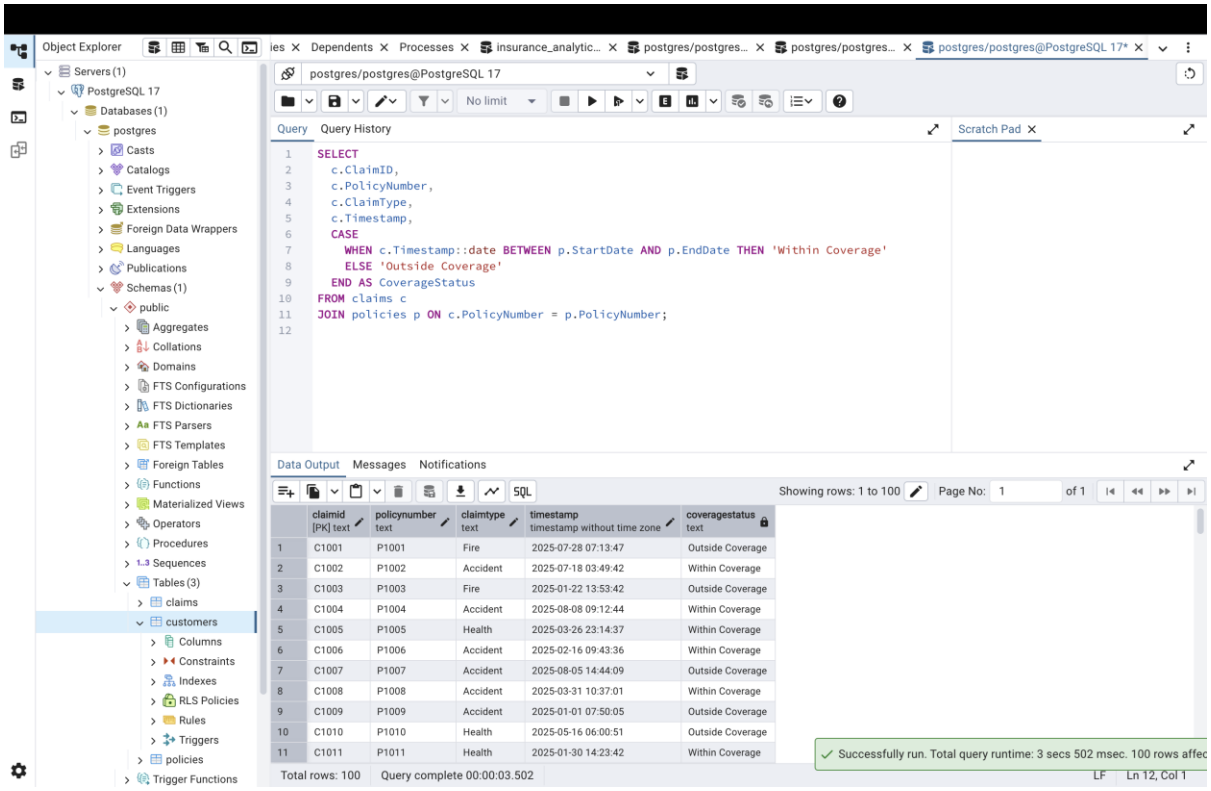
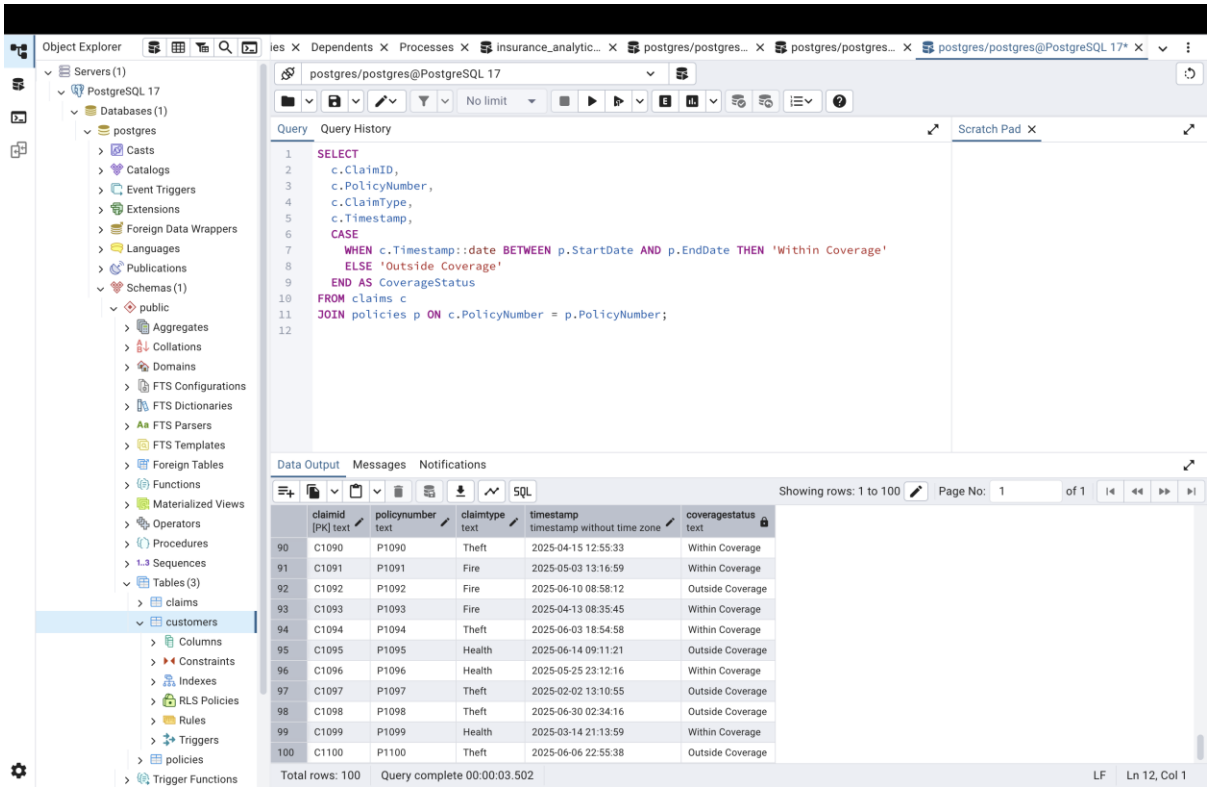
1. Policy Coverage Gap Detection

