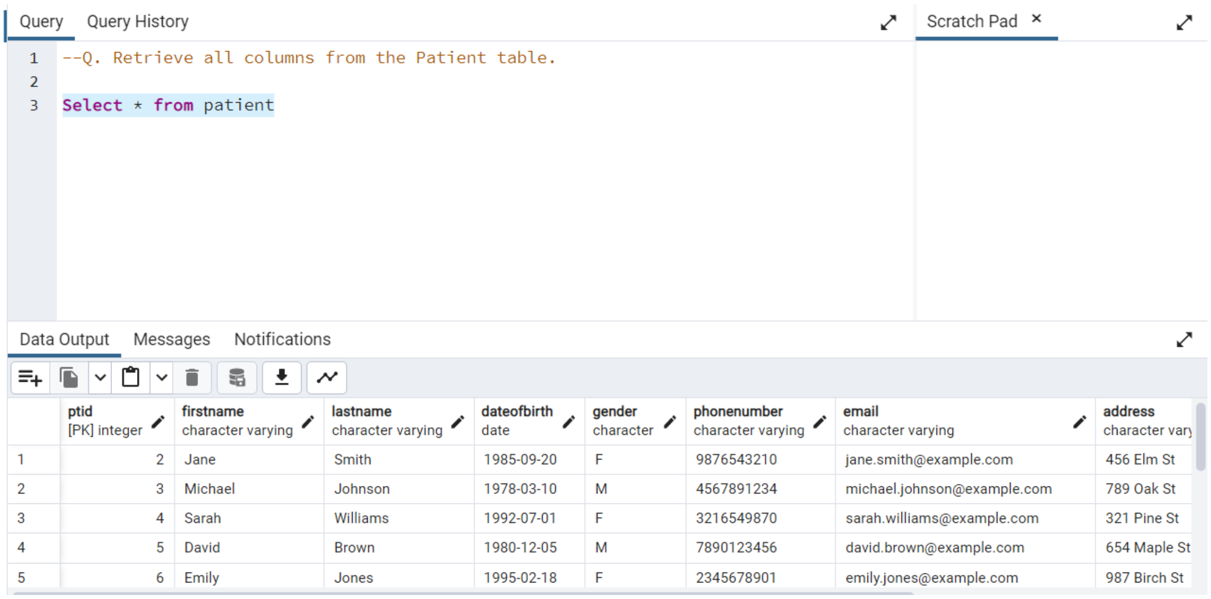
SQL Queries & Solutions for US Healthcare Data Analysis

- Basic SQL Queries: SELECT queries to retrieve data from the tables.

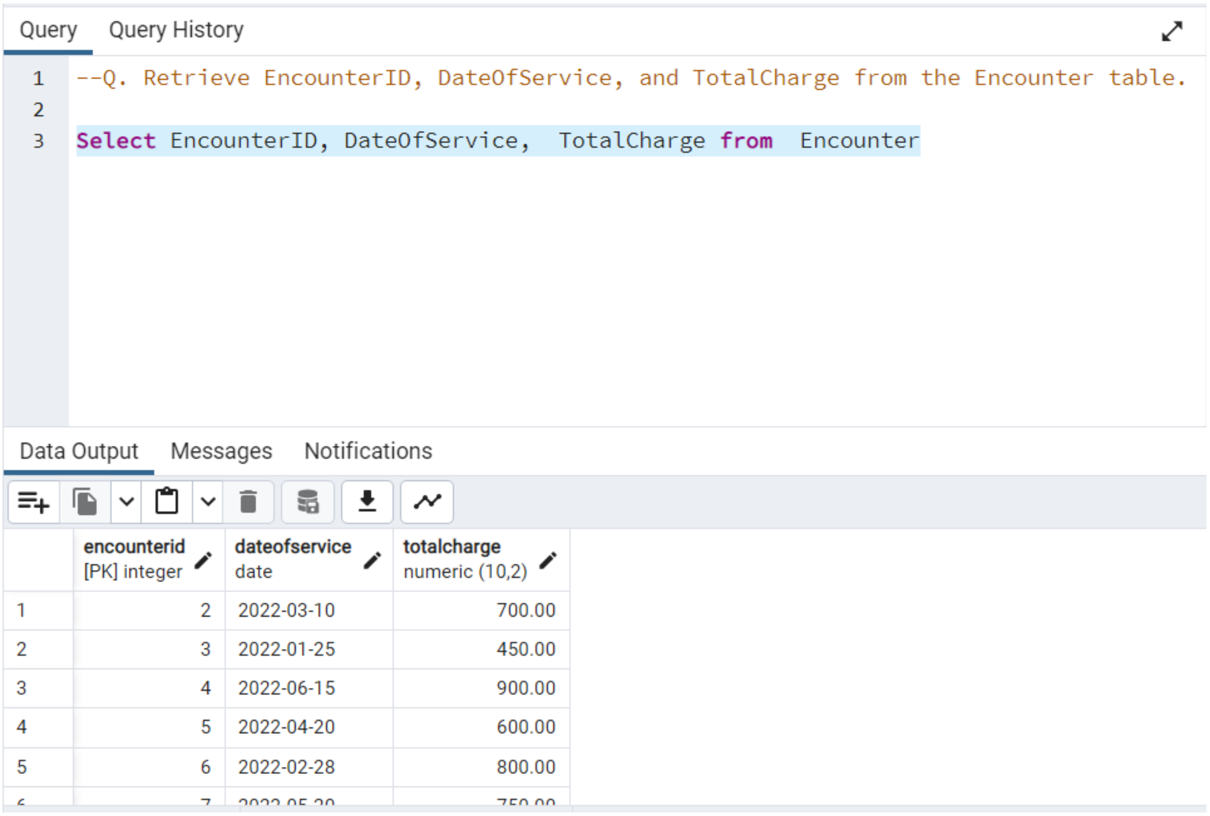
Q. Retrieve all columns from the Patient table.



```
--Q. Retrieve all columns from the Patient table.

Select * from patient
```

	ptid [PK] integer	firstname character varying	lastname character varying	dateofbirth date	gender character	phonenumber character varying	email character varying	address character vary
1	2	Jane	Smith	1985-09-20	F	9876543210	jane.smith@example.com	456 Elm St
2	3	Michael	Johnson	1978-03-10	M	4567891234	michael.johnson@example.com	789 Oak St
3	4	Sarah	Williams	1992-07-01	F	3216549870	sarah.williams@example.com	321 Pine St
4	5	David	Brown	1980-12-05	M	7890123456	david.brown@example.com	654 Maple St
5	6	Emily	Jones	1995-02-18	F	2345678901	emily.jones@example.com	987 Birch St

Q. Retrieve EncounterID, DateOfService, and TotalCharge from the Encounter table.

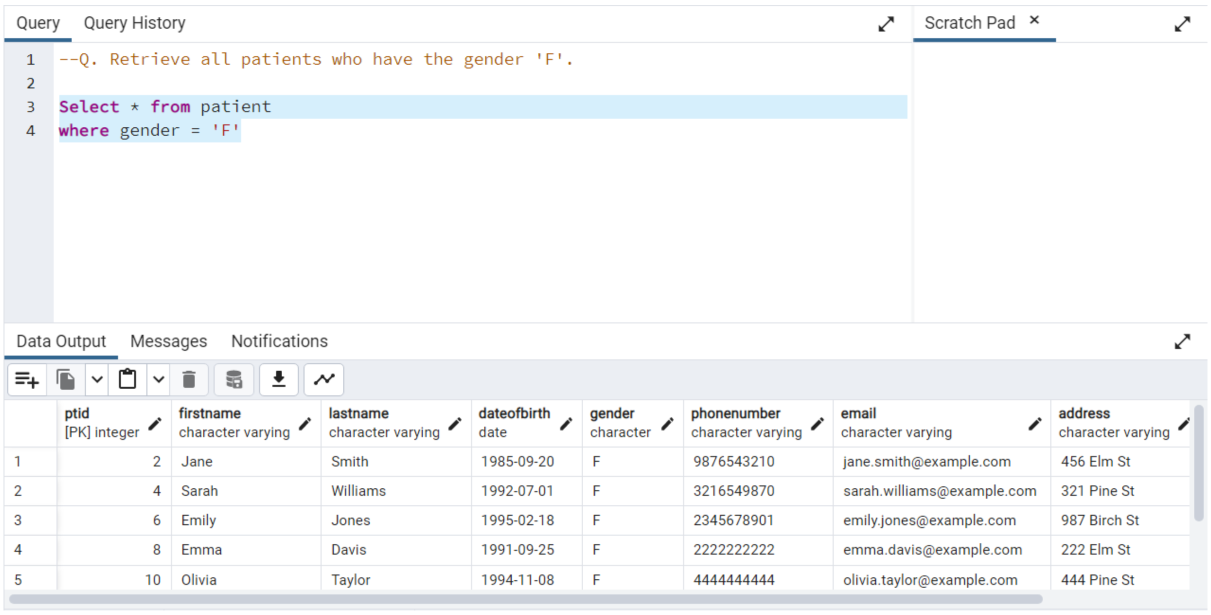


```
--Q. Retrieve EncounterID, DateOfService, and TotalCharge from the Encounter table.

Select EncounterID, DateOfService,  TotalCharge from  Encounter
```

	encounterid [PK] integer	dateofservice date	totalcharge numeric (10,2)
1	2	2022-03-10	700.00
2	3	2022-01-25	450.00
3	4	2022-06-15	900.00
4	5	2022-04-20	600.00
5	6	2022-02-28	800.00

- Filtering Data: Practice using the WHERE clause to filter data based on specific conditions.

