MAVEN HOSPITAL CHALLENGE

Challenge Objectives:

For the **Maven Hospital Challenge**, I will play the role of an Analytics Consultant for Massachusetts General Hospital (MGH).

I was tasked with building a high-level KPI report for the executive team, using a subset of patient records. I utilized PostgreSQL to uncover key insights, providing stakeholders with visibility into the hospital's recent performance and addressing the following questions:

- How many patients have been admitted or readmitted over time?
- How long are patients staying in the hospital, on average?
- How much is the average cost per visit?
- How many procedures are covered by insurance?

About the Dataset:

Synthetic data on ~1k patients of Massachussets General Hospital from 2011-2022, including information on patient demographics, insurance coverage, and medical encounters & procedures.

Dataset link: Maven Hospital Challenge

Tools Used →

Tools Used:

This project involved extensive use of **PostgreSQL**, including:

- **Data Manipulation:** Utilized **CREATE**, **UPDATE**, and **DELETE** statements.
- **Data Modeling:** Added **primary keys** and **foreign keys** to ensure data integrity.
- **Complex Queries**: Employed **joins**, **UNION** and **UNION ALL**, and **subqueries**.
- **Control Structures:** Implemented **cursors** with **loops** and **conditional statements** (**IF-ELSE, CASE).**
- **Temporary Structures**: Created **temporary tables**, common table expressions (**CTEs**), and **views**.
- **Aggregation and Analysis:** Used **aggregation functions**, **GROUP BY**, **window functions**, and **ORDER BY**.
- **Data Cleaning:** Added new columns, performed text cleaning using regular expressions (**REGEXP**), **concatenation**, and checked for **null values**.

Key Insights

Key Insights:

Encounters for Non-Admitted Patients

KPI's	
Total Patients	974
Total Encounters	27891
Admission Encounters	760
Non-Admission Encounters	26739
Males	46.88 %
Females	53.12 %
Most Encounters are Ambulatory	46.87 %
Average Cost Per Visit	$ 3442

Length of Stay for Non-Admitted Patients	
Under 15 mins	19313
Under an Hour	3376
Over an Hour	2929
Under 30 mins	645
Under 45 mins	476

Length Of Stay for Each Encounter Class (In Minutes)	
Emergency	61.03 Min
Ambulatory	55.43 Min
Outpatient	16.57 Min
Wellness	15.00 Min
Urgent Care	15.00 Min

Encounters for Admitted Patients

KPI's	
Total Patients	974
Total Admissions	623
Initial Admissions	347
Re-Admissions	276
Patients Admitted	143
Patients Never Admitted	831
Patients Re-Admitted	20
Males	65.33%
Females	34.67%
Average Cost Per Visit	$11630

Deaths	
Total Deaths	41
Patients Died During Admissions	10
Patients Died Within 30 days of their Admission	31

Reason for death	Admitted Patients	No Of Deaths	Ratio of Deaths
Alzheimer's disease (disorder)	19	19	100.00 %
COVID-19	17	8	47.00 %
Chronic congestive heart failure (disorder)	14	3	21.00 %
Familial Alzheimer's disease of early onset (disorder)	2	2	100.00 %
Primary small cell malignant neoplasm of lung TNM stage 1 (disorder)	53	1	1.00 %
COVID-19	17	8	47.00 %
Chronic congestive heart failure (disorder)	14	3	21.00 %

Length of Stay	
1 day admission	585
Under 20 days	28
Over a month	13
Under a month	4

Procedure Analysis

KPI's			
Total Procedures		47701	
Procedures done for Admitted Patients		2377	
Procedures done for Non-Admitted Patients		44546	
Total Procedure Cost		$105.52 M	
Average Procedure Cost		$2212	
Cost for Admission	**Payer Coverage**	**Cost for non-admission**	**Payer Coverage**
$8.84 M	$3.42 M **38.69%**	$92.04 M	$27.40 M **29.77 %**
Procedures Covered by Insurances		24791 51%	

