SQL Queries 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 EncounterID, DateOfService, and TotalCharge from the Encounter table.

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

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

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

- 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.

Q. Retrieve encounters sorted by TotalCharge in ascending order.

- 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 average TotalCharge for encounters.

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

Q. Retrieve patient information along with their encounter details.

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

- Subqueries: Use subqueries to nest queries within queries.

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

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

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

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

Q. Find the most common Diagnosis Code among encounters.

- 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

- 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:

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

- Updating and Deleting Data: UPDATE and DELETE statements.

Q. Update the email address for a specific patient.

Q. Delete encounters with a specific Diagnosis Code.

- 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.

Q. Create a view that shows encounters along with their corresponding insurance claim status.

- 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.

- 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.

- 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.

Q. Create a function named get_encounter_summary that takes a patient's PtID as input and returns a summary of their encounters, including the total number of encounters, the total charges incurred, and the average payment amount.

- 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.

Q. Implement a trigger named update_encounter_payment that updates the PaymentAmount in the Encounter table to 0.00 whenever an encounter record is updated with a TotalCharge less than 100.00.

- Stored Procedures: Efficiency and Reusability with Stored Procedures

Q. Write a stored procedure named add_new_patient that takes FirstName, LastName, DateOfBirth, Gender, PhoneNumber, Email, Address, InsuranceName, and InsuranceNumber as input parameters and inserts a new patient record into the Patient table.

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.