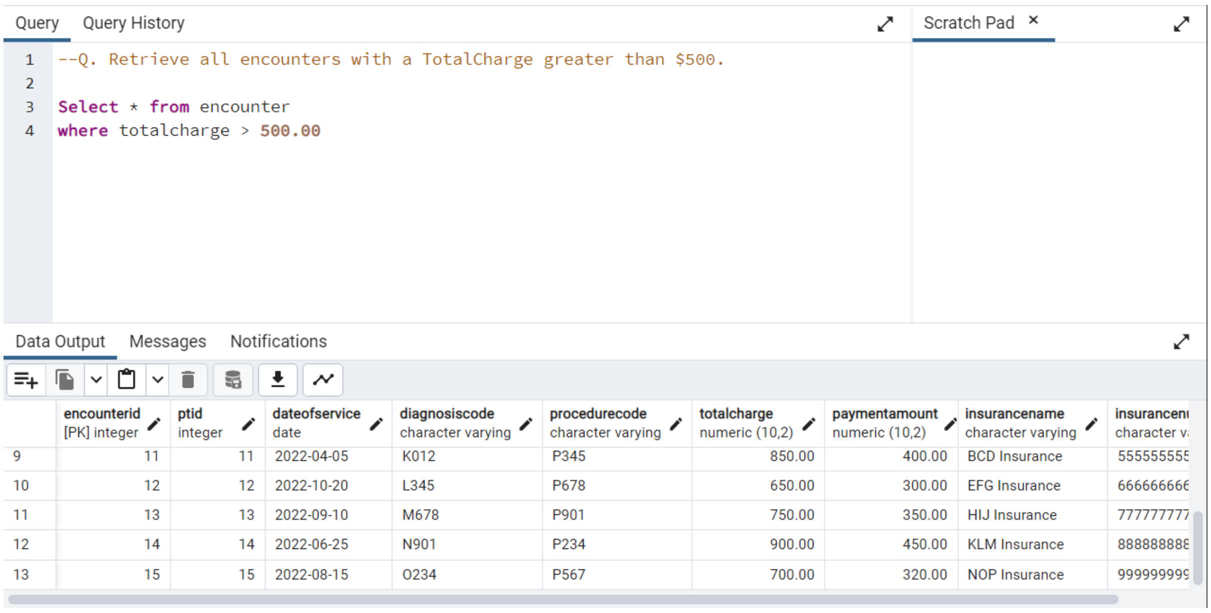
Q. Retrieve all patients who have the gender 'F'.



```sql
--Q. Retrieve all patients who have the gender 'F'.

Select * from patient
where gender = 'F'
```

	ptid [PK] integer	firstname character varying	lastname character varying	dateofbirth date	gender character	phonenumber character varying	email character varying	address character varying
1	2	Jane	Smith	1985-09-20	F	9876543210	jane.smith@example.com	456 Elm St
2	4	Sarah	Williams	1992-07-01	F	3216549870	sarah.williams@example.com	321 Pine St
3	6	Emily	Jones	1995-02-18	F	2345678901	emily.jones@example.com	987 Birch St
4	8	Emma	Davis	1991-09-25	F	2222222222	emma.davis@example.com	222 Elm St
5	10	Olivia	Taylor	1994-11-08	F	4444444444	olivia.taylor@example.com	444 Pine St

Q. Retrieve all encounters with a TotalCharge greater than $500.

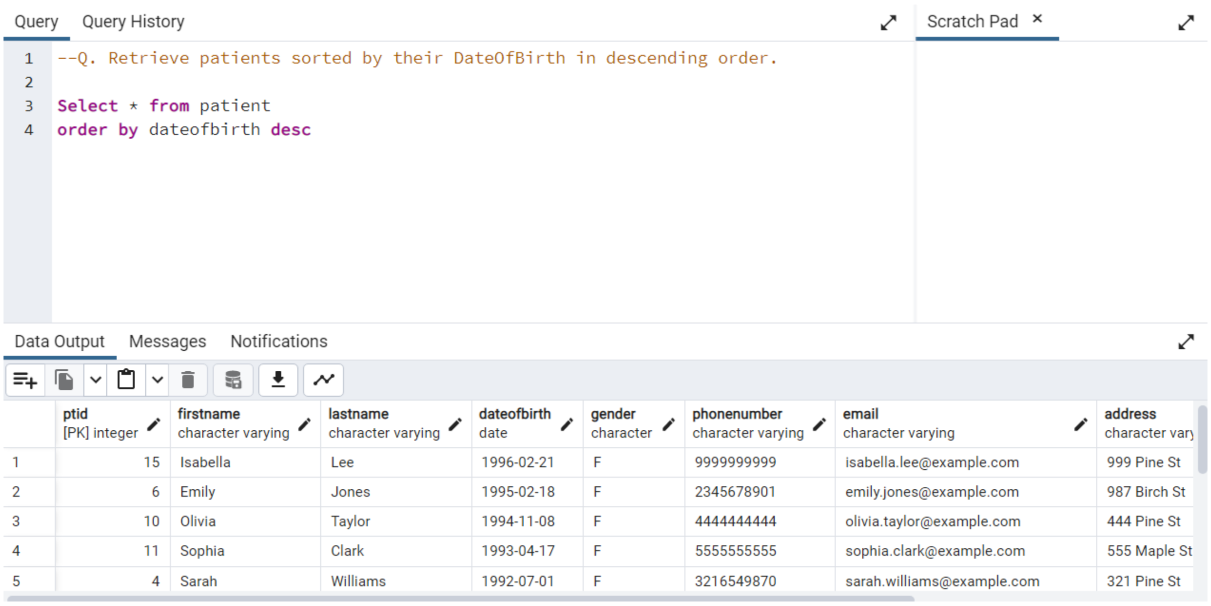


```sql
--Q. Retrieve all encounters with a TotalCharge greater than $500.

Select * from encounter
where totalcharge > 500.00
```

	encounterid [PK] integer	ptid integer	dateofservice date	diagnosiscode character varying	procedurecode character varying	totalcharge numeric (10,2)	paymentamount numeric (10,2)	insurancename character varying	insurancenu character va
9	11	11	2022-04-05	K012	P345	850.00	400.00	BCD Insurance	555555555
10	12	12	2022-10-20	L345	P678	650.00	300.00	EFG Insurance	666666666
11	13	13	2022-09-10	M678	P901	750.00	350.00	HIJ Insurance	777777777
12	14	14	2022-06-25	N901	P234	900.00	450.00	KLM Insurance	888888888
13	15	15	2022-08-15	O234	P567	700.00	320.00	NOP Insurance	999999999

- Sorting and Ordering: Use the ORDER BY clause to sort the data in ascending or descending order.

Q. Retrieve patients sorted by their DateOfBirth in descending order.



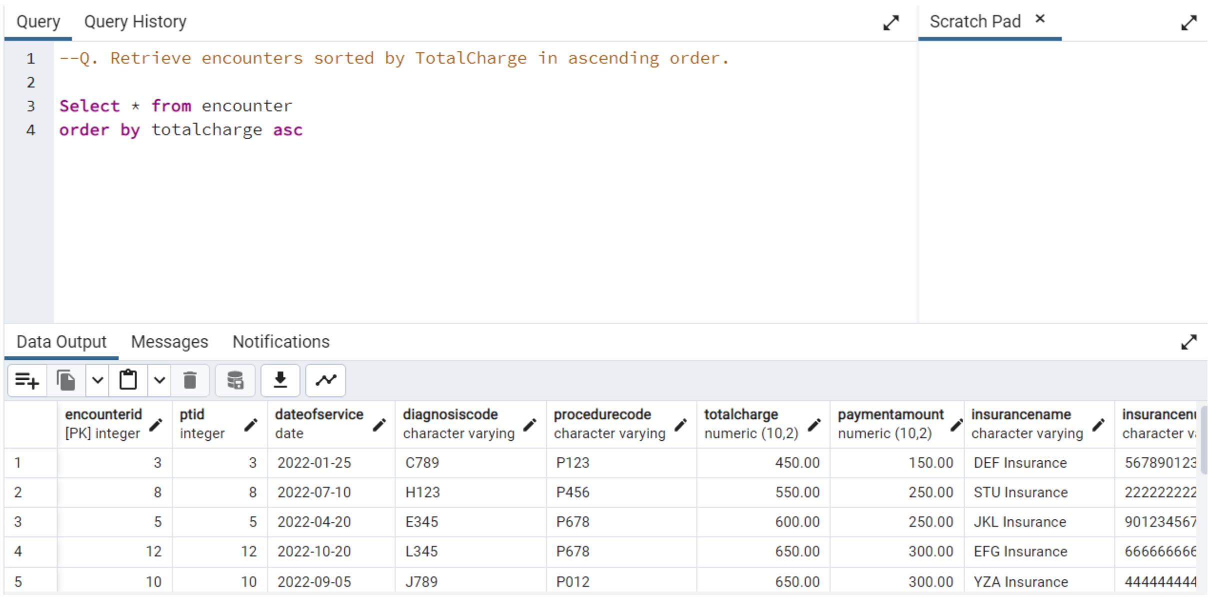
Query | Query History | Scratch Pad

```sql
--Q. Retrieve patients sorted by their DateOfBirth in descending order.

Select * from patient
order by dateofbirth desc
```

