**NYC DATA INTEGRATION DATA WAREHOUSE DESIGN**

The City of New York embarked on a project to integrate payroll data across all its agencies. The objective of the project is to develop a Data Analytics platform to achieve two primary goals which include Financial Resource Allocation Analysis and Transparency/Public Accessibility.

By developing this Data Analytics platform, the City of New York aims to improve financial management and enhance transparency in its operations, benefiting both the administration and the public.

**Project Scope**

The scope of the project encompassed the following

1. Designing a Data Warehouse for NYC
2. Developing a scalable and automated ETL Pipeline
3. Ensuring data quality and consistency
4. Creating a public user with limited privileges
5. Documenting processes for reproducibility

The tables for PostgreSQL NYC\_payroll database was created:

CREATE TABLE AgencyMaster (

AgencyID INT PRIMARY KEY,

AgencyName VARCHAR(255)

);

CREATE TABLE EmpMaster (

EmployeeID INT PRIMARY KEY,

LastName VARCHAR(255),

FirstName VARCHAR(255)

);

CREATE TABLE TitleMaster (

TitleCode INT PRIMARY KEY,

TitleDescription VARCHAR(255)

);

CREATE TABLE NYCPayroll\_2020 (

FiscalYear INT,

PayrollNumber INT,

AgencyID INT,

AgencyName VARCHAR(255),

EmployeeID INT,

LastName VARCHAR(255),

FirstName VARCHAR(255),

AgencyStartDate DATE,

WorkLocationBorough VARCHAR(255),

TitleCode INT,

TitleDescription VARCHAR(255),

LeaveStatusasofJune30 VARCHAR(255),

BaseSalary DECIMAL,

PayBasis VARCHAR(255),

RegularHours DECIMAL,

RegularGrossPaid DECIMAL,

OTHours DECIMAL,

TotalOTPaid DECIMAL,

TotalOtherPay DECIMAL,

PRIMARY KEY (FiscalYear, PayrollNumber, EmployeeID),

FOREIGN KEY (AgencyID) REFERENCES AgencyMaster(AgencyID),

FOREIGN KEY (EmployeeID) REFERENCES EmpMaster(EmployeeID),

FOREIGN KEY (TitleCode) REFERENCES TitleMaster(TitleCode)

);

CREATE TABLE NYCPayroll\_2021 (

FiscalYear INT,

PayrollNumber INT,

AgencyID INT,

AgencyName VARCHAR(255),

EmployeeID INT,

LastName VARCHAR(255),

FirstName VARCHAR(255),

AgencyStartDate DATE,

WorkLocationBorough VARCHAR(255),

TitleCode INT,

TitleDescription VARCHAR(255),

LeaveStatusasofJune30 VARCHAR(255),

BaseSalary DECIMAL,

PayBasis VARCHAR(255),

RegularHours DECIMAL,

RegularGrossPaid DECIMAL,

OTHours DECIMAL,

TotalOTPaid DECIMAL,

TotalOtherPay DECIMAL,

PRIMARY KEY (FiscalYear, PayrollNumber, EmployeeID),

FOREIGN KEY (AgencyID) REFERENCES AgencyMaster(AgencyID),

FOREIGN KEY (EmployeeID) REFERENCES EmpMaster(EmployeeID),

FOREIGN KEY (TitleCode) REFERENCES TitleMaster(TitleCode)

);

-- Function to transform and load 2020 data

CREATE OR REPLACE FUNCTION transform\_load\_2020()

RETURNS void LANGUAGE plpgsql AS $$

BEGIN

INSERT INTO NYCPayroll\_2020

SELECT \* FROM NYCPayroll\_2020\_staging;

DELETE FROM NYCPayroll\_2020\_staging; -- Clear staging table after loading

END;

$$;

-- Function to transform and load 2021 data

CREATE OR REPLACE FUNCTION transform\_load\_2021()

RETURNS void LANGUAGE plpgsql AS $$

BEGIN

INSERT INTO NYCPayroll\_2021

SELECT \* FROM NYCPayroll\_2021\_staging;

DELETE FROM NYCPayroll\_2021\_staging; -- Clear staging table after loading

END;

$$;

Created a public user named account\_manager with the following login details and select priviledges.

Username = account\_manager

Password = ‘123456789’

