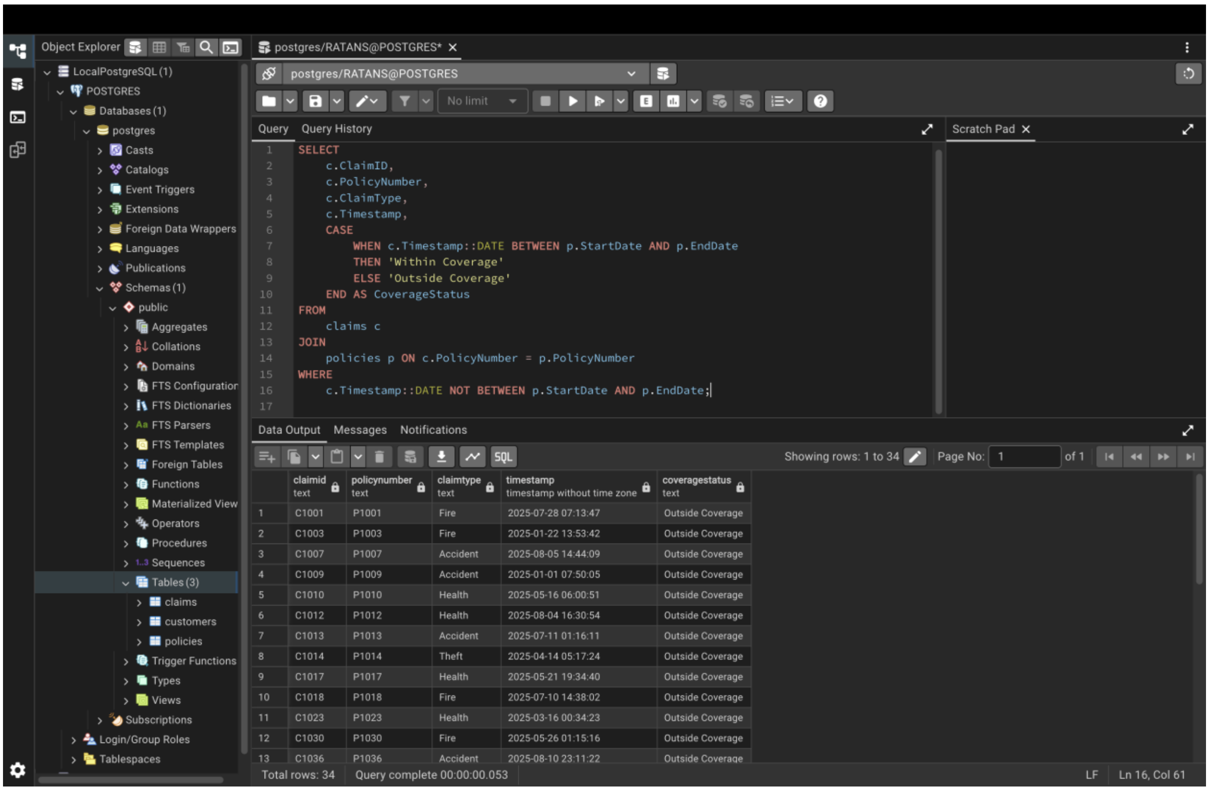
```sql
CREATE TABLE customers (
    CustomerID TEXT PRIMARY KEY,
    CustomerName TEXT,
    Region TEXT CHECK (Region IN ('East', 'West', 'North', 'South')),
    JoinDate DATE
);



Select * from customers
```

	customerid text	customername text	region text	joindate date
1	CU1001	Customer_1	North	2020-08-22
2	CU1002	Customer_2	East	2023-12-03
3	CU1003	Customer_3	North	2023-08-10
4	CU1004	Customer_4	North	2022-11-09
5	CU1005	Customer_5	East	2020-03-18
6	CU1006	Customer_6	South	2022-07-23
7	CU1007	Customer_7	South	2023-05-26
8	CU1008	Customer_8	North	2022-04-23
9	CU1009	Customer_9	East	2020-01-09
10	CU1010	Customer_10	East	2024-05-12
11	CU1011	Customer_11	West	2023-01-19
12	CU1012	Customer_12	West	2021-03-20
13	CU1013	Customer_13	North	2022-10-07
14	CU1014	Customer_14	West	2020-12-24
15	CU1015	Customer_15	South	2020-08-02
16	CU1016	Customer_16	West	2024-05-21
17	CU1017	Customer_17	South	2023-03-29

Total rows: 100 Query complete 00:00:00.059

Query 1:

Policy Coverage Gap Detection Find all claims that were made outside their policy coverage period (Timestamp not between StartDate and EndDate). Output: ClaimID, PolicyNumber, ClaimType, Timestamp, CoverageStatus (either "Within Coverage" or "Outside Coverage"). (Hint: Requires join + conditional logic.)