Data Output | Messages | Notifications

	ptid [PK] integer	firstname character varying	lastname character varying	dateofbirth date	gender character	phonenumber character varying	email character varying	address character vary
1	15	Isabella	Lee	1996-02-21	F	9999999999	isabella.lee@example.com	999 Pine St
2	6	Emily	Jones	1995-02-18	F	2345678901	emily.jones@example.com	987 Birch St
3	10	Olivia	Taylor	1994-11-08	F	4444444444	olivia.taylor@example.com	444 Pine St
4	11	Sophia	Clark	1993-04-17	F	5555555555	sophia.clark@example.com	555 Maple St
5	4	Sarah	Williams	1992-07-01	F	3216549870	sarah.williams@example.com	321 Pine St

Q. Retrieve encounters sorted by TotalCharge in ascending order.



Query | Query History | Scratch Pad

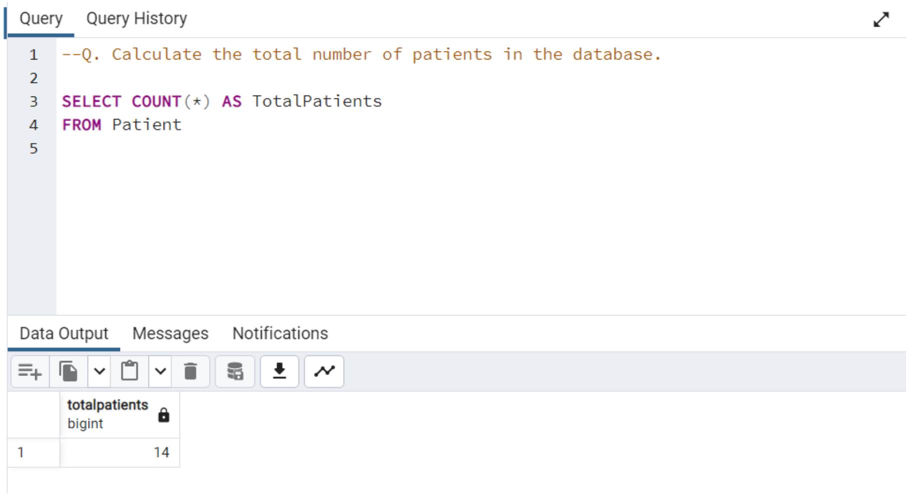
```sql
--Q. Retrieve encounters sorted by TotalCharge in ascending order.

Select * from encounter
order by totalcharge asc
```

Data Output | Messages | Notifications

	encounterid [PK] integer	ptid integer	dateofservice date	diagnosiscode character varying	procedurecode character varying	totalcharge numeric (10,2)	paymentamount numeric (10,2)	insurancename character varying	insurancenu character va
1	3	3	2022-01-25	C789	P123	450.00	150.00	DEF Insurance	567890123
2	8	8	2022-07-10	H123	P456	550.00	250.00	STU Insurance	222222222
3	5	5	2022-04-20	E345	P678	600.00	250.00	JKL Insurance	901234567
4	12	12	2022-10-20	L345	P678	650.00	300.00	EFG Insurance	666666666
5	10	10	2022-09-05	J789	P012	650.00	300.00	YZA Insurance	444444444

- Aggregation: Use of aggregate functions like COUNT, SUM, AVG, MIN, and MAX.

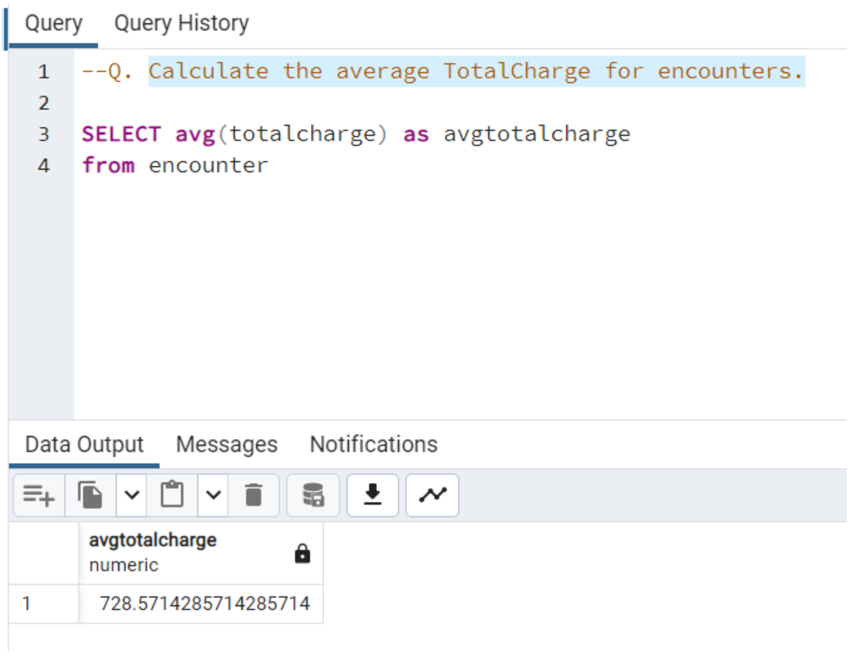
Q. Calculate the total number of patients in the database.



```
--Q. Calculate the total number of patients in the database.

SELECT COUNT(*) AS TotalPatients
FROM Patient

```

```
  totalpatients
  bigint
1            14
```

Q. Calculate the average TotalCharge for encounters.

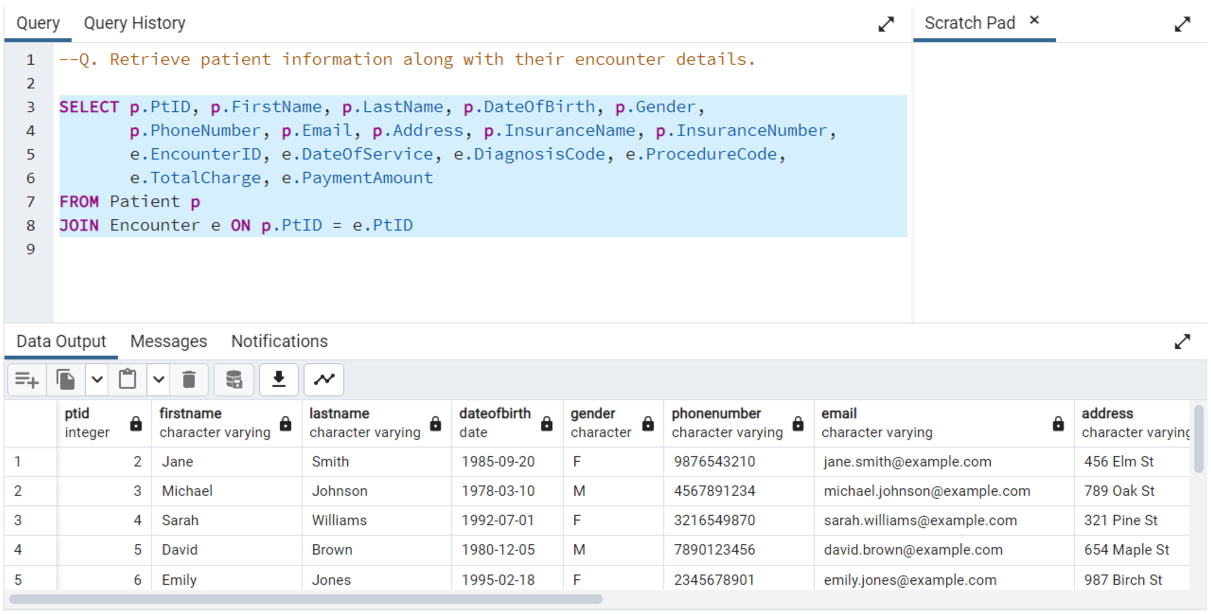


```
--Q. Calculate the average TotalCharge for encounters.

SELECT avg(totalcharge) as avgtotalcharge
from encounter
```

```
  avgtotalcharge
  numeric
1 728.5714285714285714
```

- **Joins**: Utilize JOIN clauses to combine data from multiple tables.