Payers Insights

- Medicare Paid the most whereas Anthem paid none

Queries →

Queries for Table Creation

```sql
-- Creating Table Encounters
CREATE TABLE encounters (
    Id UUID PRIMARY KEY,
    Start TIMESTAMP,
    Stop TIMESTAMP,
    Patient UUID REFERENCES patients(id),
    Organization UUID REFERENCES organizations(id),
    Payer UUID REFERENCES payers(id),
    EncounterClass VARCHAR(50),
    Code VARCHAR(50),
    Description TEXT,
    Base_Encounter_Cost NUMERIC,
    Total_Claim_Cost NUMERIC,
    Payer_Coverage NUMERIC,
    ReasonCode VARCHAR(50),
    ReasonDescription TEXT
);

-- Creating Table Organizations
CREATE TABLE organizations (
    Id UUID PRIMARY KEY,
    Name VARCHAR(255),
    Address VARCHAR(255),
    City VARCHAR(100),
    State VARCHAR(50),
    Zip VARCHAR(10),
    Lat NUMERIC,
    Lon NUMERIC
);
```

```sql
-- Creating Table Patients
CREATE TABLE patients (
    Id UUID PRIMARY KEY,
    BirthDate DATE,
    DeathDate DATE,
    Prefix VARCHAR(10),
    First VARCHAR(100),
    Last VARCHAR(100),
    Suffix VARCHAR(10),
    Maiden VARCHAR(100),
    Marital CHAR(1),
    Race VARCHAR(100),
    Ethnicity VARCHAR(100),
    Gender CHAR(1),
    BirthPlace VARCHAR(100),
    Address VARCHAR(255),
    City VARCHAR(100),
    State VARCHAR(50),
    County VARCHAR(100),
    Zip VARCHAR(10),
    Lat NUMERIC,
    Lon NUMERIC
);

-- Creating Table Payers
CREATE TABLE payers (
    Id UUID PRIMARY KEY,
    Name VARCHAR(255),
    Address VARCHAR(255),
    City VARCHAR(100),
    State_Headquartered VARCHAR(50),
    Zip VARCHAR(10),
    Phone VARCHAR(20)
);


CREATE TABLE procedures (
    Start TIMESTAMP,
    Stop TIMESTAMP,
    Patient UUID REFERENCES patients(Id),
    Encounter UUID REFERENCES encounters(Id),
    Code VARCHAR(50),
    Description TEXT,
    Base_Cost NUMERIC,
    ReasonCode VARCHAR(50),
    ReasonDescription TEXT
);
```

Data Transformation

Patients

```sql
CREATE TEMP TABLE temp_patients AS
SELECT * FROM patients;

SELECT * FROM temp_patients;

-- Checking the data type
SELECT pg_typeof(birthdate)
FROM temp_patients;

-- Creating age column
ALTER TABLE temp_patients
ADD COLUMN age INT;

-- Updating the age column with the values
UPDATE temp_patients
SET age = EXTRACT(YEAR FROM AGE(birthdate));

-- Checking persons of particular age
SELECT age, COUNT(age)
FROM temp_patients
GROUP BY 1
ORDER BY 1 DESC;

-- Grouping ages
SELECT MAX(age) - MIN(age) AS range
FROM temp_patients;

-- Dividing into age groups
SELECT age,
CASE WHEN age <= 44 THEN 'Young'
WHEN age BETWEEN 45 AND 64 THEN 'Middle-Aged'
WHEN age > 64 THEN 'SENIOR'
ELSE 'Unknown'
END AS age_groups
FROM temp_patients;

-- Updating the table with calculated age-groups
ALTER TABLE temp_patients
ADD COLUMN age_groups VARCHAR(20);

UPDATE temp_patients
SET age_groups = CASE
WHEN age <= 35 THEN 'Young'
WHEN age BETWEEN 36 AND 55 THEN 'Middle-Aged'
WHEN age > 55 THEN 'SENIOR'
ELSE 'Unknown'
END;
```

```sql
-- Cleaning first and last name
SELECT first, REGEXP_REPLACE(first, '[^A-Za-z]+', '', 'g') AS cleaned_name
FROM temp_patients;

SELECT last, REGEXP_REPLACE(last, '[^A-Za-z]+', '', 'g') AS cleaned_name
FROM temp_patients;

-- Combining prefix first last and suffix into a single column name
SELECT TRIM(CONCAT(prefix, ' ', REGEXP_REPLACE(first, '[^A-Za-z]+', '', 'g'), ' ',
REGEXP_REPLACE(last, '[^A-Za-z]+', '', 'g'), ' ', suffix)) AS name,
LENGTH(TRIM(CONCAT(prefix, ' ', REGEXP_REPLACE(first, '[^A-Za-z]+', '', 'g'), ' ',
REGEXP_REPLACE(last, '[^A-Za-z]+', '', 'g'), ' ', suffix)))
FROM temp_patients
ORDER BY 2 DESC;

-- Updating the temp_patient table
ALTER TABLE temp_patients
ADD COLUMN name varchar(30);

UPDATE temp_patients
SET name = TRIM(CONCAT(prefix, ' ', REGEXP_REPLACE(first, '[^A-Za-z]+', '', 'g'), ' ',
REGEXP_REPLACE(last, '[^A-Za-z]+', '', 'g'), ' ', suffix));

-- Dropping prefix first last suffix and maiden
ALTER TABLE temp_patients
DROP prefix,
DROP first,
DROP last,
DROP suffix,
DROP maiden;

-- Inspecting marital stauts
SELECT gender
FROM temp_patients
WHERE gender IS NULL;

-- Checking for overall duplicates
SELECT id, COUNT(*)
FROM temp_patients
GROUP BY id
HAVING COUNT(*) > 1
ORDER BY id DESC;
```