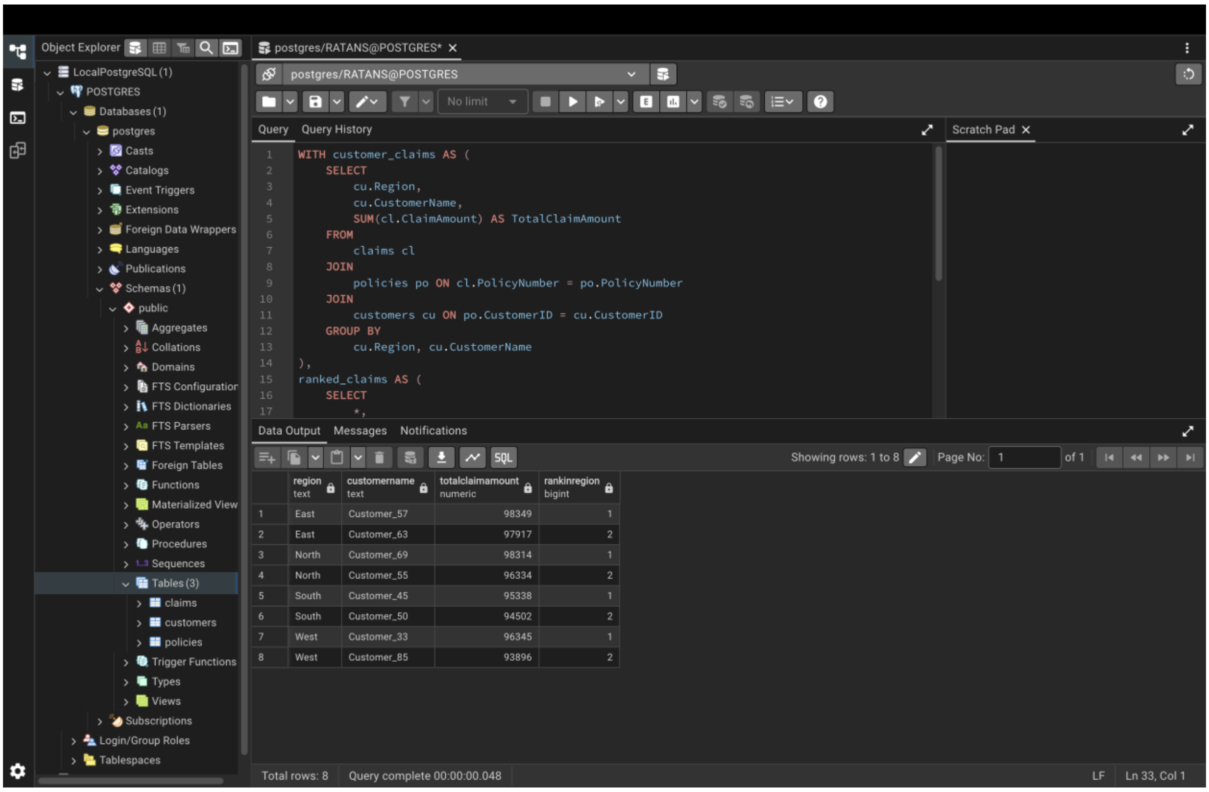
```sql
SELECT
    c.ClaimID,
    c.PolicyNumber,
    c.ClaimType,
    c.Timestamp,
    CASE
        WHEN c.Timestamp::DATE BETWEEN p.StartDate AND p.EndDate
        THEN 'Within Coverage'
        ELSE 'Outside Coverage'
    END AS CoverageStatus
FROM
    claims c
JOIN
    policies p ON c.PolicyNumber = p.PolicyNumber
WHERE
    c.Timestamp::DATE NOT BETWEEN p.StartDate AND p.EndDate;
```

	claimid text	policynumber text	claimtype text	timestamp timestamp without time zone	coveragestatus text
1	C1001	P1001	Fire	2025-07-28 07:13:47	Outside Coverage
2	C1003	P1003	Fire	2025-01-22 13:53:42	Outside Coverage
3	C1007	P1007	Accident	2025-08-05 14:44:09	Outside Coverage
4	C1009	P1009	Accident	2025-01-01 07:50:05	Outside Coverage
5	C1010	P1010	Health	2025-05-16 06:00:51	Outside Coverage
6	C1012	P1012	Health	2025-08-04 16:30:54	Outside Coverage
7	C1013	P1013	Accident	2025-07-11 01:16:11	Outside Coverage
8	C1014	P1014	Theft	2025-04-14 05:17:24	Outside Coverage
9	C1017	P1017	Health	2025-05-21 19:34:40	Outside Coverage
10	C1018	P1018	Fire	2025-07-10 14:38:02	Outside Coverage
11	C1023	P1023	Health	2025-03-16 00:34:23	Outside Coverage
12	C1030	P1030	Fire	2025-05-26 01:15:16	Outside Coverage
13	C1036	P1036	Accident	2025-08-10 23:11:22	Outside Coverage

Total rows: 34 Query complete 00:00:00.053

Query 2:

Top Claimants by Region For each region, list the top 2 customers based on the total ClaimAmount. Output: Region, CustomerName, TotalClaimAmount, RankInRegion. (Hint: Use window functions.)