Q. Retrieve patient information along with their encounter details.



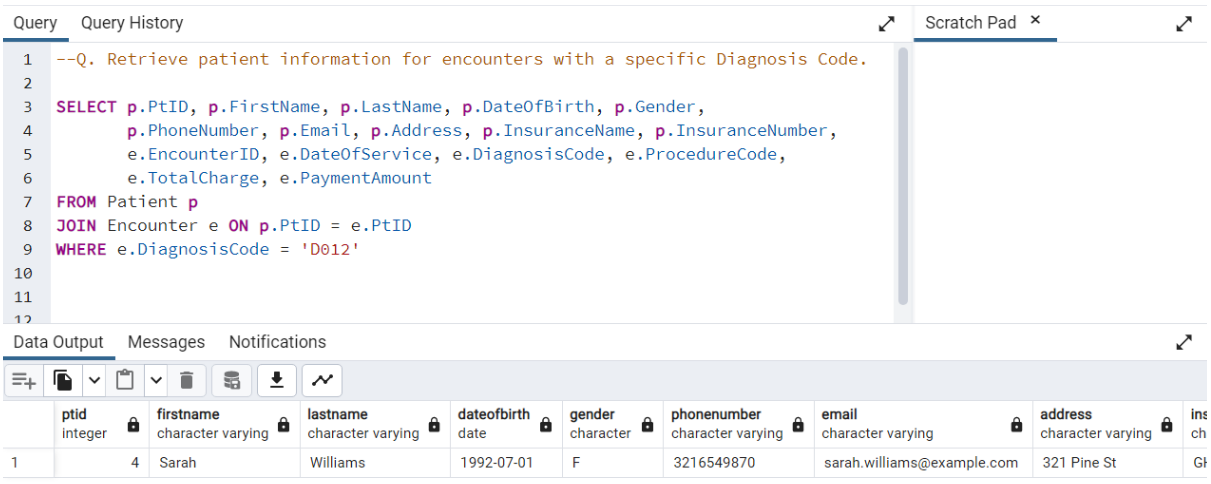
Query | Query History | Scratch Pad

```
--Q. Retrieve patient information along with their encounter details.

SELECT p.PtID, p.FirstName, p.LastName, p.DateOfBirth, p.Gender,
       p.PhoneNumber, p.Email, p.Address, p.InsuranceName, p.InsuranceNumber,
       e.EncounterID, e.DateOfService, e.DiagnosisCode, e.ProcedureCode,
       e.TotalCharge, e.PaymentAmount
FROM Patient p
JOIN Encounter e ON p.PtID = e.PtID
```

Data Output | Messages | Notifications

	ptid integer	firstname character varying	lastname character varying	dateofbirth date	gender character	phonenumber character varying	email character varying	address character varying
1	2	Jane	Smith	1985-09-20	F	9876543210	jane.smith@example.com	456 Elm St
2	3	Michael	Johnson	1978-03-10	M	4567891234	michael.johnson@example.com	789 Oak St
3	4	Sarah	Williams	1992-07-01	F	3216549870	sarah.williams@example.com	321 Pine St
4	5	David	Brown	1980-12-05	M	7890123456	david.brown@example.com	654 Maple St
5	6	Emily	Jones	1995-02-18	F	2345678901	emily.jones@example.com	987 Birch St

Q. Retrieve patient information for encounters with a specific Diagnosis Code.



Query | Query History | Scratch Pad

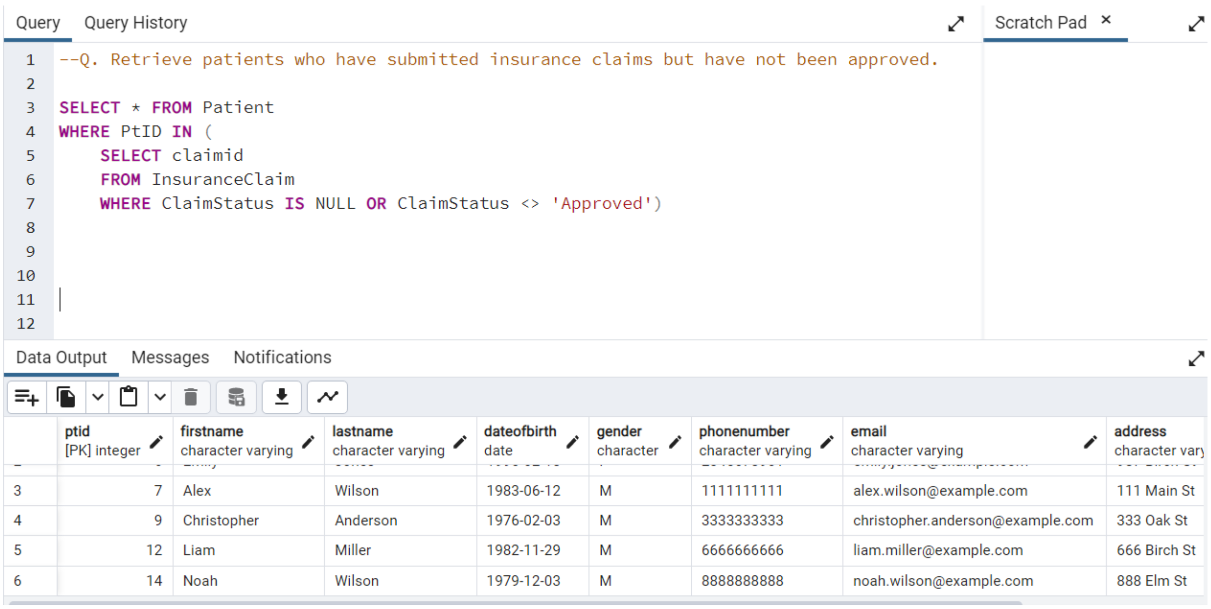
```
--Q. Retrieve patient information for encounters with a specific Diagnosis Code.

SELECT p.PtID, p.FirstName, p.LastName, p.DateOfBirth, p.Gender,
       p.PhoneNumber, p.Email, p.Address, p.InsuranceName, p.InsuranceNumber,
       e.EncounterID, e.DateOfService, e.DiagnosisCode, e.ProcedureCode,
       e.TotalCharge, e.PaymentAmount
FROM Patient p
JOIN Encounter e ON p.PtID = e.PtID
WHERE e.DiagnosisCode = 'D012'
```

Data Output | Messages | Notifications

	ptid integer	firstname character varying	lastname character varying	dateofbirth date	gender character	phonenumber character varying	email character varying	address character varying	ins ch
1	4	Sarah	Williams	1992-07-01	F	3216549870	sarah.williams@example.com	321 Pine St	GI

- **Subqueries**: Use subqueries to nest queries within queries.

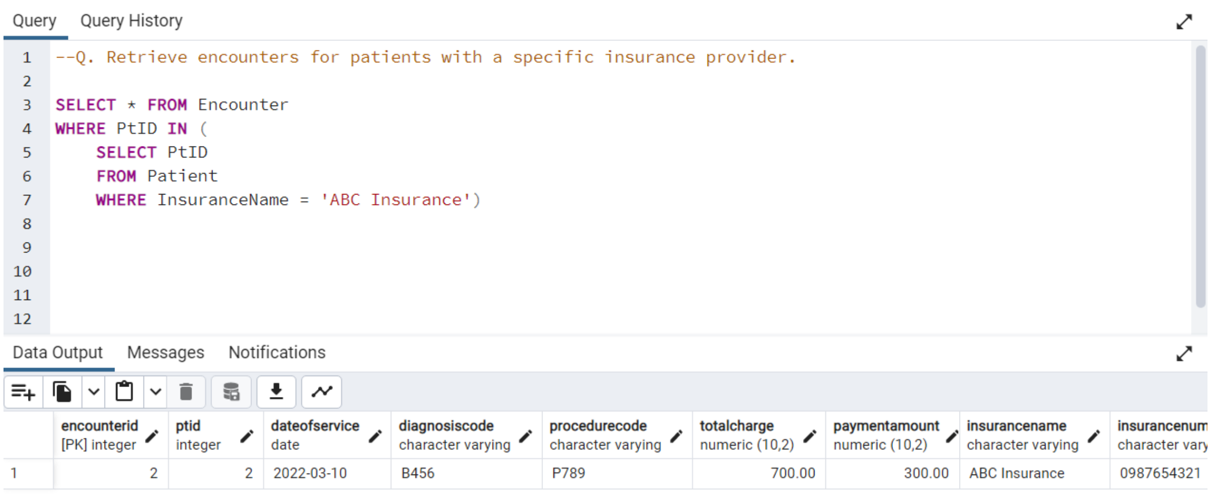
Q. Retrieve patients who have submitted insurance claims but have not been approved.



```sql
--Q. Retrieve patients who have submitted insurance claims but have not been approved.

SELECT * FROM Patient
WHERE PtID IN (
    SELECT claimid
    FROM InsuranceClaim
    WHERE ClaimStatus IS NULL OR ClaimStatus <> 'Approved')
```

	ptid [PK] integer	firstname character varying	lastname character varying	dateofbirth date	gender character	phonenumber character varying	email character varying	address character vary
3	7	Alex	Wilson	1983-06-12	M	1111111111	alex.wilson@example.com	111 Main St
4	9	Christopher	Anderson	1976-02-03	M	3333333333	christopher.anderson@example.com	333 Oak St
5	12	Liam	Miller	1982-11-29	M	6666666666	liam.miller@example.com	666 Birch St
6	14	Noah	Wilson	1979-12-03	M	8888888888	noah.wilson@example.com	888 Elm St