Encounters

```sql
-- Creating temp table for encounters
CREATE TEMP TABLE temp_encounters AS
SELECT id, patient, start, stop, payer, encounterclass, description, base_encounter_cost, total_claim_cost,
    payer_coverage, reasoncode, reasondescription
FROM encounters;

-- Adding a column to check for rows that need to be dropped (there are overlap between admissions)
ALTER TABLE temp_encounters ADD COLUMN drop_row BOOLEAN;

-- Some admission records overlap with previous ones, so we identify and remove those duplicate entries.
DO $$
DECLARE
    e_id UUID; -- to hold encounter id
    p_id UUID; --  to hold patient id
    drop_row_check BOOLEAN; -- check to drop a row or not
    prev_discharge_date TIMESTAMP; -- to hold prev stop time
    order_column_1 TIMESTAMP; -- to hold start time
    order_column_2 TIMESTAMP; -- to hold stop time
    last_patient UUID := NULL;

    cur CURSOR FOR
        SELECT id, patient, start, stop
        FROM temp_encounters
        ORDER BY patient, start ASC, stop ASC;
BEGIN
    OPEN cur;
    LOOP
        FETCH cur INTO e_id, p_id, order_column_1, order_column_2;
        EXIT WHEN NOT FOUND;

        IF last_patient IS DISTINCT FROM p_id THEN
            prev_discharge_date := NULL;  -- Reset prev_discharge_date it means now we checking for new patient
        END IF;


        IF order_column_1 < prev_discharge_date THEN
            drop_row_check = TRUE;
            prev_discharge_date = GREATEST(order_column_2, prev_discharge_date);
        ELSE
            prev_discharge_date = order_column_2;
            drop_row_check = FALSE;
        END IF;

        UPDATE temp_encounters
        SET drop_row = drop_row_check
        WHERE patient = p_id AND start = order_column_1 AND stop = order_column_2;

        last_patient := p_id;

    END LOOP;
    CLOSE cur;
END $$;

-- Removing records that overlaps
DELETE FROM temp_encounters
WHERE drop_row = TRUE
```