Find claims outside their policy coverage period.

```
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;
```


Object Explorer
Servers (1)
PostgreSQL 17
Databases (1)
postgres
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (3)
claims
customers
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
policies
Trigger Functions
Dependents
Processes
insurance_analytic...
postgres/postgres...
postgres/postgres@PostgreSQL 17*
postgres/postgres@PostgreSQL 17
No limit
Query
Query History
Scratch Pad
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;
Data Output
Messages
Notifications
Showing rows: 1 to 100
Page No: 1
of 1
claimid [PK] 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 C1002 P1002 Accident 2025-07-18 03:49:42 Within Coverage
3 C1003 P1003 Fire 2025-01-22 13:53:42 Outside Coverage
4 C1004 P1004 Accident 2025-08-08 09:12:44 Within Coverage
5 C1005 P1005 Health 2025-03-26 23:14:37 Within Coverage
6 C1006 P1006 Accident 2025-02-16 09:43:36 Within Coverage
7 C1007 P1007 Accident 2025-08-05 14:44:09 Outside Coverage
8 C1008 P1008 Accident 2025-03-31 10:37:01 Within Coverage
9 C1009 P1009 Accident 2025-01-01 07:50:05 Outside Coverage
10 C1010 P1010 Health 2025-05-16 06:00:51 Outside Coverage
11 C1011 P1011 Health 2025-01-30 14:23:42 Within Coverage
Successfully run. Total query runtime: 3 secs 502 msec. 100 rows affec
Total rows: 100
Query complete 00:00:03.502
LF
Ln 12, Col 1


Object Explorer
Servers (1)
PostgreSQL 17
Databases (1)
postgres
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (3)
claims
customers
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
policies
Trigger Functions
Dependents
Processes
insurance_analytic...
postgres/postgres...
postgres/postgres@PostgreSQL 17*
postgres/postgres@PostgreSQL 17
No limit
Query
Query History
Scratch Pad
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;
Data Output
Messages
Notifications
Showing rows: 1 to 100
Page No: 1
of 1
claimid [PK] text
policynumber text
claimtype text
timestamp timestamp without time zone
coveragestatus text
90 C1090 P1090 Theft 2025-04-15 12:55:33 Within Coverage
91 C1091 P1091 Fire 2025-05-03 13:16:59 Within Coverage
92 C1092 P1092 Fire 2025-06-10 08:58:12 Outside Coverage
93 C1093 P1093 Fire 2025-04-13 08:35:45 Within Coverage
94 C1094 P1094 Theft 2025-06-03 18:54:58 Within Coverage
95 C1095 P1095 Health 2025-06-14 09:11:21 Outside Coverage
96 C1096 P1096 Health 2025-05-25 23:12:16 Within Coverage
97 C1097 P1097 Theft 2025-02-02 13:10:55 Outside Coverage
98 C1098 P1098 Theft 2025-06-30 02:34:16 Outside Coverage
99 C1099 P1099 Health 2025-03-14 21:13:59 Within Coverage
100 C1100 P1100 Theft 2025-06-06 22:55:38 Outside Coverage
Total rows: 100
Query complete 00:00:03.502
LF
Ln 12, Col 1