Q. Retrieve encounters for patients with a specific insurance provider.

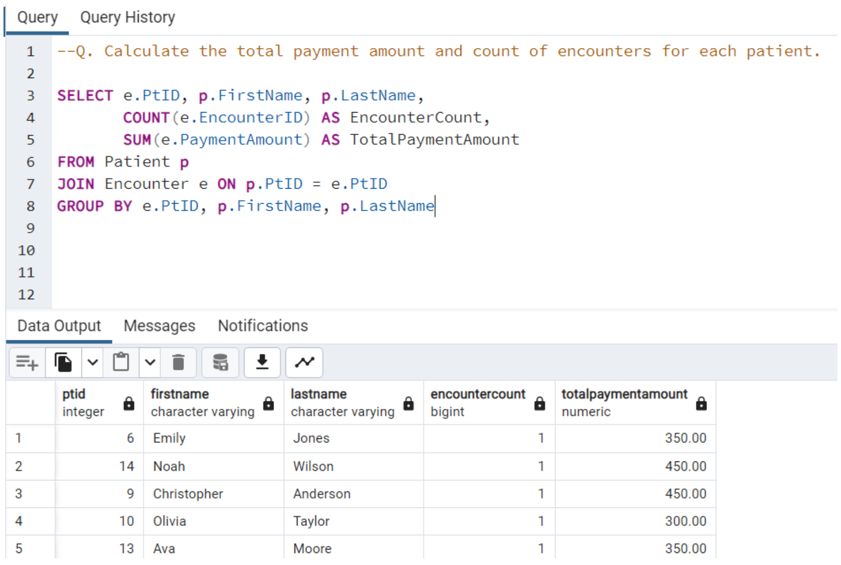


```sql
--Q. Retrieve encounters for patients with a specific insurance provider.

SELECT * FROM Encounter
WHERE PtID IN (
    SELECT PtID
    FROM Patient
    WHERE InsuranceName = 'ABC Insurance')
```

	encounterid [PK] integer	ptid integer	dateofservice date	diagnosiscode character varying	procedurecode character varying	totalcharge numeric (10,2)	paymentamount numeric (10,2)	insurancename character varying	insurancenum character vary
1	2	2	2022-03-10	B456	P789	700.00	300.00	ABC Insurance	0987654321

- **Grouping and Group Functions**: GROUP BY along with aggregate functions.

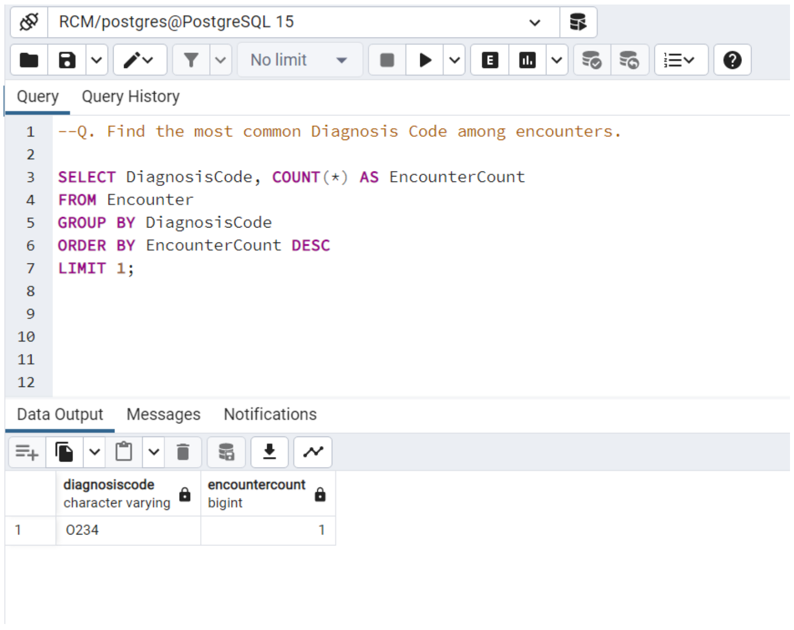
Q. Calculate the total payment amount and count of encounters for each patient.



```
--Q. Calculate the total payment amount and count of encounters for each patient.

SELECT e.PtID, p.FirstName, p.LastName,
       COUNT(e.EncounterID) AS EncounterCount,
       SUM(e.PaymentAmount) AS TotalPaymentAmount
FROM Patient p
JOIN Encounter e ON p.PtID = e.PtID
GROUP BY e.PtID, p.FirstName, p.LastName
```

	ptid integer	firstname character varying	lastname character varying	encountercount bigint	totalpaymentamount numeric
1	6	Emily	Jones	1	350.00
2	14	Noah	Wilson	1	450.00
3	9	Christopher	Anderson	1	450.00
4	10	Olivia	Taylor	1	300.00
5	13	Ava	Moore	1	350.00

Q. Find the most common Diagnosis Code among encounters.



```
--Q. Find the most common Diagnosis Code among encounters.

SELECT DiagnosisCode, COUNT(*) AS EncounterCount
FROM Encounter
GROUP BY DiagnosisCode
ORDER BY EncounterCount DESC
LIMIT 1;
```

	diagnosiscode character varying	encountercount bigint
1	O234	1

- **Cases**: Conditional Logic with CASE Statements in SQL

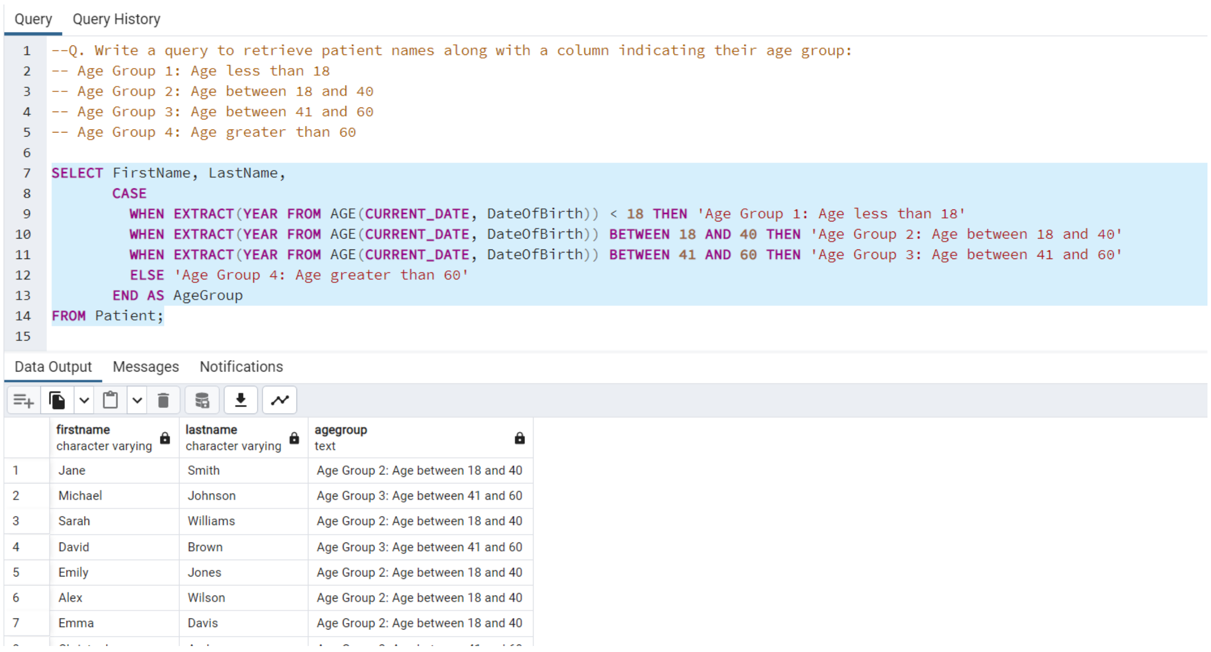
Q. Write a query to retrieve patient names along with a column indicating their age group:

Age Group 1: Age less than 18

Age Group 2: Age between 18 and 40

Age Group 3: Age between 41 and 60

Age Group 4: Age greater than 60

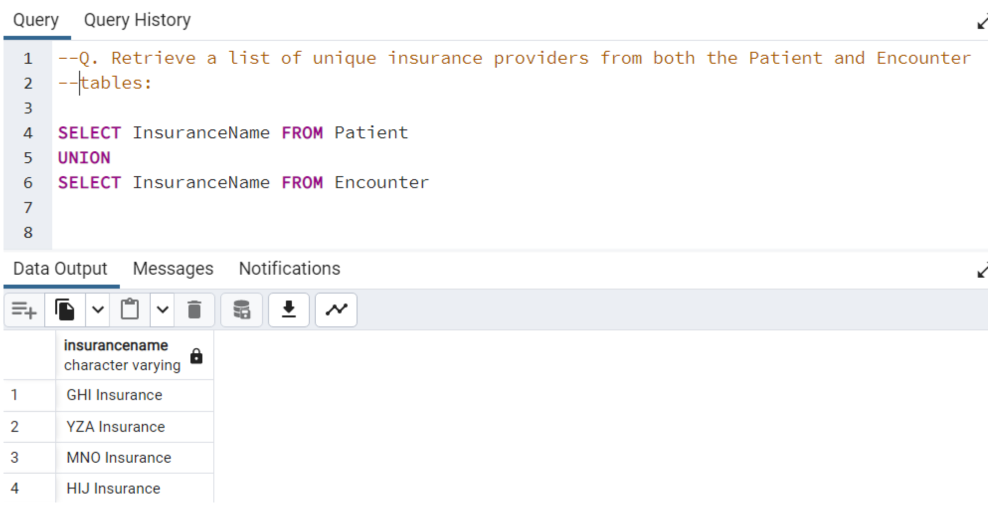


```sql
--Q. Write a query to retrieve patient names along with a column indicating their age group:
-- Age Group 1: Age less than 18
-- Age Group 2: Age between 18 and 40
-- Age Group 3: Age between 41 and 60
-- Age Group 4: Age greater than 60

SELECT FirstName, LastName,
       CASE
         WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, DateOfBirth)) < 18 THEN 'Age Group 1: Age less than 18'
         WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, DateOfBirth)) BETWEEN 18 AND 40 THEN 'Age Group 2: Age between 18 and 40'
         WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, DateOfBirth)) BETWEEN 41 AND 60 THEN 'Age Group 3: Age between 41 and 60'
         ELSE 'Age Group 4: Age greater than 60'
       END AS AgeGroup
FROM Patient;
```

	firstname character varying	lastname character varying	agegroup text
1	Jane	Smith	Age Group 2: Age between 18 and 40
2	Michael	Johnson	Age Group 3: Age between 41 and 60
3	Sarah	Williams	Age Group 2: Age between 18 and 40
4	David	Brown	Age Group 3: Age between 41 and 60
5	Emily	Jones	Age Group 2: Age between 18 and 40
6	Alex	Wilson	Age Group 2: Age between 18 and 40
7	Emma	Davis	Age Group 2: Age between 18 and 40

- **Union and Union All**: Combining Query Results with UNION and UNION ALL in SQL

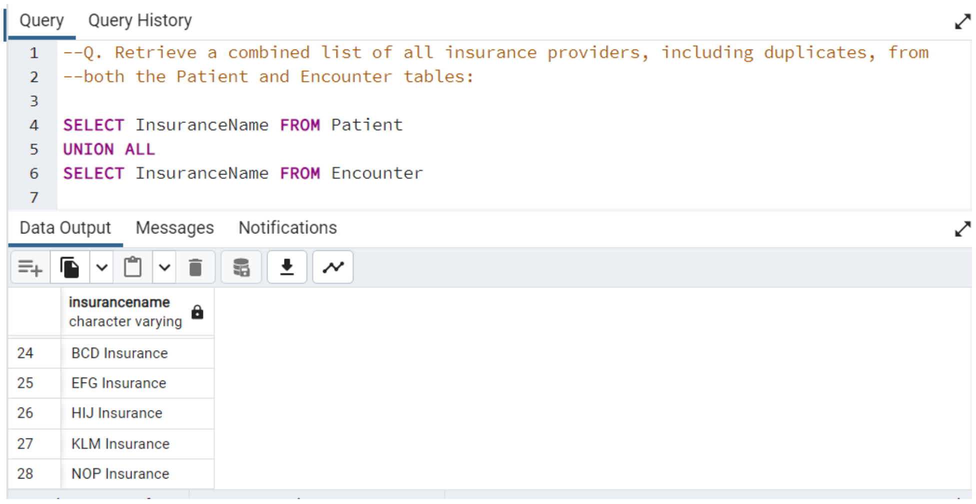
Q. Retrieve a list of unique insurance providers from both the Patient and Encounter tables:



```sql
--Q. Retrieve a list of unique insurance providers from both the Patient and Encounter
--tables:

SELECT InsuranceName FROM Patient
UNION
SELECT InsuranceName FROM Encounter
```

	insurancename character varying
1	GHI Insurance
2	YZA Insurance
3	MNO Insurance
4	HIJ Insurance

Q. Retrieve a combined list of all insurance providers, including duplicates, from both the Patient and Encounter tables:

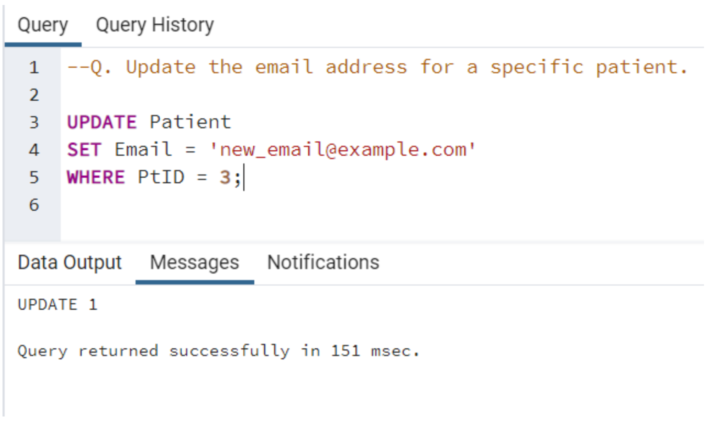


```
--Q. Retrieve a combined list of all insurance providers, including duplicates, from
--both the Patient and Encounter tables:

SELECT InsuranceName FROM Patient
UNION ALL
SELECT InsuranceName FROM Encounter
```

```
    insurancename
    character varying
24  BCD Insurance
25  EFG Insurance
26  HIJ Insurance
27  KLM Insurance
28  NOP Insurance
```

- **Updating and Deleting Data**: UPDATE and DELETE statements.

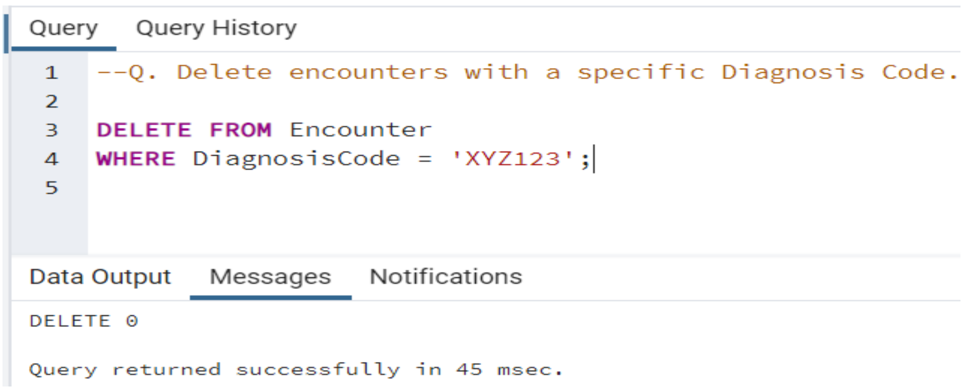
Q. Update the email address for a specific patient.



```
--Q. Update the email address for a specific patient.

UPDATE Patient
SET Email = 'new_email@example.com'
WHERE PtID = 3;
```

```
UPDATE 1

Query returned successfully in 151 msec.
```

Q. Delete encounters with a specific Diagnosis Code.

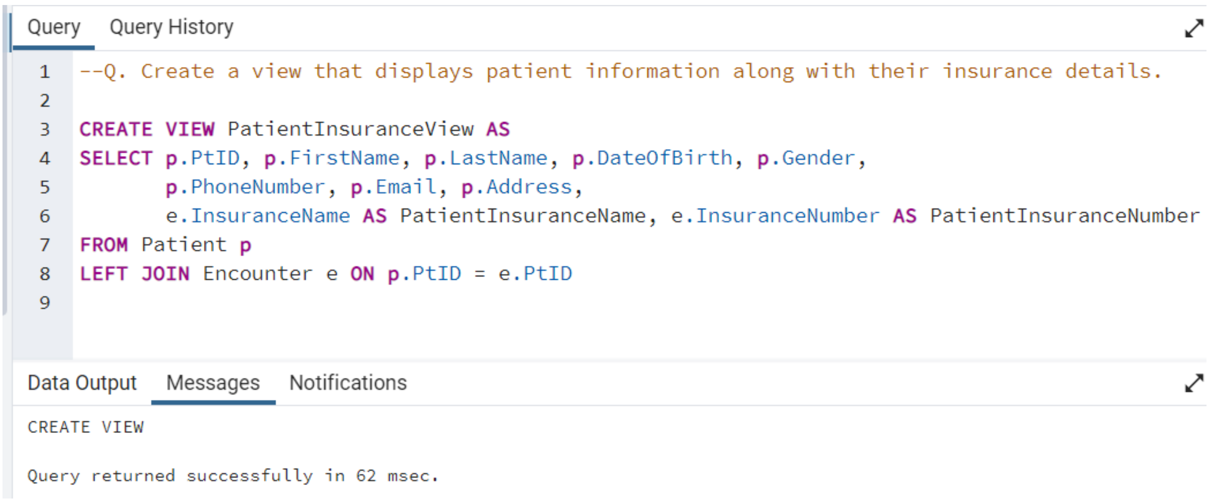


```
--Q. Delete encounters with a specific Diagnosis Code.

DELETE FROM Encounter
WHERE DiagnosisCode = 'XYZ123';
```

```
DELETE 0

Query returned successfully in 45 msec.
```

- **Views**: Create views to simplify complex queries or provide a more structured view of the data.