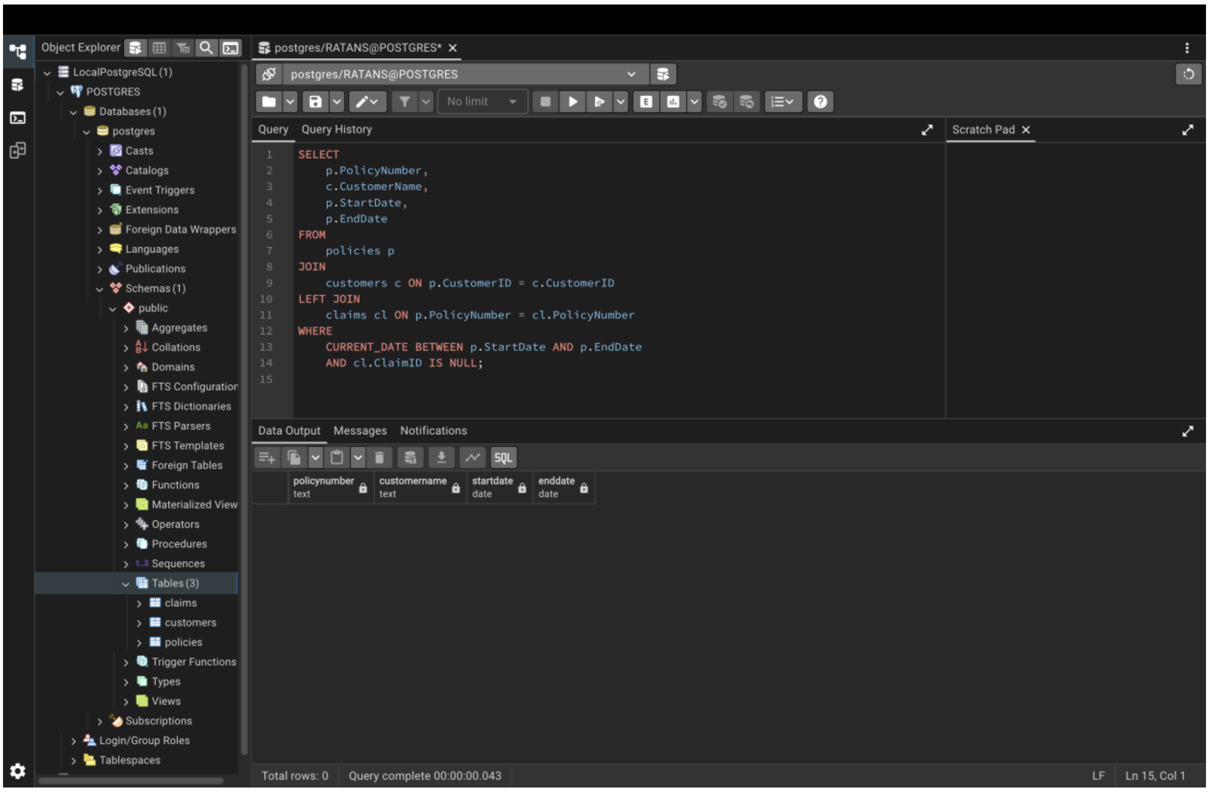
```sql
WITH customer_claims AS (
    SELECT
        cu.Region,
        cu.CustomerName,
        SUM(cl.ClaimAmount) AS TotalClaimAmount
    FROM
        claims cl
    JOIN
        policies po ON cl.PolicyNumber = po.PolicyNumber
    JOIN
        customers cu ON po.CustomerID = cu.CustomerID
    GROUP BY
        cu.Region, cu.CustomerName
),
ranked_claims AS (
    SELECT
        *,
```

	region text	customername text	totalclaimamount numeric	rankinregion bigint
1	East	Customer_57	98349	1
2	East	Customer_63	97917	2
3	North	Customer_69	98314	1
4	North	Customer_55	96334	2
5	South	Customer_45	95338	1
6	South	Customer_50	94502	2
7	West	Customer_33	96345	1
8	West	Customer_85	93896	2

Total rows: 8 Query complete 00:00:00.048

Query 3:

Unclaimed Active Policies Find all policies that are still active today (current date between StartDate and EndDate) but have never had a claim. Output: PolicyNumber, CustomerName, StartDate, EndDate.



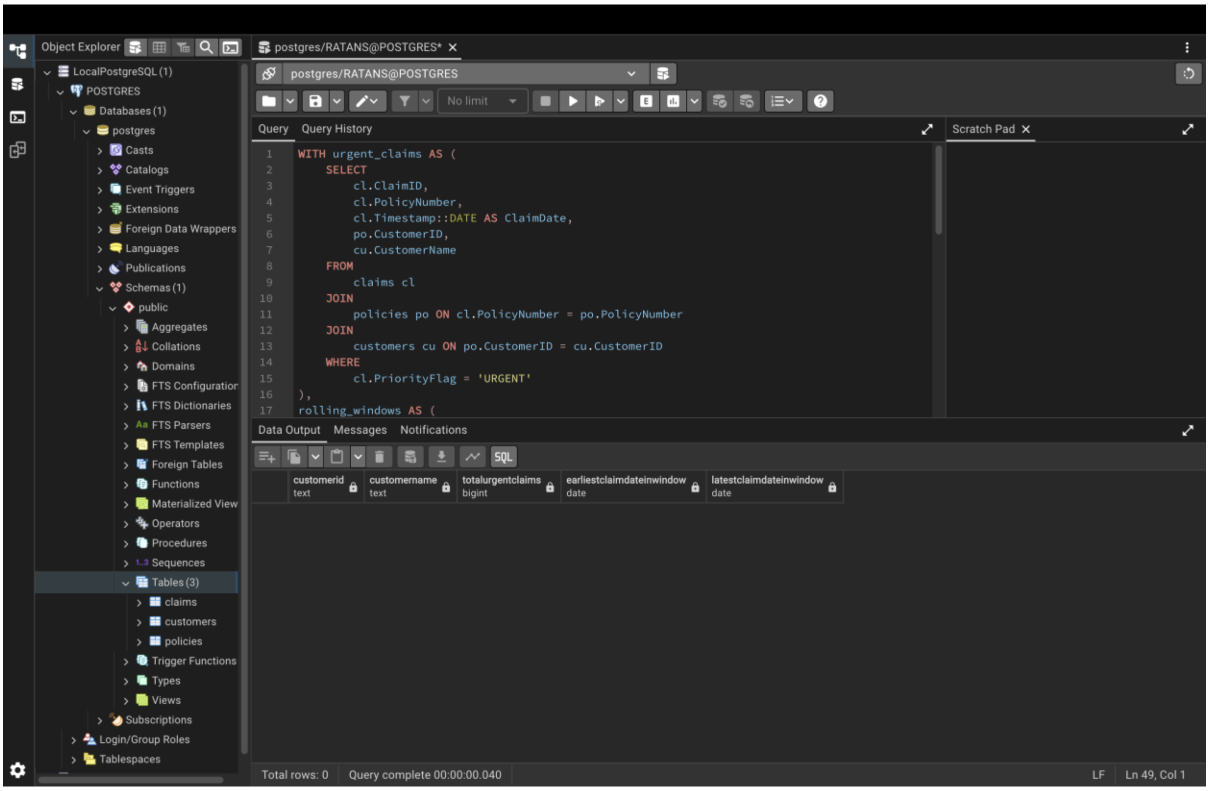
```
SELECT
    p.PolicyNumber,
    c.CustomerName,
    p.StartDate,
    p.EndDate
FROM
    policies p
JOIN
    customers c ON p.CustomerID = c.CustomerID
LEFT JOIN
    claims cl ON p.PolicyNumber = cl.PolicyNumber
WHERE
    CURRENT_DATE BETWEEN p.StartDate AND p.EndDate
    AND cl.ClaimID IS NULL;
```