2. Top Claimants by Region

Top 2 customers by total claim amount in each region.

```
WITH CustomerClaims AS (
  SELECT
    cu.Region,
    cu.CustomerName,
    SUM(c.ClaimAmount) AS TotalClaimAmount
  FROM claims c
  JOIN policies p ON c.PolicyNumber = p.PolicyNumber
  JOIN customers cu ON p.CustomerID = cu.CustomerID
  GROUP BY cu.Region, cu.CustomerName
),
RankedClaims AS (
  SELECT
    Region,
    CustomerName,
    TotalClaimAmount,
    RANK() OVER (PARTITION BY Region ORDER BY TotalClaimAmount DESC) AS
RankInRegion
  FROM CustomerClaims
)
SELECT
  Region,
```

CustomerName,

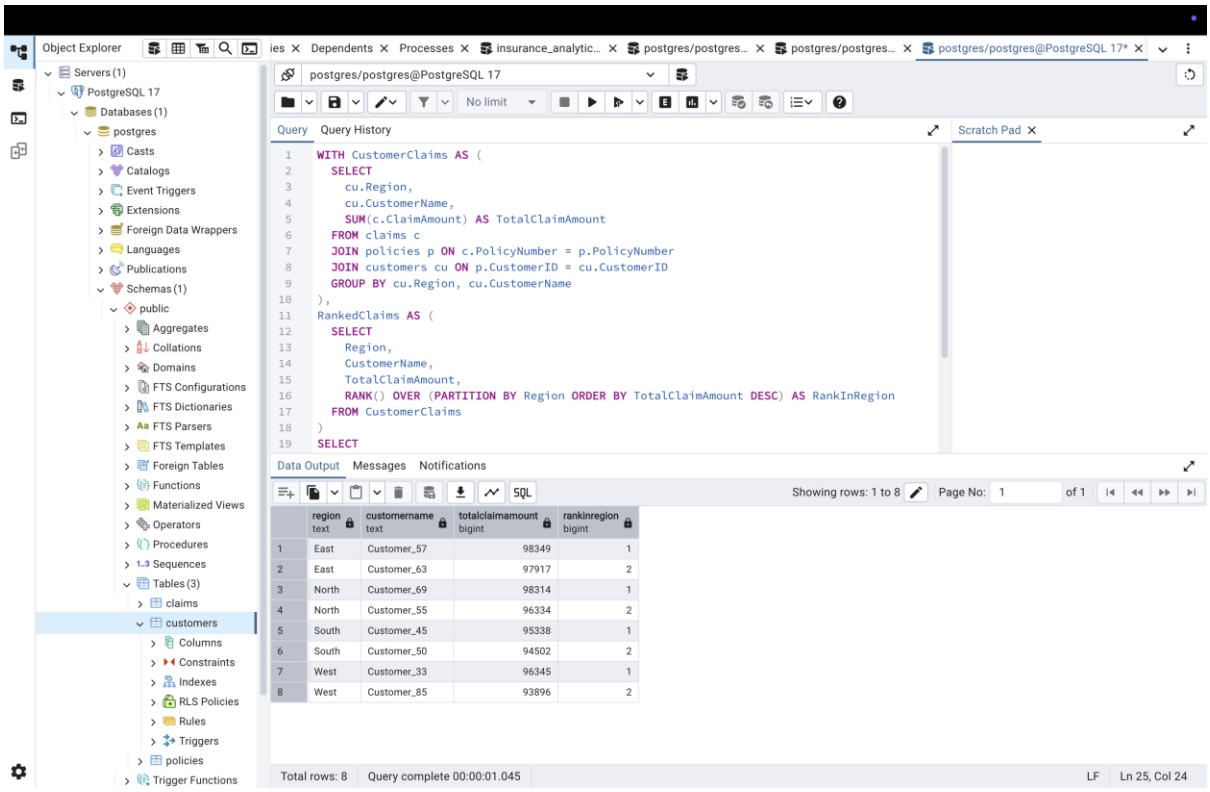
TotalClaimAmount,

RankInRegion

FROM RankedClaims

WHERE RankInRegion <= 2

ORDER BY Region, RankInRegion;