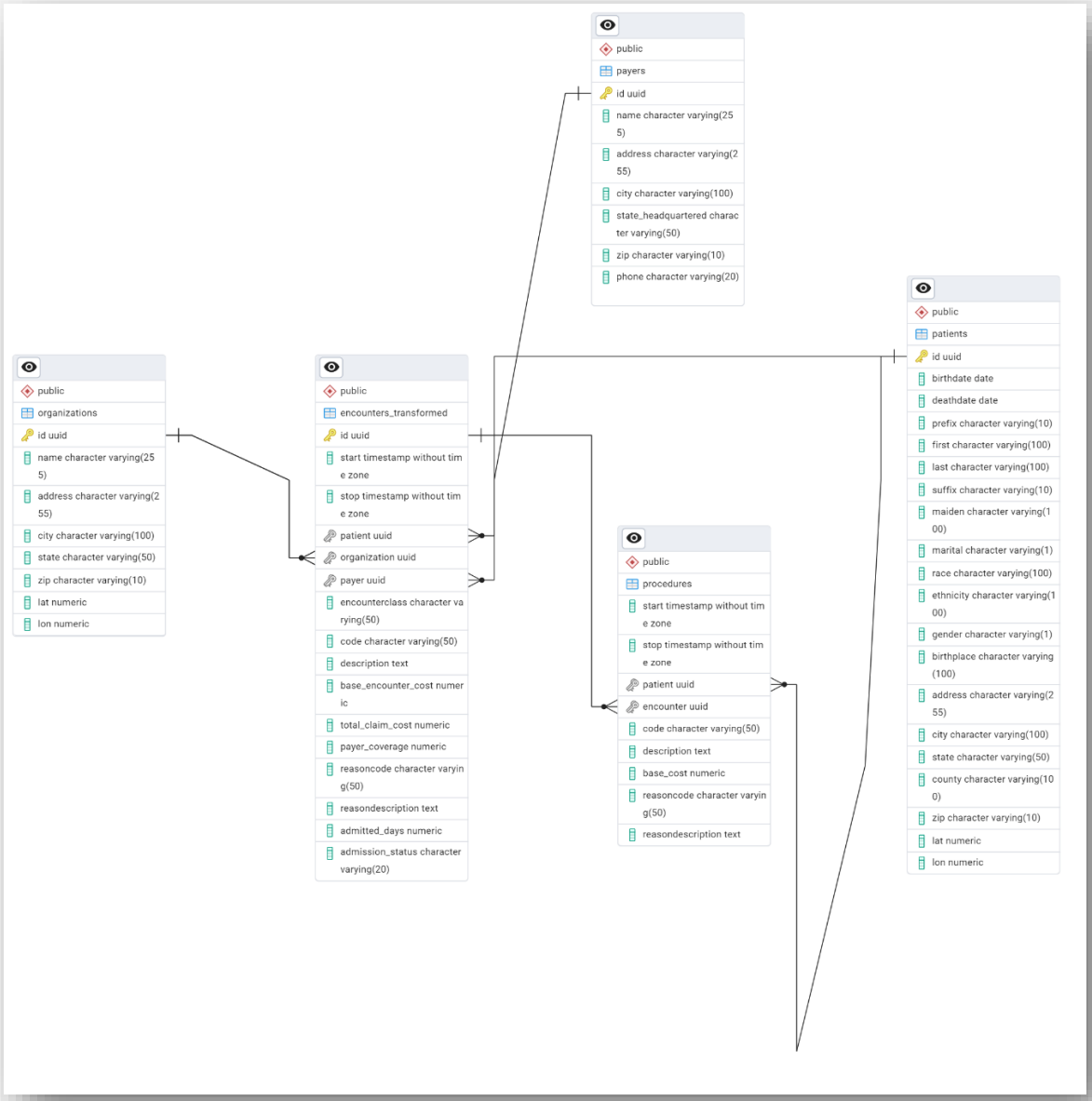
```sql
-- Creating another temp table just to make things clear
CREATE TEMP TABLE  admission_analysis AS
 SELECT id,
           patient,
           start,
           stop,
           EXTRACT(DAY FROM (stop - start)) AS admitted_days,
           LAG(stop) OVER (PARTITION BY patient ORDER BY start ASC, stop ASC) AS prev_discharge,
           EXTRACT(DAY FROM (start - LAG(stop) OVER (PARTITION BY patient ORDER BY start ASC, stop ASC))) AS days_interval_between_admission,
           0 AS running_total
    FROM temp_encounters;


-- Adding running total that helps us in identifying readmissions
DO $$
DECLARE
    r_t INT := 0; -- variable to update running total
    r_t_c INT := 0; -- variable to hold current running total in each iteration
    admitt_days INT := 0;
    days_interval INT := 0;
    order_column_1 TIMESTAMP; -- to store start time
    order_column_2 TIMESTAMP; --  to store stop time
    p_id UUID; -- to store patient id
    p_d TIMESTAMP; -- to store previous discharge of patient
    cur CURSOR FOR
        SELECT admitted_days, days_interval_between_admission, patient, prev_discharge, start, stop
        FROM admission_analysis
        ORDER BY patient, start ASC, stop ASC;
BEGIN
    OPEN cur;
    LOOP
        FETCH cur INTO admitt_days, days_interval, p_id, p_d, order_column_1, order_column_2;
        EXIT WHEN NOT FOUND;

        IF p_d IS NULL THEN
            r_t := 0;
            r_t_c := 0;
        ELSIF admitt_days >= 1 THEN
            r_t := r_t_c + days_interval;
            r_t_c := 0;
        ELSE
            r_t_c := r_t_c + days_interval;
            r_t := r_t_c;
        END IF;

        UPDATE admission_analysis
        SET running_total = r_t
        WHERE patient = p_id AND start = order_column_1 AND stop = order_column_2;
    END LOOP;
    CLOSE cur;
END $$;
```

```sql
-- Adding a column that checks for admission status
ALTER TABLE admission_analysis ADD COLUMN admission_status VARCHAR(20);

-- Updating the admission status column with the initial admission, readmission and visited status
WITH MinStart AS (
    SELECT
        id,
        patient,
        start,
        admitted_days,
        running_total,
        days_interval_between_admission,
        MIN(CASE WHEN admitted_days >= 1 THEN start END) OVER (PARTITION BY patient) AS min_start
    FROM admission_analysis
)
UPDATE admission_analysis
SET admission_status = subquery.admission_status
FROM (
    SELECT
        id,
        patient,
        start,
        admitted_days,
        CASE
            WHEN (start = min_start AND admitted_days >= 1) OR (start > min_start AND running_total > 30 AND admitted_days >= 1 ) THEN 'initial admission'
            WHEN (admitted_days >= 1 AND days_interval_between_admission = 0 AND running_total = 0) THEN 'continuous admission'
            WHEN (admitted_days >= 1) THEN 'readmission'
            ELSE 'visited'
        END AS admission_status
    FROM MinStart
) AS subquery
WHERE admission_analysis.id = subquery.id AND admission_analysis.patient = subquery.patient AND admission_analysis.start = subquery.start;



-- creating a new table that contains both the admission details as well as encounter detail
CREATE TABLE encounters_transformed AS
SELECT t1.*, t2.admitted_days, t2.admission_status -- Add more columns as needed
FROM encounters t1
LEFT JOIN admission_analysis t2
ON t1.id = t2.id;

ALTER TABLE encounters_transformed
ADD CONSTRAINT fk_org
FOREIGN KEY (organization)
REFERENCES organizations (id);

-- Inspecting encounterclass to check for missplled or incorrect data
SELECT DISTINCT encounterclass
FROM encounters_transformed

-- Now checking for null values
SELECT COUNT(encounterclass)
FROM encounters_transformed
WHERE encounterclass IS NULL;

-- Inspecting encounters description
SELECT DISTINCT description
FROM encounters_transformed

UPDATE encounters_transformed
SET description = 'Encounter for check-up'
WHERE description = 'Encounter for ''check-up'''
```

