

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 PostgreSQL 17 Databases (1) postgres public

```
1 SELECT
2   c.ClaimID,
3   c.PolicyNumber,
4   c.ClaimType,
5   c.Timestamp,
6   CASE
7     WHEN c.Timestamp::date BETWEEN p.StartDate AND p.EndDate THEN 'Within Coverage'
8     ELSE 'Outside Coverage'
9   END AS CoverageStatus
10 FROM claims c
11 JOIN policies p ON c.PolicyNumber = p.PolicyNumber;
12
```

Data Output Messages Notifications

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

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

Successfully run. Total query runtime: 3 secs 502 msec. 100 rows affected

Total rows: 100 Query complete 00:00:03.502

Object Explorer PostgreSQL 17 Databases (1) postgres public

```
1 SELECT
2   c.ClaimID,
3   c.PolicyNumber,
4   c.ClaimType,
5   c.Timestamp,
6   CASE
7     WHEN c.Timestamp::date BETWEEN p.StartDate AND p.EndDate THEN 'Within Coverage'
8     ELSE 'Outside Coverage'
9   END AS CoverageStatus
10 FROM claims c
11 JOIN policies p ON c.PolicyNumber = p.PolicyNumber;
12
```

Data Output Messages Notifications

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

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

Total rows: 100 Query complete 00:00:03.502

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;

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
1 WITH CustomerClaims AS (  
2     SELECT  
3         cu.Region,  
4         cu.CustomerName,  
5         SUM(c.ClaimAmount) AS TotalClaimAmount  
6     FROM claims c  
7     JOIN policies p ON c.PolicyNumber = p.PolicyNumber  
8     JOIN customers cu ON p.CustomerID = cu.CustomerID  
9     GROUP BY cu.Region, cu.CustomerName  
10 )  
11 RankedClaims AS (  
12     SELECT  
13         Region,  
14         CustomerName,  
15         TotalClaimAmount,  
16         RANK() OVER (PARTITION BY Region ORDER BY TotalClaimAmount DESC) AS RankInRegion  
17     FROM CustomerClaims  
18 )  
19 SELECT
```

The query results are displayed in a table with 8 rows and 4 columns: region, customername, totalclaimamount, and rankinregion.

	region	customername	totalclaimamount	rankinregion
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

The interface also shows a sidebar with a tree view of the database structure, including servers, databases, schemas, and tables. The status bar at the bottom indicates "Total rows: 8" and "Query complete 00:00:01.045".

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;

The screenshot shows a PostgreSQL query editor interface. On the left, the Object Explorer displays the database structure, with the 'customers' table selected under the 'public' schema. The main query window contains the following SQL code:

```
1 SELECT
2   p.PolicyNumber,
3   c.CustomerName,
4   p.StartDate,
5   p.EndDate
6 FROM policies p
7 JOIN customers c ON p.CustomerID = c.CustomerID
8 LEFT JOIN claims cl ON p.PolicyNumber = cl.PolicyNumber
9 WHERE CURRENT_DATE BETWEEN p.StartDate AND p.EndDate
10    AND cl.ClaimID IS NULL
11 ORDER BY p.PolicyNumber;
12
```

Below the query window, the Data Output tab shows the schema of the result set:

policynumber	customername	startdate	enddate
text	text	date	date

The status bar at the bottom indicates 'Total rows: 0' and 'Query complete 00:00:01.028'.

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;

The screenshot shows a PostgreSQL IDE interface. On the left is the 'Object Explorer' showing the database structure. The main window displays a SQL query in the 'Query' tab. The query is as follows:

```
16 uc1.ClaimDate AS StartDate,
17 uc2.ClaimDate AS EndDate,
18 COUNT(*) AS TotalUrgentClaims
19 FROM UrgentClaims uc1
20 JOIN UrgentClaims uc2
21 ON uc1.CustomerID = uc2.CustomerID
22 AND uc2.ClaimDate BETWEEN uc1.ClaimDate AND uc1.ClaimDate + INTERVAL '30 days'
23 GROUP BY uc1.CustomerID, uc1.CustomerName, uc1.ClaimDate, uc2.ClaimDate
24 )
25 SELECT DISTINCT
26 CustomerID,
27 CustomerName,
28 TotalUrgentClaims,
29 StartDate AS EarliestClaimDateInWindow,
30 StartDate + INTERVAL '30 days' AS LatestClaimDateInWindow
31 FROM ClaimWindows
32 WHERE TotalUrgentClaims > 2
33 ORDER BY CustomerID;
34
```

Below the query, the 'Data Output' tab shows the results of the query. The results are displayed in a table with the following columns: customerid, customername, totalurgentclaims, earliestclaimdateinwindow, and latestclaimdateinwindow. The table is currently empty, showing 'Total rows: 0'.

customerid	customername	totalurgentclaims	earliestclaimdateinwindow	latestclaimdateinwindow
------------	--------------	-------------------	---------------------------	-------------------------

The status bar at the bottom indicates 'Query complete 00:00:01.066' and 'Ln 26, Col 14'.

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;

The screenshot shows the PostgreSQL 17 IDE interface. On the left, the Object Explorer displays the database structure, including the 'public' schema and the 'customers' table. The main query editor contains the following SQL code:

```

1 WITH ClaimRatios AS (
2     SELECT
3         cl.ClaimID,
4         cl.ClaimType,
5         cl.ClaimAmount,
6         p.PremiumAmount,
7         (cl.ClaimAmount::float / p.PremiumAmount) AS Ratio
8     FROM claims cl
9     JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
10 )
11 SELECT
12     ClaimID,
13     ClaimType,
14     ClaimAmount,
15     PremiumAmount,
16     Ratio,
17     RANK() OVER (PARTITION BY ClaimType ORDER BY Ratio DESC) AS RankInType
18 FROM ClaimRatios
19 ORDER BY ClaimType, RankInType;

```

The Data Output pane shows the results of the query, displaying 11 rows. The columns are: claimid, claimtype, claimamount, premiumamount, ratio, and rankintype. The data is sorted by claimtype and then by rankintype.

claimid	claimtype	claimamount	premiumamount	ratio	rankintype	
1	C1060	Accident	60235	7991	7.5378550869728445	1
2	C1075	Accident	36149	5160	7.00562015503876	2
3	C1045	Accident	95338	15581	6.118862717412233	3
4	C1022	Accident	62056	16527	3.7548254371634298	4
5	C1059	Accident	45647	14075	3.243126110124334	5
6	C1009	Accident	52623	20289	2.593671447582434	6
7	C1013	Accident	76018	38286	1.9855299587316513	7
8	C1072	Accident	59925	33174	1.806384517996021	8
9	C1004	Accident	38606	21596	1.7876458603445082	9
10	C1006	Accident	47439	29337	1.6170365068002863	10
11	C1040	Accident	25634	16518	1.5518827945271825	11

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';
```

The screenshot shows a PostgreSQL IDE interface. On the left is the Object Explorer showing the database structure. The main window displays a SQL query and its results.

Query:

```

11 SELECT DISTINCT ON (p.CustomerID)
12    p.CustomerID,
13    cl.PriorityFlag AS MostRecentClaimPriority
14 FROM claims cl
15 JOIN policies p ON cl.PolicyNumber = p.PolicyNumber
16 ORDER BY p.CustomerID, cl.Timestamp DESC
17 )
18 SELECT
19    a.CustomerID,
20    cu.CustomerName,
21    a.AvgClaimAmount,

```

Data Output:

	customerid text	customername text	avgclaimamount numeric	avgpremiumamount numeric	mostrecentclaimpriority text
1	CU1009	Customer_9	52623.0000000000000000	20289.0000000000000000	NORMAL
2	CU1016	Customer_16	56543.0000000000000000	13570.0000000000000000	NORMAL
3	CU1079	Customer_79	72446.0000000000000000	16711.0000000000000000	NORMAL
4	CU1060	Customer_60	60235.0000000000000000	7991.0000000000000000	NORMAL
5	CU1093	Customer_93	60944.0000000000000000	17238.0000000000000000	NORMAL
6	CU1075	Customer_75	36149.0000000000000000	5160.0000000000000000	NORMAL
7	CU1003	Customer_3	70344.0000000000000000	31762.0000000000000000	NORMAL
8	CU1045	Customer_45	95338.0000000000000000	15581.0000000000000000	NORMAL
9	CU1015	Customer_15	66540.0000000000000000	21279.0000000000000000	NORMAL
10	CU1050	Customer_50	94502.0000000000000000	20589.0000000000000000	NORMAL
11	CU1055	Customer_55	96334.0000000000000000	9239.0000000000000000	NORMAL
12	CU1084	Customer_84	82358.0000000000000000	38410.0000000000000000	NORMAL
13	CU1037	Customer_37	38249.0000000000000000	19027.0000000000000000	NORMAL
14	CU1076	Customer_76	74265.0000000000000000	10947.0000000000000000	NORMAL
15	CU1049	Customer_49	82980.0000000000000000	7347.0000000000000000	NORMAL

Total rows: 15 Query complete 00:00:04.560 LF Ln 29, Col 1

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;
```

The screenshot shows the PostgreSQL Enterprise Manager interface. On the left, the Object Explorer displays the database structure, including the 'public' schema and the 'customers' table. The main pane shows a SQL query that joins 'RegionClaims' and 'RegionClaims' tables to calculate a combined claim amount. The query is as follows:

```

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

```

The Data Output pane shows the results of the query, displaying 24 rows. The columns are 'regiona', 'regionb', 'claimtype', and 'combinedclaimamount'. The results are as follows:

regiona	regionb	claimtype	combinedclaimamount
East	South	Fire	3559103
East	South	Health	4561396
East	South	Theft	4812801
East	West	Accident	921599
East	West	Fire	1825084
East	West	Health	4805136
East	West	Theft	7101445
North	South	Accident	4866401
North	South	Fire	4916401
North	South	Health	1496593
North	South	Theft	3061684
North	West	Accident	942667
North	West	Fire	2521946
North	West	Health	1394433
North	West	Theft	4642711
South	West	Accident	822347
South	West	Fire	2213673
South	West	Health	4910052
South	West	Theft	7000429

The status bar at the bottom indicates 'Total rows: 24' and 'Query complete 00:00:02.947'.

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

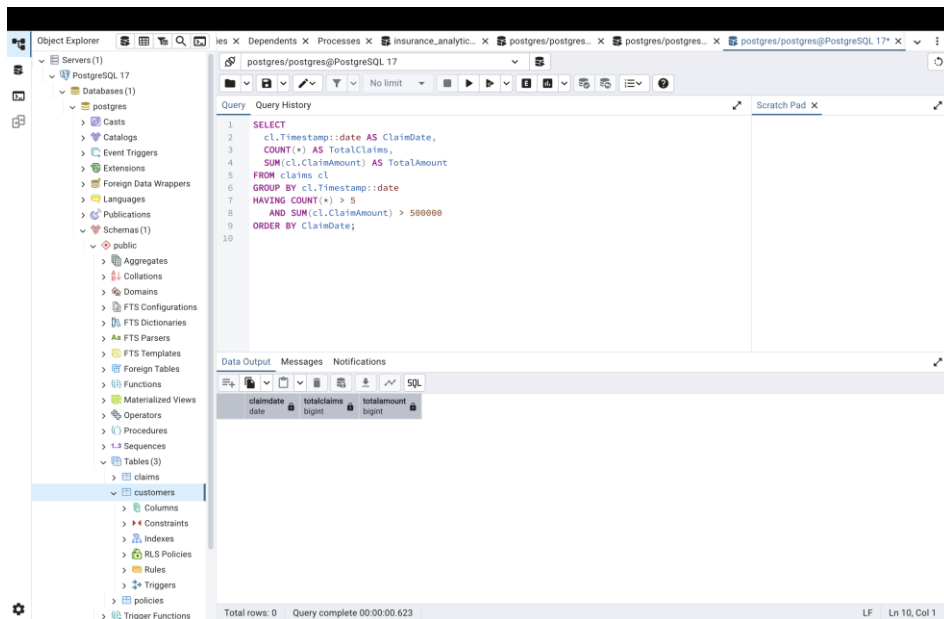
FROM claims cl

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,

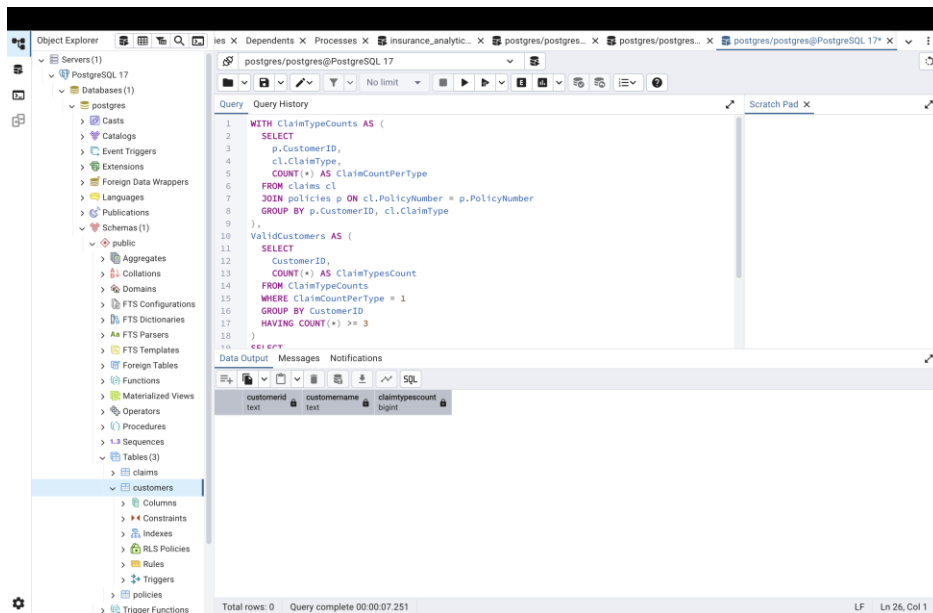
    cu.CustomerName,

    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.