policynumber text	customername text	startdate date	enddate date

Total rows: 0 Query complete 00:00:00.043

Query 4

Suspicious High-Priority Patterns Find customers who have made more than 2 "URGENT" claims within any 30-day rolling window. Output: CustomerID, CustomerName, TotalUrgentClaims, EarliestClaimDateInWindow, LatestClaimDateInWindow. (Hint: Rolling time windows with self-joins or window functions.)



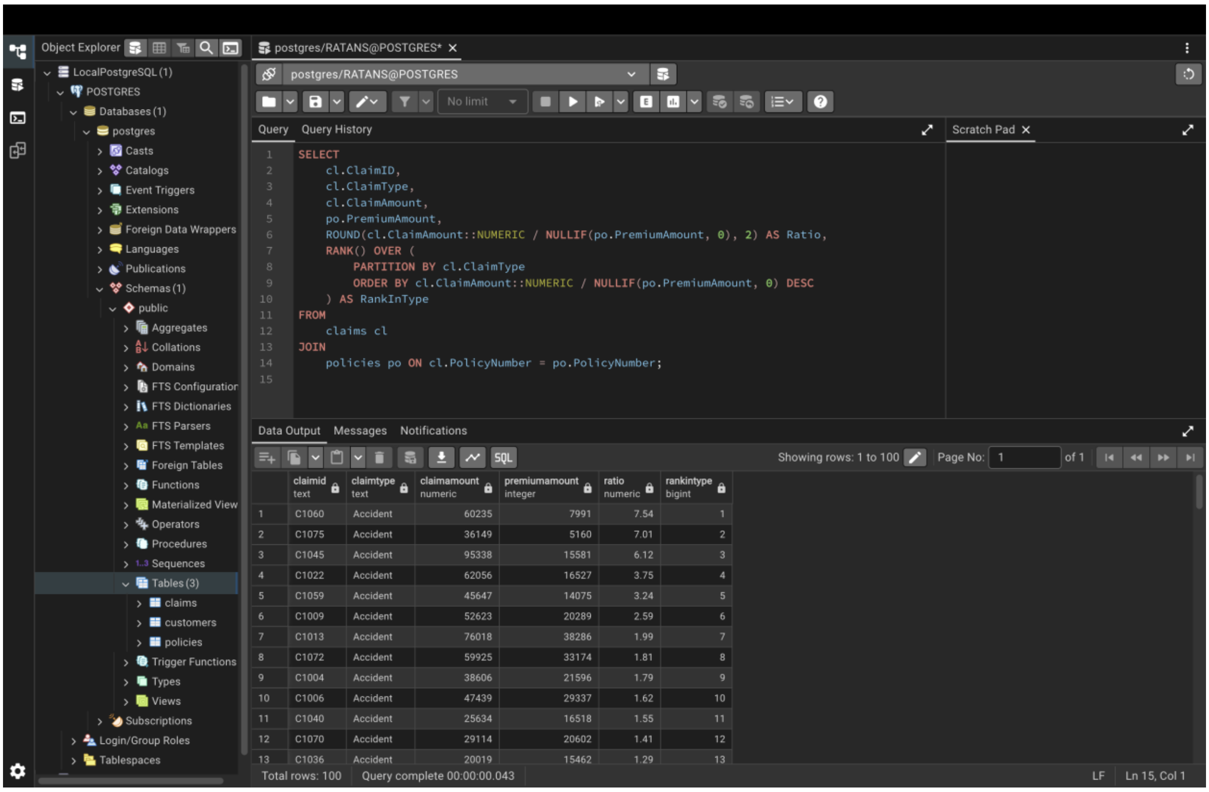
```sql
WITH urgent_claims AS (
    SELECT
        cl.ClaimID,
        cl.PolicyNumber,
        cl.Timestamp::DATE AS ClaimDate,
        po.CustomerID,
        cu.CustomerName
    FROM
        claims cl
    JOIN
        policies po ON cl.PolicyNumber = po.PolicyNumber
    JOIN
        customers cu ON po.CustomerID = cu.CustomerID
    WHERE
        cl.PriorityFlag = 'URGENT'
),
rolling_windows AS (
```

customerid	customername	totalurgentclaims	earliestclaimdateinwindow	latestclaimdateinwindow
text	text	bigint	date	date