Data Modelling

```sql
-- Now making the id of encounters_transformed primary key
-- Step 1: Ensure the column has unique values
-- You can add a unique constraint if necessary
ALTER TABLE encounters_transformed
ADD CONSTRAINT unique_encounters_id UNIQUE (id);

-- Step 2: Add the primary key constraint
ALTER TABLE encounters_transformed
ADD CONSTRAINT pk_encounters_id PRIMARY KEY (id);

-- Managing relations with this new encounters_transformed table
ALTER TABLE encounters_transformed
ADD CONSTRAINT fk_patient
FOREIGN KEY (patient) REFERENCES patients(id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE encounters_transformed
ADD CONSTRAINT fk_organization
FOREIGN KEY (organization) REFERENCES organizations(id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE encounters_transformed
ADD CONSTRAINT fk_payer
FOREIGN KEY (payer) REFERENCES payers(id)
ON DELETE CASCADE
ON UPDATE CASCADE;

-- Managing relations between encounters_transformed and procedure table
ALTER TABLE procedures
ADD CONSTRAINT fk_enc_transformed
FOREIGN KEY (encounter) REFERENCES encounters_transformed(id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

ERD


public
payers
id uuid
name character varying(255)
address character varying(255)
city character varying(100)
state_headquartered character varying(50)
zip character varying(10)
phone character varying(20)
public
patients
id uuid
birthdate date
deathdate date
prefix character varying(10)
first character varying(100)
last character varying(100)
suffix character varying(10)
maiden character varying(100)
marital character varying(1)
race character varying(100)
ethnicity character varying(100)
gender character varying(1)
birthplace character varying(100)
address character varying(255)
city character varying(100)
state character varying(50)
county character varying(100)
zip character varying(10)
lat numeric
lon numeric
public
organizations
id uuid
name character varying(255)
address character varying(255)
city character varying(100)
state character varying(50)
zip character varying(10)
lat numeric
lon numeric
public
encounters_transformed
id uuid
start timestamp without time zone
stop timestamp without time zone
patient uuid
organization uuid
payer uuid
encounterclass character varying(50)
code character varying(50)
description text
base_encounter_cost numeric
total_claim_cost numeric
payer_coverage numeric
reasoncode character varying(50)
reasondescription text
admitted_days numeric
admission_status character varying(20)
public
procedures
start timestamp without time zone
stop timestamp without time zone
patient uuid
encounter uuid
code character varying(50)
description text
base_cost numeric
reasoncode character varying(50)
reasondescription text

Analysis of Non-Admissions

```sql
-- Total Number of Patients
SELECT COUNT(DISTINCT id) Total_Number_of_Patients
FROM patients

-- Creating a view for non-admissions
CREATE VIEW enc_non_adm_view AS
SELECT *
FROM encounters_transformed
WHERE (admission_status IS NULL OR admission_status = 'visited') AND encounterclass <> 'inpatient'

-- Creating a view for non-admission total encounters to use it multiple times
CREATE VIEW total_enc_view AS
SELECT COUNT(id) AS total_encounters
FROM enc_non_adm_view;

-- Total number of encounters vs admission and non-admission encounters
SELECT
(SELECT COUNT(id) TotalEncounters FROM encounters_transformed),
(SELECT COUNT(id) FROM enc_adm_view) admission_encounters,
(SELECT total_encounters FROM total_enc_view) non_adm_encounters

-- Non-admissions by gender
SELECT p.gender, COUNT(*) total_encounters, ROUND(COUNT(*) * 100 / (SELECT total_encounters::NUMERIC FROM total_enc_view), 2) AS Ratio_of_gender
FROM patients p
JOIN enc_non_adm_view et
ON p.id = et.patient
GROUP BY 1;

-- Number of encounters by encounter class splitted by gender
SELECT ev.encounterclass, p.gender, COUNT(*) numberofencounters, ROUND(COUNT(*) *100 / (SELECT total_encounters::NUMERIC FROM total_enc_view), 2) AS Ratio_of_total
FROM enc_non_adm_view ev
JOIN patients p
ON ev.patient = p.id
GROUP BY 1, 2
ORDER BY 3 DESC;

-- Number of encounters by encounter class
SELECT ev.encounterclass, COUNT(*) numberofencounters, ROUND(COUNT(*) *100 / (SELECT total_encounters::NUMERIC FROM total_enc_view), 2) AS Ratio_of_total
FROM enc_non_adm_view ev
GROUP BY 1
ORDER BY 3 DESC;

-- Number of encounters by encounter description
SELECT description, COUNT(*)
FROM enc_non_adm_view
GROUP BY 1
ORDER BY 2 DESC

-- Number of encounters by encounter reason
SELECT reasondescription, COUNT(*)
FROM enc_non_adm_view
GROUP BY 1
ORDER BY 2 DESC

-- Procedures by encounter class
SELECT encounterclass, COUNT(*) total_procedures, CONCAT('$', ROUND(AVG(proc.base_cost), 2)) average_procedure_cost
FROM enc_non_adm_view ev
JOIN procedures proc
ON ev.id = proc.encounter
GROUP BY 1
ORDER BY 2 DESC
```