Q. Create a view that displays patient information along with their insurance details.

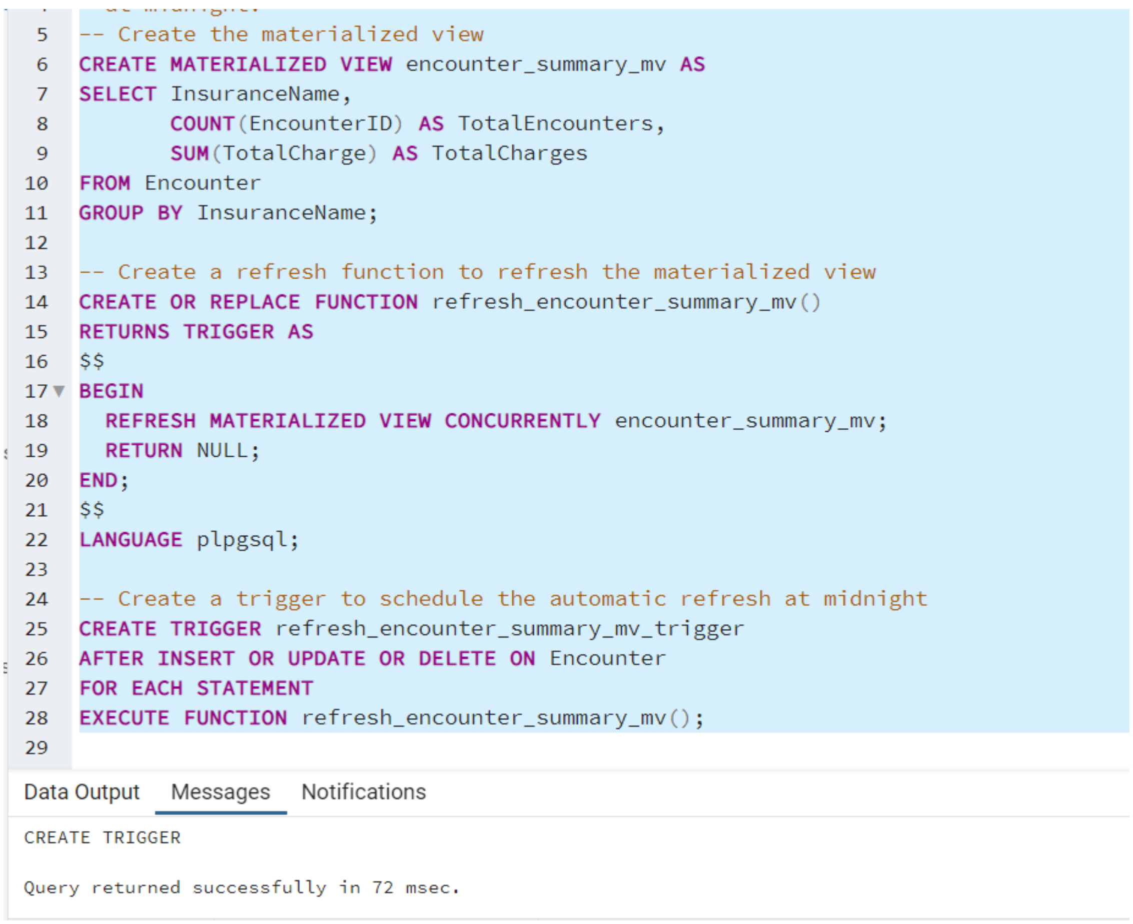


```sql
--Q. Create a view that displays patient information along with their insurance details.

CREATE VIEW PatientInsuranceView AS
SELECT p.PtID, p.FirstName, p.LastName, p.DateOfBirth, p.Gender,
       p.PhoneNumber, p.Email, p.Address,
       e.InsuranceName AS PatientInsuranceName, e.InsuranceNumber AS PatientInsuranceNumber
FROM Patient p
LEFT JOIN Encounter e ON p.PtID = e.PtID
```

```
CREATE VIEW

Query returned successfully in 62 msec.
```

- **Materialized view**: Accelerate Query Performance with Materialized Views

Q. Designed a materialized view named encounter_summary_mv that calculates and stores the total number of encounters and the total charges incurred for each insurance provider in the Encounter table. The materialized view should be refreshed automatically every day at midnight.

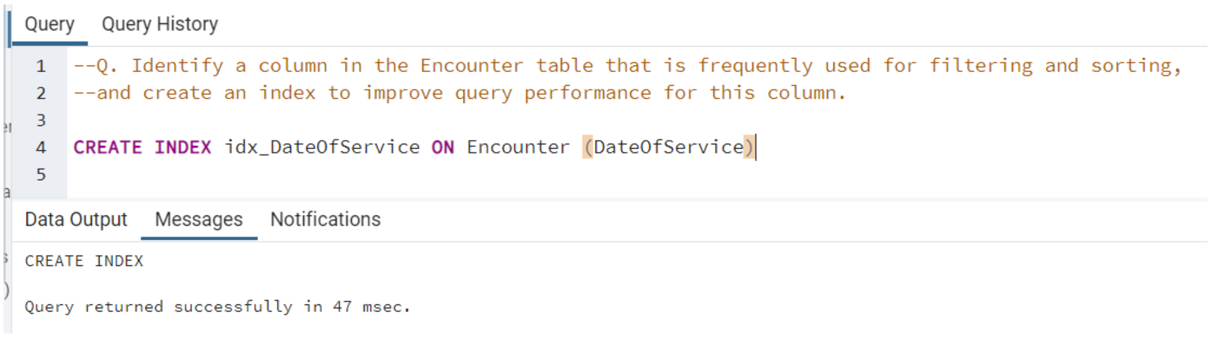


```sql
at midnight.
-- Create the materialized view
CREATE MATERIALIZED VIEW encounter_summary_mv AS
SELECT InsuranceName,
       COUNT(EncounterID) AS TotalEncounters,
       SUM(TotalCharge) AS TotalCharges
FROM Encounter
GROUP BY InsuranceName;

-- Create a refresh function to refresh the materialized view
CREATE OR REPLACE FUNCTION refresh_encounter_summary_mv()
RETURNS TRIGGER AS
$$
BEGIN
  REFRESH MATERIALIZED VIEW CONCURRENTLY encounter_summary_mv;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;

-- Create a trigger to schedule the automatic refresh at midnight
CREATE TRIGGER refresh_encounter_summary_mv_trigger
AFTER INSERT OR UPDATE OR DELETE ON Encounter
FOR EACH STATEMENT
EXECUTE FUNCTION refresh_encounter_summary_mv();
```

```
CREATE TRIGGER

Query returned successfully in 72 msec.
```

- Indexes and Performance: Enhancing Query Performance with Indexes in SQL

Q. Identify a column in the Encounter table that is frequently used for filtering and sorting, and create an index to improve query performance for this column.