Total rows: 0 Query complete 00:00:00.040 LF Ln 49, Col 1

Query 5:

Claim Amount vs. Premium Ratio For each claim, calculate the ratio of ClaimAmount to the policy's PremiumAmount, then rank claims in descending ratio order per ClaimType. Output: ClaimID, ClaimType, ClaimAmount, PremiumAmount, Ratio, RankInType.



```sql
SELECT
    cl.ClaimID,
    cl.ClaimType,
    cl.ClaimAmount,
    po.PremiumAmount,
    ROUND(cl.ClaimAmount::NUMERIC / NULLIF(po.PremiumAmount, 0), 2) AS Ratio,
    RANK() OVER (
        PARTITION BY cl.ClaimType
        ORDER BY cl.ClaimAmount::NUMERIC / NULLIF(po.PremiumAmount, 0) DESC
    ) AS RankInType
FROM
    claims cl
JOIN
    policies po ON cl.PolicyNumber = po.PolicyNumber;
```

	claimid text	claimtype text	claimamount numeric	premiumamount integer	ratio numeric	rankintype bigint
1	C1060	Accident	60235	7991	7.54	1
2	C1075	Accident	36149	5160	7.01	2
3	C1045	Accident	95338	15581	6.12	3
4	C1022	Accident	62056	16527	3.75	4
5	C1059	Accident	45647	14075	3.24	5
6	C1009	Accident	52623	20289	2.59	6
7	C1013	Accident	76018	38286	1.99	7
8	C1072	Accident	59925	33174	1.81	8
9	C1004	Accident	38606	21596	1.79	9
10	C1006	Accident	47439	29337	1.62	10
11	C1040	Accident	25634	16518	1.55	11
12	C1070	Accident	29114	20602	1.41	12
13	C1036	Accident	20019	15462	1.29	13

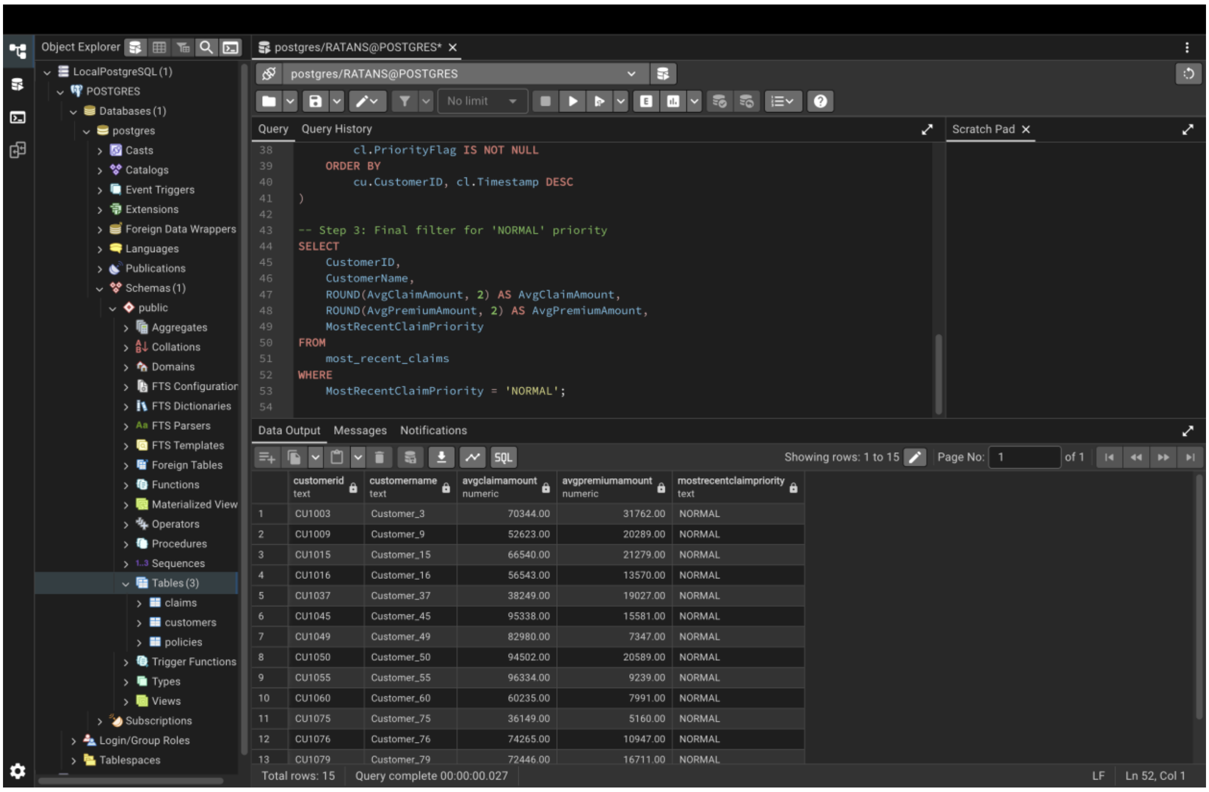
Showing rows: 1 to 100 Page No: 1 of 1

Total rows: 100 Query complete 00:00:00.043

Query 6:

CTE Challenge — Multi-step Filtering Use a CTE to first find customers whose average claim amount is greater than double their policy’s average premium amount. Then, from those customers, output only the ones whose most recent claim is "NORMAL" priority.