```sql
-- Inspecting different costs
SELECT ev.encounterclass, CONCAT('$', ROUND(AVG(ev.base_encounter_cost), 1)) avg_base_cost,
CONCAT('$', ROUND(AVG(proc.base_cost), 1)) avg_proc_cost, CONCAT('$', ROUND(AVG(ev.total_claim_cost), 1)) avg_total_cost
FROM enc_non_adm_view ev
JOIN procedures proc
ON ev.id = proc.encounter
GROUP BY 1

-- Inspecting payer cost
SELECT encounterclass, CONCAT('$', SUM(total_claim_cost)) TotalCost, CONCAT('$', SUM(payer_coverage)) PayerCoverage,
CONCAT(ROUND(SUM(payer_coverage) * 100 / SUM(total_claim_cost), 2), ' %') Ratio_of_coverage
FROM enc_non_adm_view
GROUP BY 1
ORDER BY 4 DESC

-- Inspecting average length of stay in mins for non-admission by encounter class
SELECT encounterclass, CONCAT(AVG(EXTRACT(EPOCH FROM (stop - start)) / 60)::numeric(10, 2), ' min') AS lengthofst
FROM enc_non_adm_view
GROUP BY 1
ORDER BY 2 DESC;

-- Inspecting most non-admitted patients length of stay
WITH stay AS(
SELECT id, (EXTRACT(EPOCH FROM (stop-start)) / 60)::numeric(10, 2) los
FROM enc_non_adm_view),

grouped_los AS (
SELECT id, CASE
WHEN los <= 15.00 THEN 'Under 15 mins'
WHEN los <= 30.00 THEN 'Under 30 mins'
WHEN los <= 45.00 THEN 'Under 45 mins'
WHEN los <= 60.00 THEN 'Under an Hour'
ELSE 'Over an Hour'
END length_of_stay
FROM stay
)

SELECT length_of_stay, COUNT(id)
FROM grouped_los
GROUP BY 1
ORDER BY 2 DESC;

-- How much on average per visit costs
SELECT CONCAT('$', (AVG(total_claim_cost))::numeric(10, 2)) Average_Cost_per_Visit
FROM encounters_transformed

-- How much on average per visit costs for non-admitted patients
SELECT CONCAT('$', (AVG(total_claim_cost))::numeric(10, 2)) Average_Cost_per_Visit
FROM enc_non_adm_view
```

Analysis of Admissions

```sql
-- Creating a view for admissions
CREATE VIEW enc_adm_view AS
SELECT *
FROM encounters_transformed
WHERE admission_status IN ('initial admission', 'readmission', 'continuous admission')

-- Counting the total number of initial admission
SELECT COUNT(*) AS total_initial_admissions
FROM enc_adm_view
WHERE admission_status = 'initial admission'

-- Counting the total number of readmissions
SELECT COUNT(*) AS total_readmissions
FROM enc_adm_view
WHERE admission_status = 'readmission'

-- Counting the total number of admissions
SELECT COUNT(*) AS total_admissions
FROM enc_adm_view
WHERE admission_status = 'initial admission' OR admission_status = 'readmission'

-- Counting the total number of patients admitted
SELECT
    COUNT(DISTINCT patient) AS admitted_patients_count
FROM
    enc_adm_view;

-- Counting the total number of patients who have never been admitted.
SELECT
    COUNT(DISTINCT patient) AS never_admitted_patients_count
FROM
    encounters_transformed
WHERE
    patient NOT IN (SELECT DISTINCT patient FROM enc_adm_view);


-- Counting the total number of patients who have been re-admitted.
WITH re_admitted_patients AS (
    SELECT
        DISTINCT patient
    FROM
        enc_adm_view
    WHERE
        admission_status = 'readmission'
)
SELECT
    p.gender, COUNT(DISTINCT patient) AS readmitted_patients
FROM
    re_admitted_patients
JOIN patients p
ON re_admitted_patients.patient = p.id
GROUP BY 1
ORDER BY 2

-- Gender distribution for admissions
SELECT p.gender, COUNT(*),
ROUND(COUNT(*) * 100 / (SELECT COUNT(*)::NUMERIC FROM enc_adm_view WHERE admission_status <> 'continuous admission'), 2) AS Ratio_of_gender
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status <> 'continuous admission'
GROUP BY 1
ORDER BY 2 DESC;
```

```sql
-- Gender distribution for initial admissions
SELECT p.gender, COUNT(*)
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status = 'initial admission'
GROUP BY 1
ORDER BY 2 DESC;

-- Gender distribution for re-admissions
SELECT p.gender, COUNT(*)
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status = 'readmission'
GROUP BY 1
ORDER BY 2 DESC;

-- admission by encounter description
SELECT ev.description, COUNT(*)
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status = 'initial admission'
GROUP BY 1
ORDER BY 2 DESC;

-- admission by encounter reason
SELECT ev.reasondescription, COUNT(*)
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status = 'initial admission'
GROUP BY 1
ORDER BY 2 DESC;