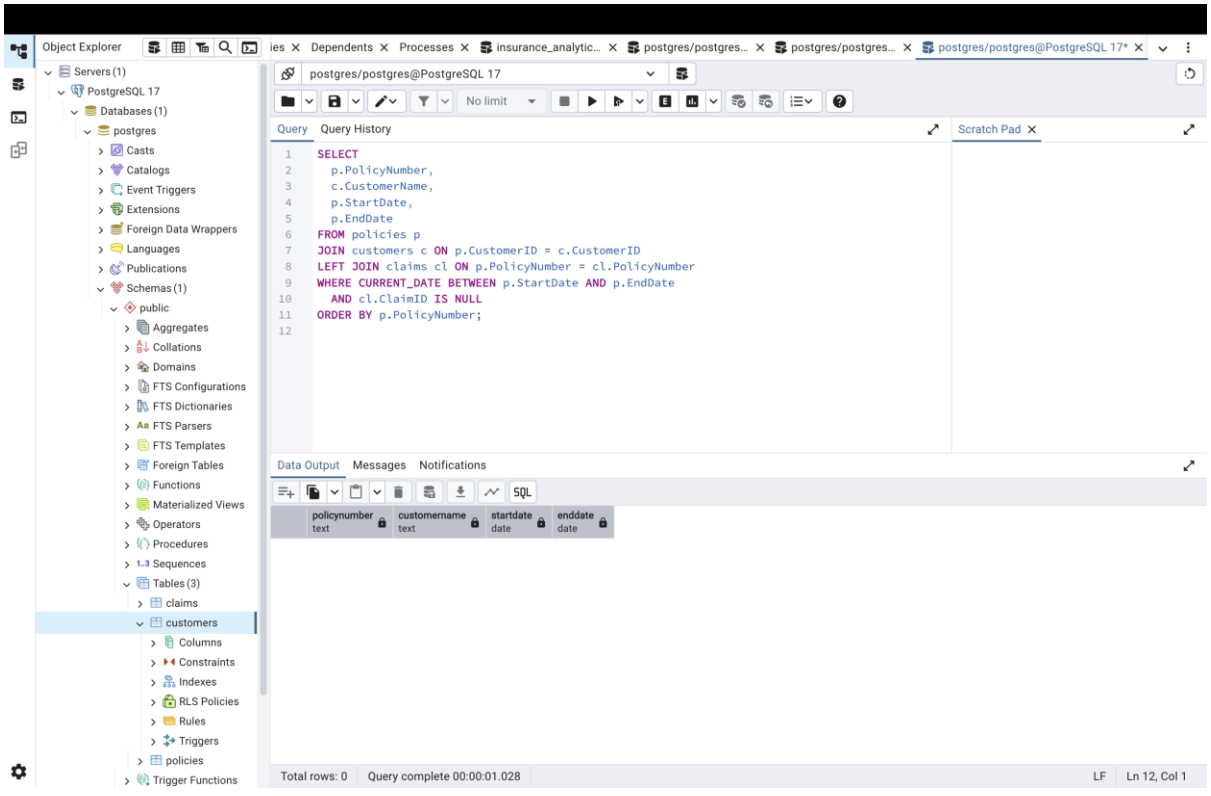
3) Unclaimed Active Policies

Policies active today but never had a claim:

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
ORDER BY p.PolicyNumber;
```

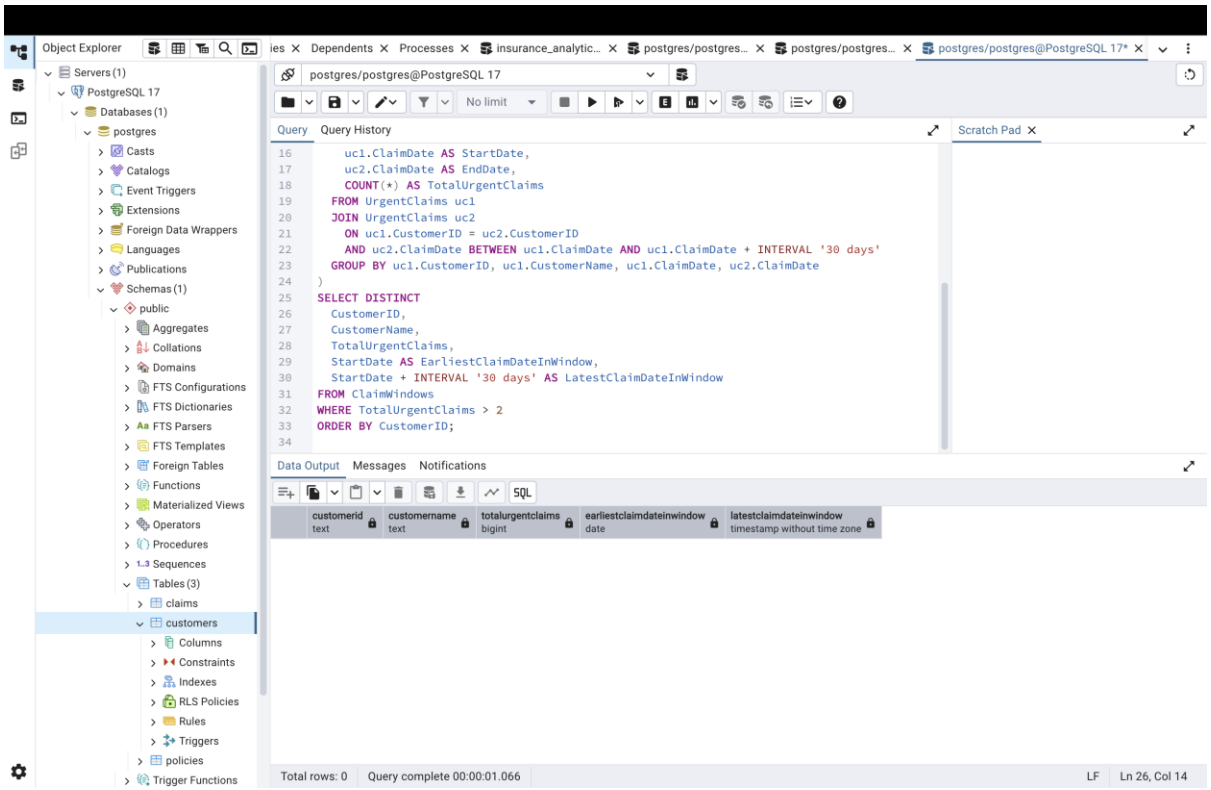


4) Suspicious High-Priority Patterns

Customers with >2 "URGENT" claims within any 30-day rolling window:

```
WITH UrgentClaims AS (
  SELECT
    p.CustomerID,
    c.CustomerName,
    cl.ClaimID,
    cl.Timestamp::date AS ClaimDate
  FROM claims cl
  JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
  JOIN customers c ON p.CustomerID = c.CustomerID
  WHERE cl.PriorityFlag = 'URGENT'
),
ClaimWindows AS (
  SELECT
    uc1.CustomerID,
    uc1.CustomerName,
    uc1.ClaimDate AS StartDate,
    uc2.ClaimDate AS EndDate,
    COUNT(*) AS TotalUrgentClaims
  FROM UrgentClaims uc1
  JOIN UrgentClaims uc2
    ON uc1.CustomerID = uc2.CustomerID
```

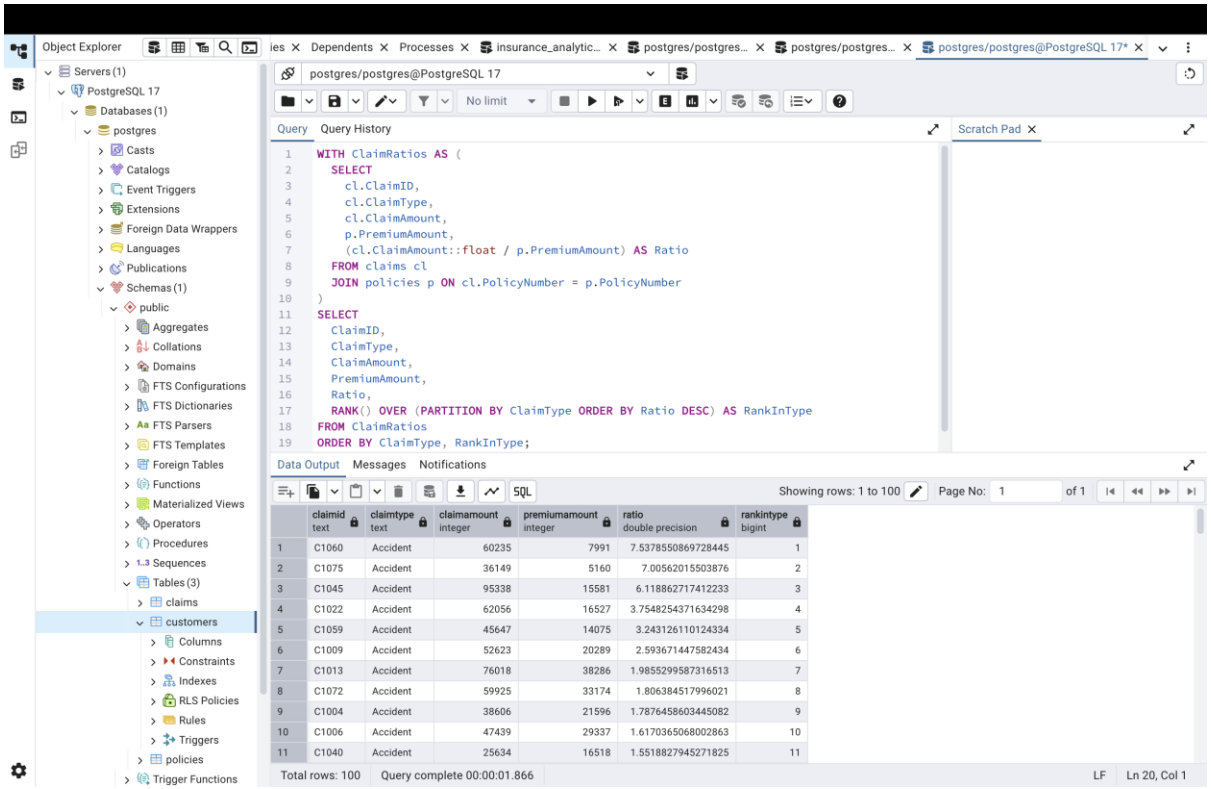
```
    AND uc2.ClaimDate BETWEEN uc1.ClaimDate AND uc1.ClaimDate + INTERVAL '30 days'
  GROUP BY uc1.CustomerID, uc1.CustomerName, uc1.ClaimDate, uc2.ClaimDate
)
SELECT DISTINCT
  CustomerID,
  CustomerName,
  TotalUrgentClaims,
  StartDate AS EarliestClaimDateInWindow,
  StartDate + INTERVAL '30 days' AS LatestClaimDateInWindow
FROM ClaimWindows
WHERE TotalUrgentClaims > 2
ORDER BY CustomerID;
```



5) Claim Amount vs. Premium Ratio

For each claim, ratio of ClaimAmount to PremiumAmount, rank within ClaimType:

```
WITH ClaimRatios AS (
  SELECT
    cl.ClaimID,
    cl.ClaimType,
    cl.ClaimAmount,
    p.PremiumAmount,
    (cl.ClaimAmount::float / p.PremiumAmount) AS Ratio
  FROM claims cl
  JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
)
SELECT
  ClaimID,
  ClaimType,
  ClaimAmount,
  PremiumAmount,
  Ratio,
  RANK() OVER (PARTITION BY ClaimType ORDER BY Ratio DESC) AS RankInType
FROM ClaimRatios
ORDER BY ClaimType, RankInType;
```

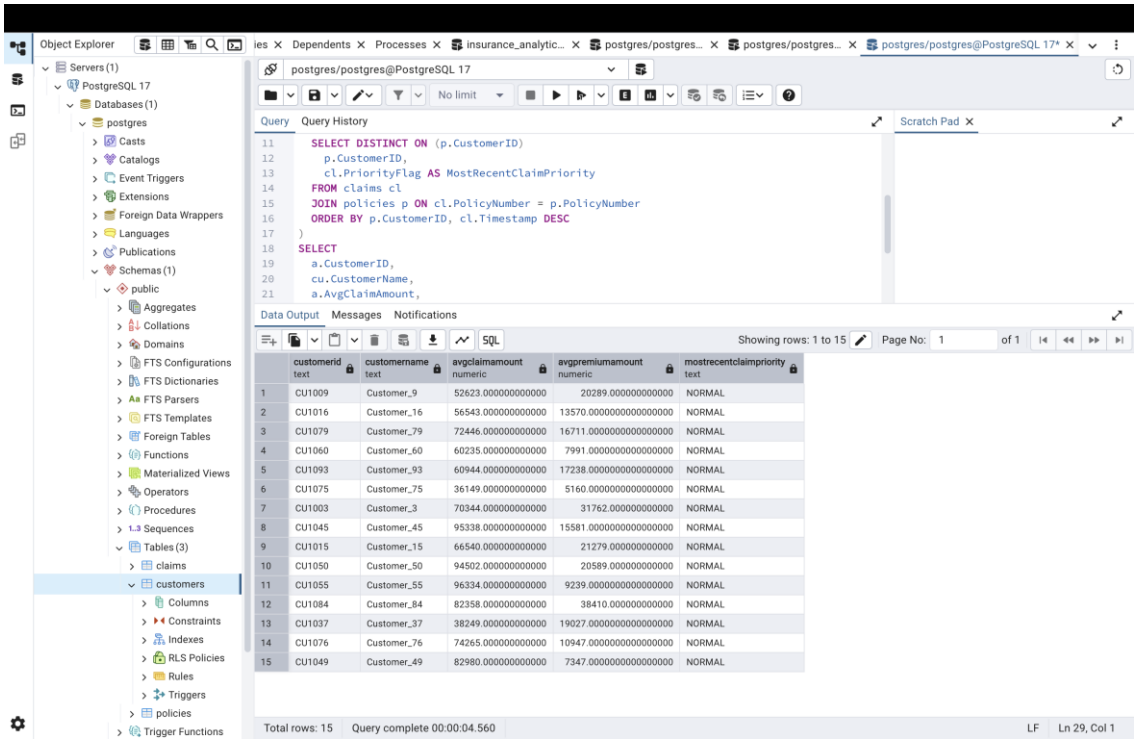


6) CTE Challenge — Multi-step Filtering

Customers whose avg claim amount > 2x avg premium and their most recent claim is "NORMAL":

```
WITH AvgAmounts AS (
  SELECT
    p.CustomerID,
    AVG(cl.ClaimAmount) AS AvgClaimAmount,
    AVG(p.PremiumAmount) AS AvgPremiumAmount
  FROM claims cl
  JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
  GROUP BY p.CustomerID
),
RecentPriority AS (
  SELECT DISTINCT ON (p.CustomerID)
```