Output: CustomerID, CustomerName, AvgClaimAmount, AvgPremiumAmount, MostRecentClaimPriority.



```
        cl.PriorityFlag IS NOT NULL
    ORDER BY
        cu.CustomerID, cl.Timestamp DESC
)

-- Step 3: Final filter for 'NORMAL' priority
SELECT
    CustomerID,
    CustomerName,
    ROUND(AvgClaimAmount, 2) AS AvgClaimAmount,
    ROUND(AvgPremiumAmount, 2) AS AvgPremiumAmount,
    MostRecentClaimPriority
FROM
    most_recent_claims
WHERE
    MostRecentClaimPriority = 'NORMAL';
```

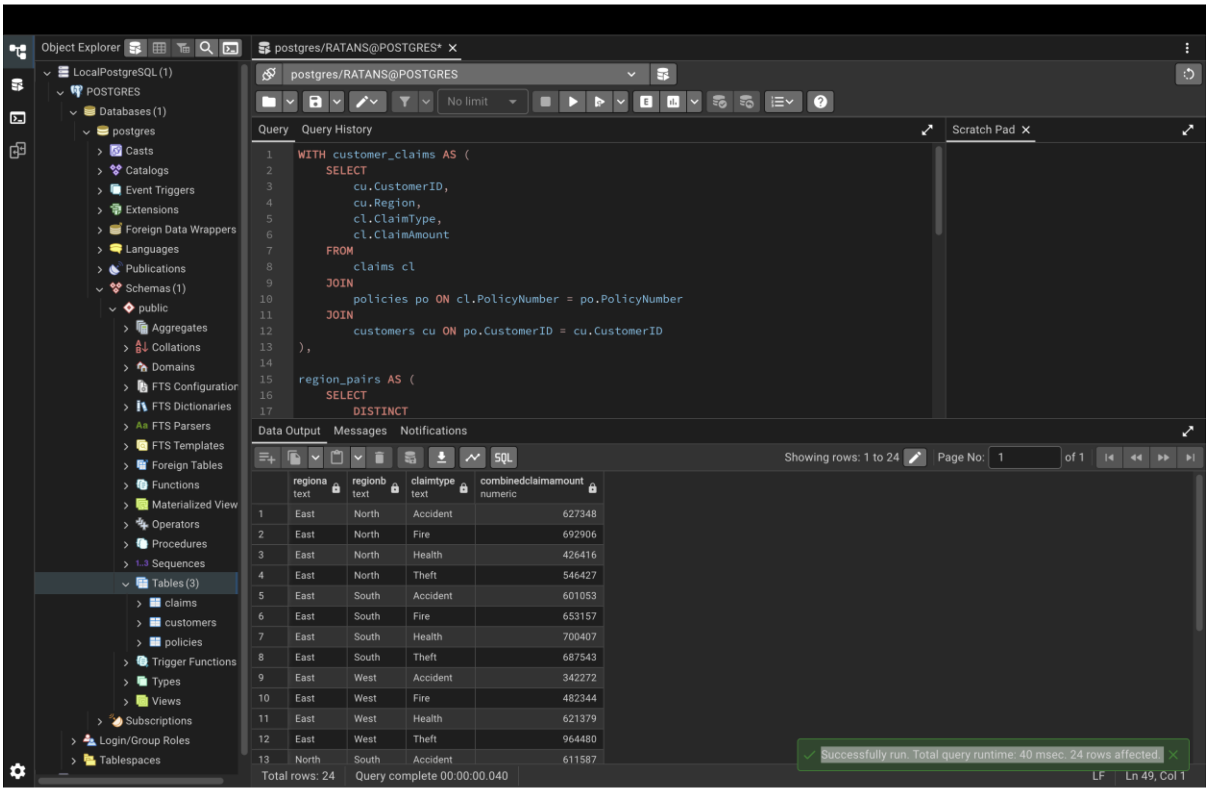
Showing rows: 1 to 15

	customerid text	customername text	avgclaimamount numeric	avgpremiumamount numeric	mostrecentclaimpriority text
1	CU1003	Customer_3	70344.00	31762.00	NORMAL
2	CU1009	Customer_9	52623.00	20289.00	NORMAL
3	CU1015	Customer_15	66540.00	21279.00	NORMAL
4	CU1016	Customer_16	56543.00	13570.00	NORMAL
5	CU1037	Customer_37	38249.00	19027.00	NORMAL
6	CU1045	Customer_45	95338.00	15581.00	NORMAL
7	CU1049	Customer_49	82980.00	7347.00	NORMAL
8	CU1050	Customer_50	94502.00	20589.00	NORMAL
9	CU1055	Customer_55	96334.00	9239.00	NORMAL
10	CU1060	Customer_60	60235.00	7991.00	NORMAL
11	CU1075	Customer_75	36149.00	5160.00	NORMAL
12	CU1076	Customer_76	74265.00	10947.00	NORMAL
13	CU1079	Customer_79	72446.00	16711.00	NORMAL

Total rows: 15 Query complete 00:00:00.027

Query 7:

Cross Join Trick — Region Combination Claim Analysis Generate all possible pairs of different regions and, for each pair, find the combined total claim amount for customers from both regions in the same claim type. Output: RegionA, RegionB, ClaimType, CombinedClaimAmount. (Hint: Self join with region comparison.)



```sql
WITH customer_claims AS (
    SELECT
        cu.CustomerID,
        cu.Region,
        cl.ClaimType,
        cl.ClaimAmount
    FROM
        claims cl
    JOIN
        policies po ON cl.PolicyNumber = po.PolicyNumber
    JOIN
        customers cu ON po.CustomerID = cu.CustomerID
),

region_pairs AS (
    SELECT
        DISTINCT
```