-- readmissions by encounter description
SELECT ev.description, COUNT(*)
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status = 'readmission'
GROUP BY 1
ORDER BY 2 DESC;

-- readmissions by encounter reason
SELECT ev.reasondescription, COUNT(*)
FROM patients p
JOIN enc_adm_view ev
ON p.id = ev.patient AND ev.admission_status = 'readmission'
GROUP BY 1
ORDER BY 2 DESC;

-- Creating a view for patients died within 30 days of their admissi
on
CREATE VIEW deaths_within_month AS

WITH latest_date AS (
SELECT patient, MAX(STOP) last_date
FROM enc_adm_view
WHERE admitted_days >= 1
GROUP BY patient
)

SELECT ev.*
FROM enc_adm_view ev
JOIN patients p
ON p.id = ev.patient AND p.deathdate > ev.stop AND p.deathdate <= (e
v.stop + INTERVAL '30 days')
JOIN latest_date
ON ev.patient = latest_date.patient AND ev.stop = latest_date.last_d
ate
```

```sql
-- total number of patients died with 30 days of their previous admission
SELECT COUNT(patient)
FROM deaths_within_month

-- Whats their encounter reason
SELECT description, COUNT(*)
FROM deaths_within_month
GROUP BY 1
ORDER BY 2 DESC

-- Whats their disease
SELECT reasondescription, COUNT(*)
FROM deaths_within_month
GROUP BY 1
ORDER BY 2 DESC

-- VIEW for patients dies during admission in hospital
CREATE VIEW deaths_during_admission AS

SELECT patient, description, reasondescription
FROM enc_adm_view ev
JOIN patients p
ON ev.patient = p.id AND p.deathdate BETWEEN ev.start AND ev.stop

-- Total number of patient died during admission
SELECT COUNT(*)
FROM deaths_during_admission

-- Whats their encounter reason
SELECT description, COUNT(*)
FROM deaths_during_admission
GROUP BY 1
ORDER BY 2 DESC

-- Whats their disease
SELECT reasondescription, COUNT(*)
FROM deaths_during_admission
GROUP BY 1
ORDER BY 2 DESC

-- total deaths during admssion or within 30 days from last discharge
SELECT
(SELECT COUNT(*) FROM deaths_within_month dm) +
(SELECT COUNT(*) FROM deaths_during_admission) AS total_deaths

-- We have inspected individually for boht death cases now inspecting overall
-- view to handle that
CREATE VIEW total_deaths AS
(
SELECT patient, description, reasondescription FROM deaths_within_month
UNION ALL
SELECT patient, description, reasondescription FROM deaths_during_admission
)
```

```sql
-- Gender Distribution of deaths
SELECT p.gender, COUNT(*) deaths
FROM patients p
JOIN total_deaths td
ON p.id = td.patient
GROUP BY 1
ORDER BY 2 DESC

-- desciption for most deaths
SELECT description, COUNT(*)
FROM total_deaths
GROUP BY 1
ORDER BY 2 DESC

-- Reason for most deaths
SELECT reasondescription, COUNT(*)
FROM total_deaths
GROUP BY 1
ORDER BY 2 DESC

-- Number of patients died vs admitted patients for the same disease
WITH adm AS (
SELECT reasondescription, COUNT(*) admitted_patients
FROM enc_adm_view
WHERE admission_status <> 'continuous admission'
GROUP BY 1
),

dead AS (
SELECT reasondescription, COUNT(*) died_patients
FROM total_deaths
GROUP BY 1
)

SELECT a.reasondescription, a.admitted_patients, d.died_patients, (d.died_patients * 100
/ a.admitted_patients)::numeric(8,2) ratio_of_death
FROM adm a
JOIN dead d
ON a.reasondescription = d.reasondescription
ORDER BY 3 DESC

-- Average length of stay for admitted patients
WITH stay AS(
SELECT id, (EXTRACT(DAY FROM (stop-start))) los
FROM enc_adm_view
WHERE admission_status <>'continuous admission' OR admitted_days > 1),

grouped_los AS (
SELECT id, CASE
WHEN los = 1 THEN '1 day admission'
WHEN los <= 15 THEN 'Under 20 days'
WHEN los <= 30 THEN 'Under a month'
ELSE 'Over an month'
END length_of_stay
FROM stay
)

SELECT length_of_stay, COUNT(id)
FROM grouped_los
GROUP BY 1
ORDER BY 2 DESC;

-- How much on average per visit costs for admitted patients
SELECT CONCAT('$', (AVG(total_claim_cost))::numeric(10, 2)) Average_Cost_per_Visit
FROM enc_adm_view
```

Procedure Analysis

```sql
-- Inspecting the procedure table
SELECT *
FROM procedures;