```
    p.CustomerID,
    cl.PriorityFlag AS MostRecentClaimPriority
  FROM claims cl
  JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
  ORDER BY p.CustomerID, cl.Timestamp DESC
)
SELECT
  a.CustomerID,
  cu.CustomerName,
  a.AvgClaimAmount,
  a.AvgPremiumAmount,
  r.MostRecentClaimPriority
FROM AvgAmounts a
JOIN RecentPriority r ON a.CustomerID = r.CustomerID
JOIN customers cu ON a.CustomerID = cu.CustomerID
WHERE a.AvgClaimAmount > 2 * a.AvgPremiumAmount
  AND r.MostRecentClaimPriority = 'NORMAL';
```

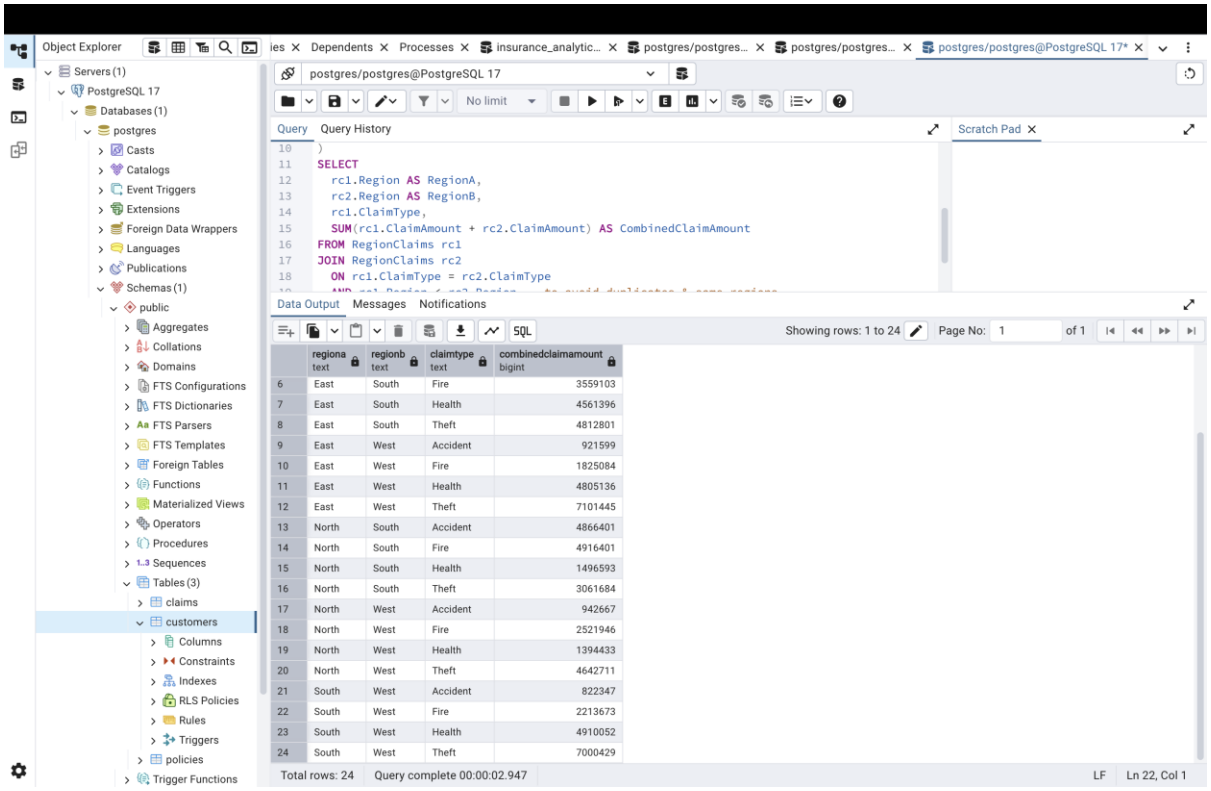


7) Cross Join Trick — Region Combination Claim Analysis

All pairs of different regions and combined total claim amount for same claim type:

```
WITH RegionClaims AS (
  SELECT
    cu.Region,
    cl.ClaimType,
```

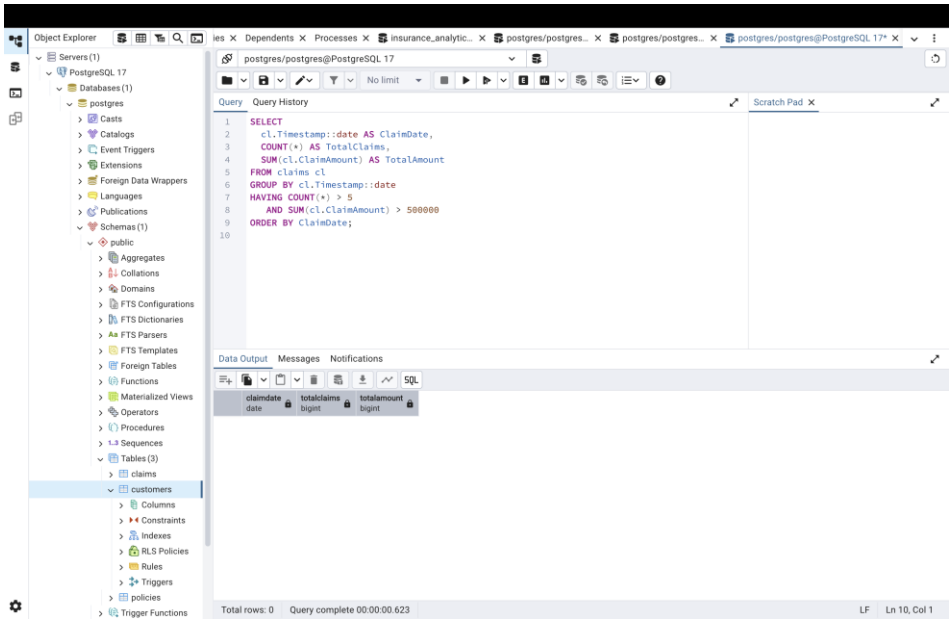
```
    cl.ClaimAmount,
    cl.ClaimID
  FROM claims cl
  JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
  JOIN customers cu ON p.CustomerID = cu.CustomerID
)
SELECT
  rc1.Region AS RegionA,
  rc2.Region AS RegionB,
  rc1.ClaimType,
  SUM(rc1.ClaimAmount + rc2.ClaimAmount) AS CombinedClaimAmount
FROM RegionClaims rc1
JOIN RegionClaims rc2
  ON rc1.ClaimType = rc2.ClaimType
  AND rc1.Region < rc2.Region -- to avoid duplicates & same regions
GROUP BY rc1.Region, rc2.Region, rc1.ClaimType
ORDER BY RegionA, RegionB, ClaimType;
```



8) Claim Clusters by Date

Dates with more than 5 claims and total claim amount > ₹5,00,000:

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

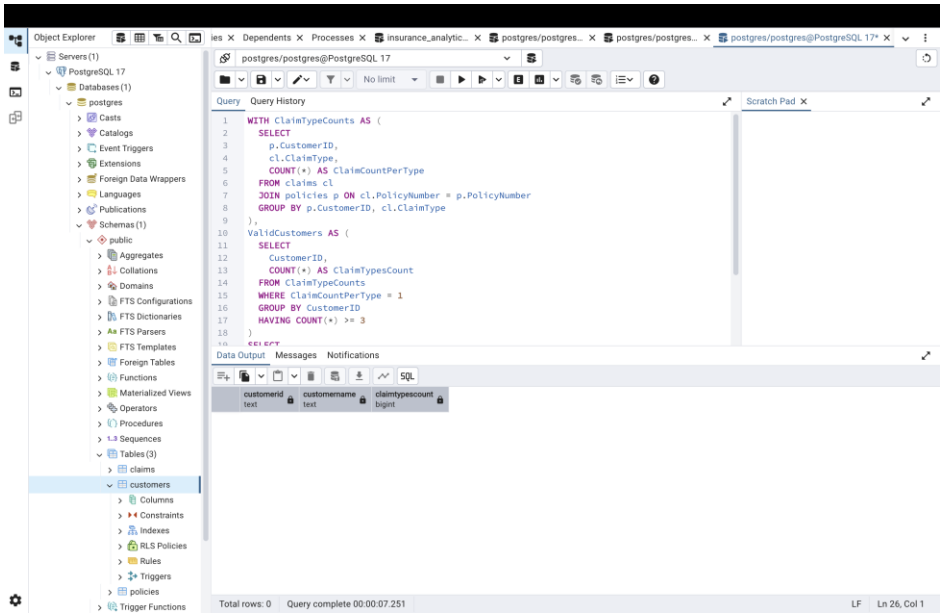


Complex Subquery Puzzle

Customers who never claimed same ClaimType twice but have claims in 3+ different claim types:

```
WITH ClaimTypeCounts AS (
  SELECT
    p.CustomerID,
    cl.ClaimType,
    COUNT(*) AS ClaimCountPerType
  FROM claims cl
  JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
  GROUP BY p.CustomerID, cl.ClaimType
),
ValidCustomers AS (
  SELECT
    CustomerID,
    COUNT(*) AS ClaimTypesCount
```

```
  FROM ClaimTypeCounts
  WHERE ClaimCountPerType = 1
  GROUP BY CustomerID
  HAVING COUNT(*) >= 3
)
SELECT
  vc.CustomerID,
  cu.CustomerName,
  vc.ClaimTypesCount
FROM ValidCustomers vc
JOIN customers cu ON vc.CustomerID = cu.CustomerID
ORDER BY vc.CustomerID;
```



First Normal Form (1NF):

- All tables store atomic values (no repeating groups or arrays).
- Each column holds only one value per row.

Second Normal Form (2NF):

- Every non-key column fully depends on the whole primary key.
- No partial dependencies since primary keys are simple or composite keys are respected.

Third Normal Form (3NF):

- No transitive dependencies exist; non-key columns do not depend on other non-key columns.
- Each attribute depends only on the primary key.

Normalization Issues:

- The datasets appear well normalized.
- If any repeating or derived data appears, separate it into related tables.