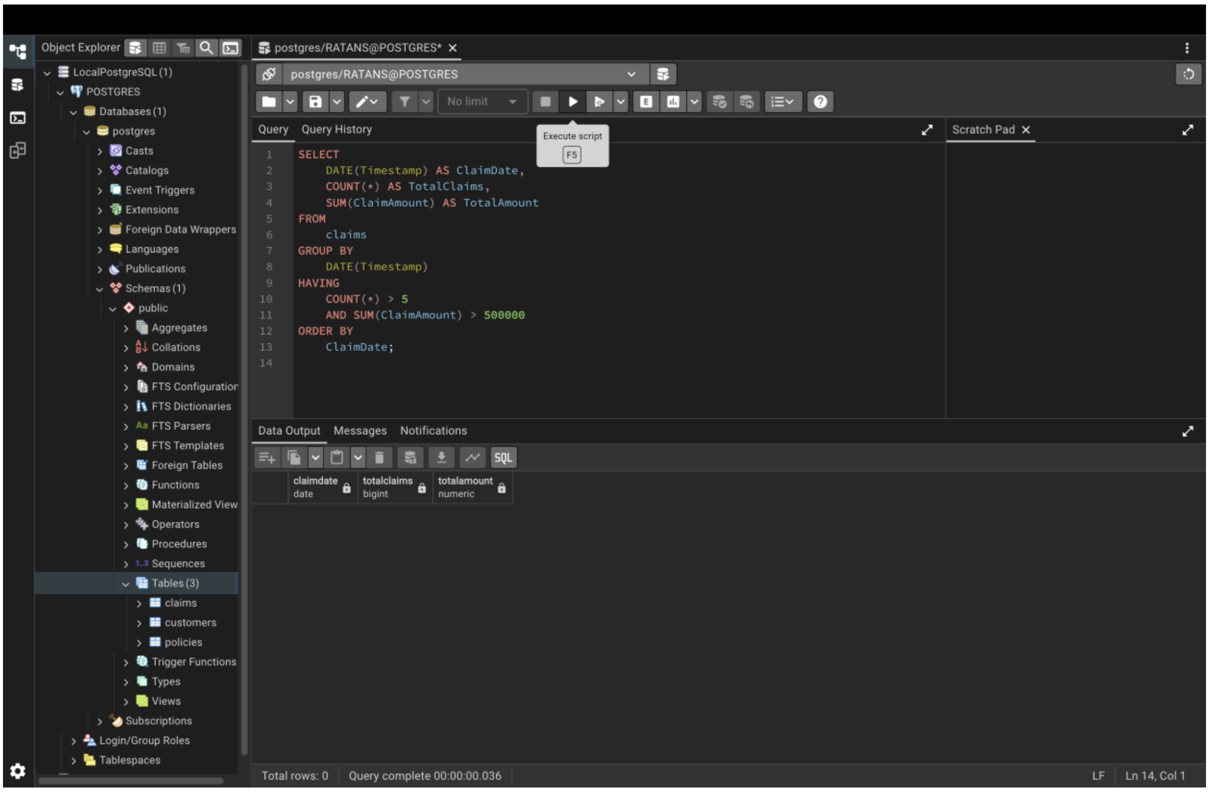
	regiona text	regionb text	claimtype text	combinedclaimamount numeric
1	East	North	Accident	627348
2	East	North	Fire	692906
3	East	North	Health	426416
4	East	North	Theft	546427
5	East	South	Accident	601053
6	East	South	Fire	653157
7	East	South	Health	700407
8	East	South	Theft	687543
9	East	West	Accident	342272
10	East	West	Fire	482344
11	East	West	Health	621379
12	East	West	Theft	964480
13	North	South	Accident	611587

Successfully run. Total query runtime: 40 msec. 24 rows affected.

Total rows: 24 Query complete 00:00:00.040

Query 8:

Claim Clusters by Date Group claims by claim date (ignoring time) and find the dates where more than 5 claims occurred and the total amount exceeded ₹5,00,000. Output: ClaimDate, TotalClaims, TotalAmount.



