Use Case: Diabetes Patient Data Analysis

Objective: To explore the dataset of diabetes patients, identify trends, and uncover insights such as –

- Demographic distribution of patients.
- Trends in glucose levels across age groups.
- Readmission rates and associated factors.

Our Schema:

PatientID: *Unique identifier for each patient.*
Age: *Patient's age group (e.g., "20–30", "30–40").*
Gender: *Male or Female.*
BMI: *Body Mass Index.*
GlucoseLevel: *Fasting glucose level.*
Outcome: *0 (No diabetes) or 1 (Diabetes diagnosis).*
Readmission: *1 (Readmitted) or 0 (Not readmitted).*
VisitDate: *Date of hospital visit.*

Dataset:

Download the dataset: *diabetes_data.csv,* from the below GitHub location:

https://github.com/ItishaK/codeChallenges/blob/main/diabetes_dataset1.csv

Discussion Topics:

- **EDA (Exploratory Data Analysis):** *Detailed process of examining Data with an intention to uncover patterns, trends and relationships.*

- **Multi-Hop Architecture:** Historical -> Raw_Table -> Refined_Table -> Derived/Transformed_Table

Exercise:

We will implement EDA steps in postgresql to analyze our Healthcare dataset.

Step 1: Create table- 'patientDetails_rw' in postgres as per the schema provided above but we will keep the data types as 'VARCHAR' in order to prevent any loss of data due to data type mismatch.

DDL:

```
CREATE TABLE patientDetails_rw (
   patientID VARCHAR(10) PRIMARY KEY,
   age VARCHAR(10),
   gender VARCHAR(10),
   bmi VARCHAR(10),
   glucoseLevel VARCHAR(10),
   outcome VARCHAR(10),
   readmission VARCHAR(10),
   visitDate VARCHAR(10)
);
```

Step 2: Import dataset- '*diabetes_data.csv'* into the table '*patientDetails_rw*'. This table will serve as our raw table which contains the data as it is. We have used this naming convention for maintaining an easy-to-understand code.

Script-

```
COPY patientDetails_rw(patientID, age, gender, bmi, glucoseLevel, outcome, readmission,
visitDate)
FROM 'input_path\diabetes_data.csv'
delimiter ','
csv header ;
```

Also, maintain a backup of this data (as historical data), let's say: '*patientDetails_history'.*

```
create table patientDetails_history
as select * from patientDetails_rw;
```

Step 3: Cross-check the data import with a simple select statement on the table- '*patientDetails*'.

select count(*) from patientDetails_rw as TotalRows;

select count(*) from patientDetails_history as TotalRows;

Step 4: Perform Basic **Data Quality Checks** to ensure high quality data in our raw tables.

- Check for Overall Row Count.
- Check for NULL value counts for each Column.
- Inspect the NULL values (if any).
- Check for duplicate Primary Key values.
- Check for invalid date formats.
- Check for Outliers like:
    - Out of range BMI values (<10 and >50)
    - Invalid Glucose Level or BMI values
    - Logical Inconsistency in outcome and readmission values
- Check patient counts for distinct age groups and gender.
- Validate distinct values for columns with restricted value range like age group, outcome, readmission etc.

Step 5: Data Cleaning – We will clean the data by performing the below checks.

- Cross-check the row-counts and values in the refined table- '*patientDetails_ref*'.
- Remove or update the records that are not needed (delete only when it is utmost necessary).
- Change date formats into required format (if needed).

Hints:

- Add steps for outcome value 0 written as 'o' and '0'.
- Gender values written as 'NA', None, ''.
- Boolean columns having more than 2 values like 'Yes' or 'No'.
- Incorrect date formats or invalid dates.
- Negative value for glucose levels and BMI values.
- Out of range values for BMI.

Step 6: In this step we will create a new refined table '*patientDetails_ref*' with updated data types as per our schema. Maintaining this naming convention plays a significant role in code maintenance.

Script:

```
create table patientDetails_ref(
   patientID VARCHAR(10) PRIMARY KEY,
   age VARCHAR(10),
   gender VARCHAR(10),
   bmi DOUBLE PRECISION,
   glucoseLevel INTEGER,
   outcome INTEGER,
   readmission INTEGER,
   visitDate DATE
);
```

Now, using our insertion script, we will cast all values from our raw table before inserting them into our refined table.

Script:

```
insert into patientDetails_ref
select
   patientID,
   age,
   gender,
   CAST(bmi as DOUBLE PRECISION),
   CAST(glucoseLevel AS INTEGER),
   CAST(outcome as INTEGER),
   CAST(readmission as INTEGER),
   CAST(visitDate as DATE)
from patientDetails_rw;
```

Step 7: In this step we will add derived columns on top of our refined table. For this, we will create a derived temporary table/view (as per the suitability) - '*patientDetails_derived_tbl' or 'patientDetails_derived_vw'.*

- Categorize BMI values into meaningful ranges (like 'Overweight', 'Obese' etc).
- Add a derived column to indicate Patient status using columns 'outcome' and 'readmission'.
- Count of Diabetic vs Non-Diabetic patients by Gender.
- Percentage of Diabetic Patients by Gender.
- Count of Patients (Diabetic/Non-Diabetic) by Gender and BMI Category.