Query | Query History

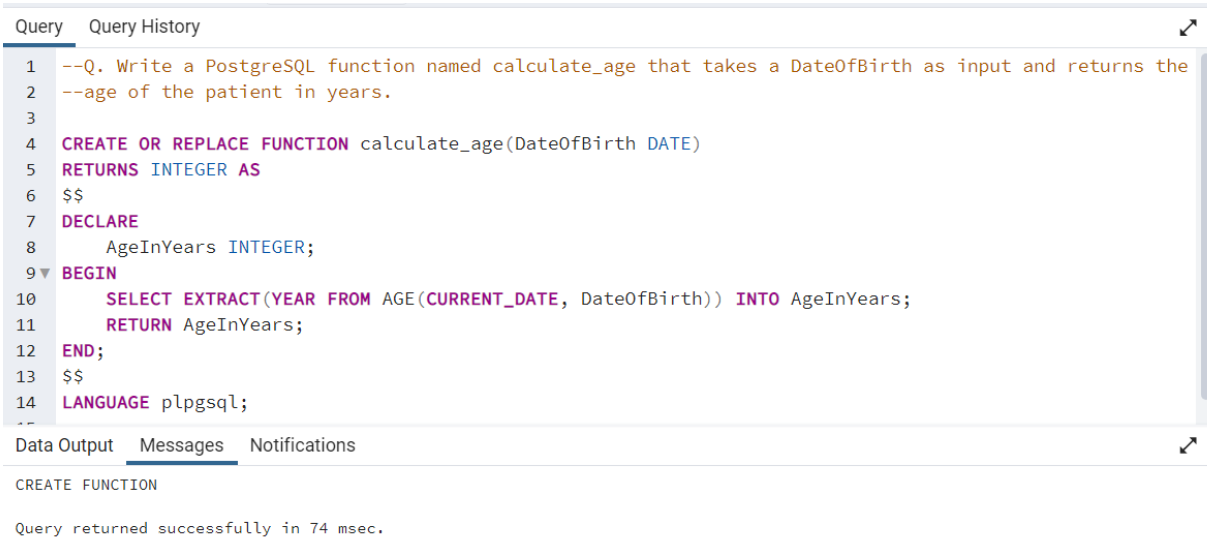
```sql
--Q. Identify a column in the Encounter table that is frequently used for filtering and sorting,
--and create an index to improve query performance for this column.

CREATE INDEX idx_DateOfService ON Encounter (DateOfService)

```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 47 msec.
```

- Functions: Custom Functions for Powerful SQL Queries

Q. Write a PostgreSQL function named calculate_age that takes a DateOfBirth as input and returns the age of the patient in years.



Query | Query History

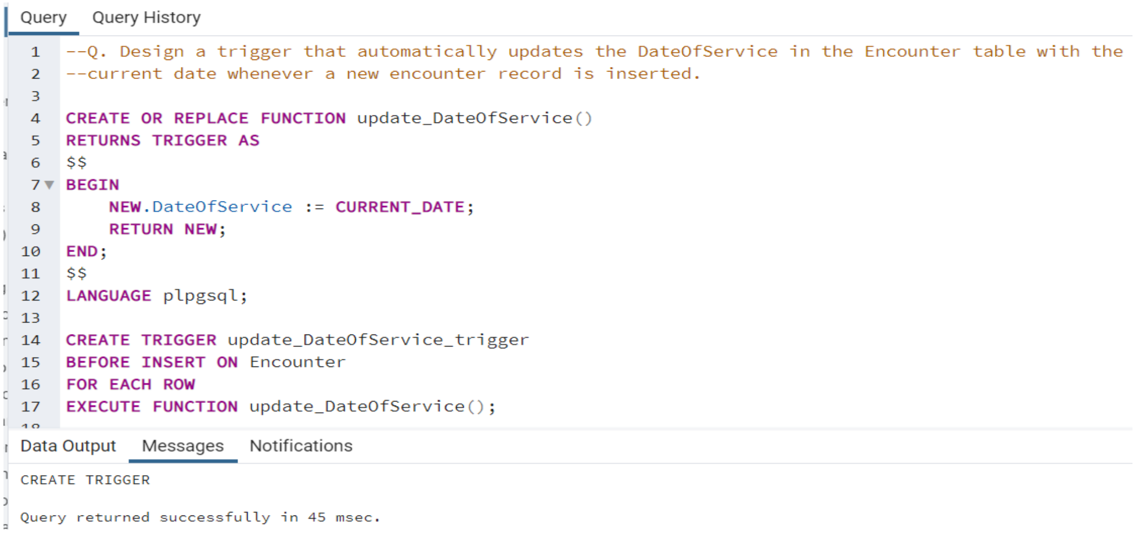
```sql
--Q. Write a PostgreSQL function named calculate_age that takes a DateOfBirth as input and returns the
--age of the patient in years.

CREATE OR REPLACE FUNCTION calculate_age(DateOfBirth DATE)
RETURNS INTEGER AS
$$
DECLARE
    AgeInYears INTEGER;
BEGIN
    SELECT EXTRACT(YEAR FROM AGE(CURRENT_DATE, DateOfBirth)) INTO AgeInYears;
    RETURN AgeInYears;
END;
$$
LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 74 msec.
```

- Triggers: Automating Actions with SQL Triggers

Q. Design a trigger that automatically updates the DateOfService in the Encounter table with the current date whenever a new encounter record is inserted.

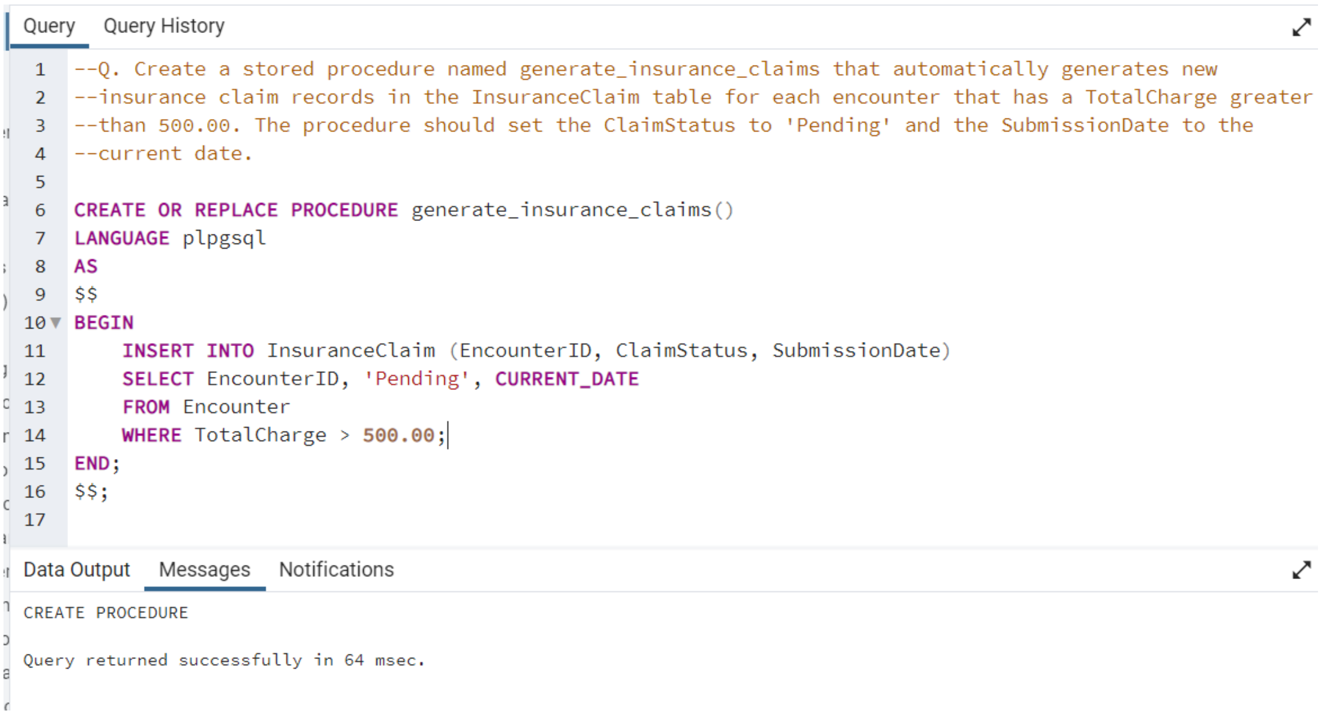


```sql
--Q. Design a trigger that automatically updates the DateOfService in the Encounter table with the
--current date whenever a new encounter record is inserted.

CREATE OR REPLACE FUNCTION update_DateOfService()
RETURNS TRIGGER AS
$$
BEGIN
    NEW.DateOfService := CURRENT_DATE;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_DateOfService_trigger
BEFORE INSERT ON Encounter
FOR EACH ROW
EXECUTE FUNCTION update_DateOfService();
```

```
CREATE TRIGGER

Query returned successfully in 45 msec.
```

- Stored Procedures: Efficiency and Reusability with Stored Procedures

Q. Create a stored procedure named generate_insurance_claims that automatically generates new insurance claim records in the InsuranceClaim table for each encounter that has a TotalCharge greater than 500.00. The procedure should set the ClaimStatus to 'Pending' and the SubmissionDate to the current date.



```sql
--Q. Create a stored procedure named generate_insurance_claims that automatically generates new
--insurance claim records in the InsuranceClaim table for each encounter that has a TotalCharge greater
--than 500.00. The procedure should set the ClaimStatus to 'Pending' and the SubmissionDate to the
--current date.

CREATE OR REPLACE PROCEDURE generate_insurance_claims()
LANGUAGE plpgsql
AS
$$
BEGIN
    INSERT INTO InsuranceClaim (EncounterID, ClaimStatus, SubmissionDate)
    SELECT EncounterID, 'Pending', CURRENT_DATE
    FROM Encounter
    WHERE TotalCharge > 500.00;
END;
$$;
```

```
CREATE PROCEDURE

Query returned successfully in 64 msec.
```

-----------------------------------END-----------------------------------