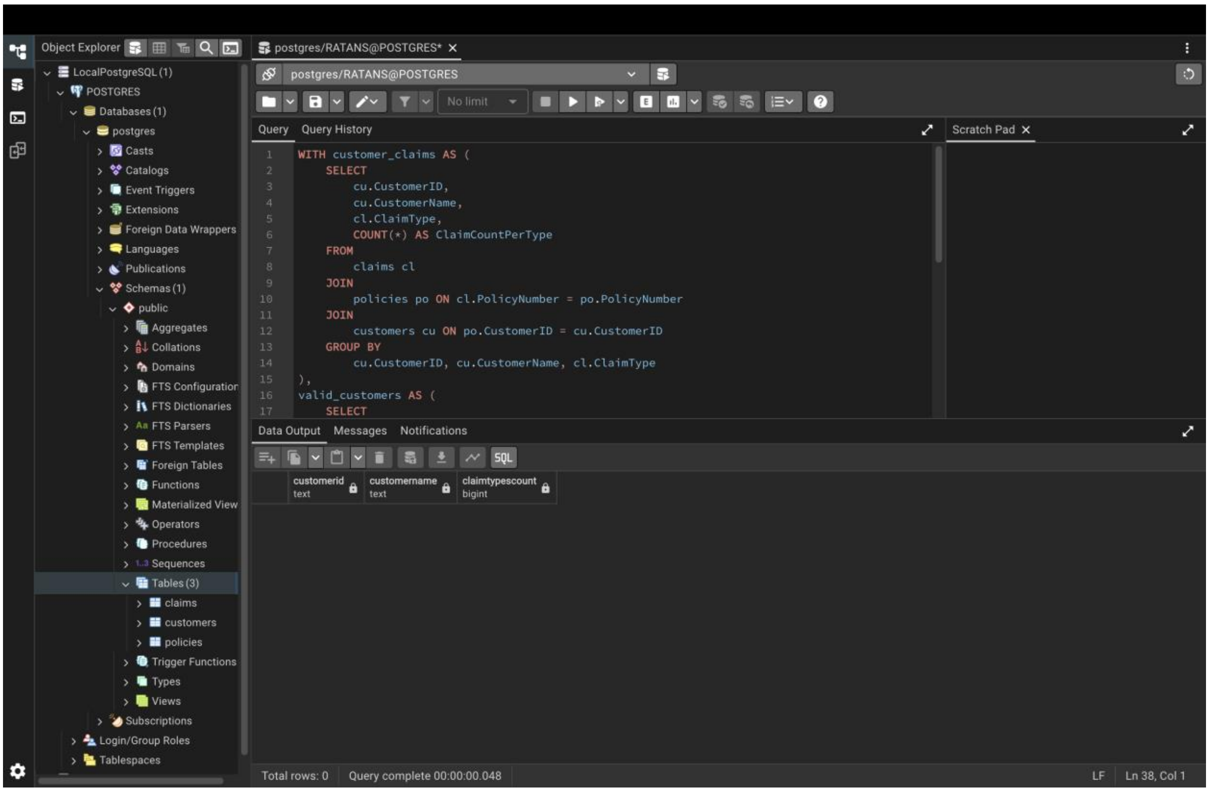
```
SELECT
    DATE(Timestamp) AS ClaimDate,
    COUNT(*) AS TotalClaims,
    SUM(ClaimAmount) AS TotalAmount
FROM
    claims
GROUP BY
    DATE(Timestamp)
HAVING
    COUNT(*) > 5
    AND SUM(ClaimAmount) > 500000
ORDER BY
    ClaimDate;
```

claimdate date	totalclaims bigint	totalamount numeric

Total rows: 0 Query complete 00:00:00.036 LF Ln 14, Col 1

Query 9:

Complex Subquery Puzzle Find customers who have never claimed in the same ClaimType twice but have at least one claim in each of three different claim types. Output: CustomerID, CustomerName, ClaimTypesCount.



```sql
WITH customer_claims AS (
    SELECT
        cu.CustomerID,
        cu.CustomerName,
        cl.ClaimType,
        COUNT(*) AS ClaimCountPerType
    FROM
        claims cl
    JOIN
        policies po ON cl.PolicyNumber = po.PolicyNumber
    JOIN
        customers cu ON po.CustomerID = cu.CustomerID
    GROUP BY
        cu.CustomerID, cu.CustomerName, cl.ClaimType
),
valid_customers AS (
    SELECT
```

```
customerid   customername   claimtypescount
text         text           bigint

Total rows: 0    Query complete 00:00:00.048    LF    Ln 38, Col 1
```

Query 10:

Advanced Normalization Check (Theoretical) Given the three datasets, explain in detail: Whether they follow 1NF, 2NF, and 3NF. If any normalization issues exist, suggest how to fix them.

Answer:

First Normal Form (1NF)

All three tables satisfy 1NF because their fields contain atomic, indivisible values with no repeating groups or arrays. For example:

- The **customers** table contains single values for CustomerID, CustomerName, Region, and JoinDate.
- The **policies** table stores atomic values such as PolicyNumber, CustomerID, StartDate, EndDate, PremiumAmount, and PolicyType.
- The **claims** table includes ClaimID, PolicyNumber, ClaimAmount, ClaimType, Timestamp, and PriorityFlag, all of which are atomic.

Thus, there are no multi-valued attributes or nested data structures, meeting the requirements of 1NF.

Second Normal Form (2NF)

Each table also satisfies 2NF because they are all in 1NF and have no partial dependencies. Specifically:

- In **customers**, the primary key is CustomerID, and all other columns depend fully on it.
- In **policies**, PolicyNumber is the primary key, and attributes like CustomerID, StartDate, EndDate, PremiumAmount, and PolicyType depend entirely on it.
- In **claims**, ClaimID uniquely identifies each row, with all other attributes depending fully on this key.

There are no cases where an attribute depends on only part of a composite key, so 2NF is satisfied.

Third Normal Form (3NF)

The 3NF evaluation checks for transitive dependencies—where non-key attributes depend on other non-key attributes rather than the primary key.

- **Customers** meet 3NF since CustomerName, Region, and JoinDate depend only on CustomerID.
- **Claims** also meet 3NF as all attributes depend solely on ClaimID.
- However, the **policies** table may violate 3NF if **PremiumAmount** depends on **PolicyType** rather than directly on the primary key PolicyNumber. For instance, if each PolicyType corresponds to a fixed PremiumAmount, then PremiumAmount is transitively dependent through PolicyType.

Suggested Fix for Policies Table

To fully normalize the **policies** table to 3NF, you should separate the potentially transitive dependency by creating a new table for policy types and their standard premium amounts:

- Create a policy_types table with columns: PolicyType (primary key) and StandardPremiumAmount.
- Modify the policies table to reference policy_types via PolicyType, removing PremiumAmount from policies.

This eliminates the transitive dependency and ensures all attributes depend only on the primary key.