-- Counting the total number of procedures for admssions and non-admissions
WITH non_adm_proc AS(
SELECT COUNT(*) Non_Adm
FROM procedures proc
JOIN enc_non_adm_view ev
ON proc.encounter = ev.id),

adm_proc AS (
SELECT COUNT(*) Adm
FROM procedures proc
JOIN enc_adm_view ev -- view for admissions
ON proc.encounter = ev.id
)

SELECT
(SELECT COUNT(*) Total_Procedures FROM procedures),
(SELECT Non_Adm FROM non_adm_proc),
(SELECT Adm FROM adm_proc)

-- Total Procedure cost
SELECT CONCAT('$', (SUM(base_cost)/1000000)::numeric(10,2), ' M') total_procedure_cost
FROM procedures

-- Procedure cost vs procedure description
SELECT description,
COUNT(*) number_of_procedures_performed, (SUM(base_cost)/1000000)::numeric(10,2) total_procedure_cost_in_million,
(SUM(base_cost)/COUNT(*))::numeric(10,2) avg_procedure_cost_in_million
FROM procedures
GROUP BY 1
ORDER BY 3 DESC

-- Total procedure cost for encounter class
SELECT et.encounterclass,
(SUM(proc.base_cost)/1000000)::numeric(10,2) total_procedure_cost_in_million
FROM encounters_transformed et
JOIN procedures proc
ON et.id = proc.encounter
GROUP BY 1
ORDER BY 2 DESC

-- Analysing procedure cost between admissions and non-admissions
WITH adm_proc AS (
SELECT CONCAT('$', (SUM(proc.base_cost) / 1000000)::numeric(8,2), ' M') admission_procedures_cost
FROM procedures proc
JOIN enc_adm_view ev
ON proc.encounter = ev.id
),

non_adm_proc AS (
SELECT CONCAT('$', (SUM(proc.base_cost) / 1000000)::numeric(8,2), ' M') non_admissions_procedures_cost
FROM procedures proc
JOIN enc_non_adm_view ev
ON proc.encounter = ev.id
)

SELECT (SELECT CONCAT('$', (SUM(base_cost) / 1000000)::numeric(8,2), ' M') FROM procedures) totalprocedurescost,
ad.admission_procedures_cost, nd.non_admissions_procedures_cost
FROM adm_proc ad, non_adm_proc nd
```

```sql
-- Analyzing payer coverage between admissions and not admissions
WITH adm_cost AS (
SELECT 'Admitted' AS admission_status,
CONCAT('$', (SUM(total_claim_cost) / 1000000)::numeric(8,2), ' M') procedures_cost,
(SUM(total_claim_cost) / 1000000)::numeric(8,2) procedures_cost_value,
CONCAT('$', (SUM(payer_coverage) / 1000000)::numeric(8,2), ' M') payer_coverage,
(SUM(payer_coverage) / 1000000)::numeric(8,2) payer_coverage_value
FROM enc_adm_view ev
),

non_adm_cost AS (
SELECT 'Non-Admitted' AS admission_status,
CONCAT('$', (SUM(total_claim_cost) / 1000000)::numeric(8,2), ' M') procedures_cost,
(SUM(total_claim_cost) / 1000000)::numeric(8,2) procedures_cost_value,
CONCAT('$', (SUM(payer_coverage) / 1000000)::numeric(8,2), ' M') payer_coverage,
(SUM(payer_coverage) / 1000000)::numeric(8,2) payer_coverage_value
FROM enc_non_adm_view ev
)

SELECT admission_status, procedures_cost, payer_coverage,
CONCAT('$', (payer_coverage_value / procedures_cost_value)::numeric(8,2), ' M') coverage_ratio FROM adm_cost
UNION
SELECT admission_status, procedures_cost, payer_coverage,
CONCAT('$', (payer_coverage_value / procedures_cost_value)::numeric(8,2), ' M') coverage_ratio FROM non_adm_cost

-- How many procedures are covered by insurance
WITH proc_count as (
SELECT COUNT(*) total_procedures
FROM procedures)

SELECT (SELECT total_procedures FROM proc_count) Total_Procedures, COUNT(*) Procedures_Covered_by_Insurance,
(COUNT(*) * 100 / (SELECT total_procedures FROM proc_count))::numeric(10, 2) Percent_of_Total
FROM encounters_transformed et
JOIN procedures proc
ON et.id = proc.encounter AND et.patient = proc.patient
WHERE payer_coverage > 0

-- Number of procedures for each gender
SELECT p.gender, COUNT(proc.*) total_procedures, (COUNT(proc.*) * 100 /(SELECT COUNT(*) FROM procedures))::numeric(10,2) percent_of_total,
SUM(proc.base_cost) total_cost, (SUM(proc.base_cost) * 100/(SELECT SUM(base_cost) FROM procedures))::numeric(10,2) percent_of_total_cost
FROM patients p
JOIN procedures proc
ON p.id = proc.patient
GROUP BY 1
ORDER BY 2 DESC

-- which payer pays the most
SELECT p.name, SUM(et.payer_coverage)
FROM payers p
JOIN encounters_transformed et
ON p.id = et.payer
GROUP BY 1
ORDER BY 2 DESC
```