CREATE ROLE account\_manager WITH

LOGIN

NOSUPERUSER

NOCREATEDB

NOCREATEROLE

INHERIT

REPLICATION

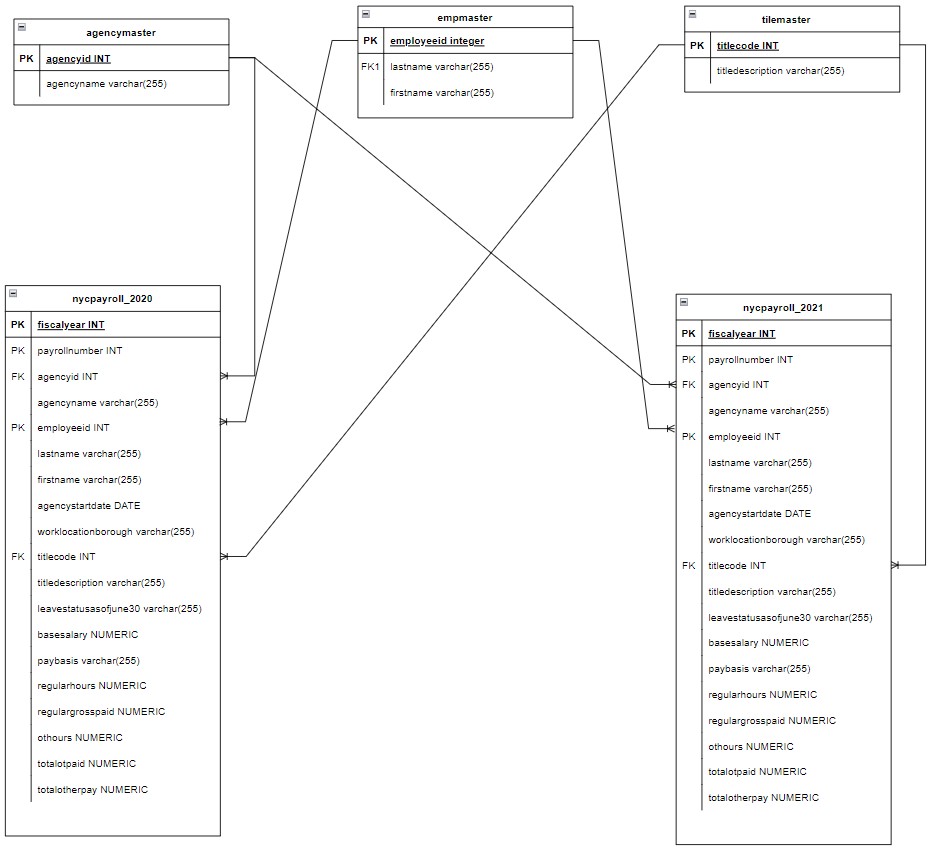
CONNECTION LIMIT -1

PASSWORD 'xxxxxx';

GRANT INSERT, SELECT, REFERENCES, TRIGGER, UPDATE ON TABLE public.agencymaster TO account\_manager;

GRANT INSERT, SELECT, REFERENCES, TRIGGER, UPDATE ON TABLE public.empmaster TO account\_manager;

GRANT INSERT, SELECT, REFERENCES, TRIGGER, UPDATE ON TABLE public.titlemaster TO account\_manager;



AUTOMATION OF ETL ON WINDOWS WITH POSTGRESQL AND TASK SCHEDULER

SQL Functions for Data Transformation

To facilitate the ETL process, necessary SQL functions were created to transform and load data from staging tables to the final tables. The functions handle the insertion of data into the final tables and clear the staging tables once the data is successfully transferred. Below are the functions created:

sql

Copy code

-- Function to transform and load 2020 data

CREATE OR REPLACE FUNCTION transform\_load\_2020()

RETURNS void LANGUAGE plpgsql AS $$

BEGIN

INSERT INTO NYCPayroll\_2020

SELECT \* FROM NYCPayroll\_2020\_staging;

DELETE FROM NYCPayroll\_2020\_staging; -- Clear staging table after loading

END;

$$;

-- Function to transform and load 2021 data

CREATE OR REPLACE FUNCTION transform\_load\_2021()

RETURNS void LANGUAGE plpgsql AS $$

BEGIN

INSERT INTO NYCPayroll\_2021

SELECT \* FROM NYCPayroll\_2021\_staging;

DELETE FROM NYCPayroll\_2021\_staging; -- Clear staging table after loading

END;

$$;

These functions ensure that the data is accurately moved from the staging tables to the final tables, ready for analysis and reporting.

Python Script for ETL Execution

A Python script was developed to call these SQL functions. The script connects to the PostgreSQL database, executes the transformation functions, and commits the changes. This script is saved in a directory for easy access and reuse. Below is the content of the Python script (etl\_script.py):

python

Copy code

import psycopg2

def run\_etl():

# Connect to your PostgreSQL database

conn = psycopg2.connect(

host="your\_host",

database="your\_database",

user="your\_user",

password="your\_password"

)

cursor = conn.cursor()

# Call the transformation functions

cursor.execute("CALL transform\_load\_2020();")

cursor.execute("CALL transform\_load\_2021();")

# Commit the transaction

conn.commit()

# Close the cursor and connection

cursor.close()

conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_etl()

Scheduling with Windows Task Scheduler

To automate the execution of the Python script, Windows Task Scheduler was utilized. This ensures that the script runs at a specified time each day without manual intervention.

Create a New Task: Open Task Scheduler and create a new task named "ETL Script".

Set the Trigger: Configure the task to trigger daily at a specific time (e.g., midnight).

Set the Action: Define the action to start a program and specify the path to the Python executable and the script:

Program/script: C:\Path\To\Python\python.exe

Add arguments: C:\Path\To\Your\Script\etl\_script.py

Save the Task: Ensure the task is saved and enabled to run as per the schedule.

Data Protection and Compliance

Throughout the ETL process, all data protection regulations were strictly followed. Privacy of data was maintained in compliance with GDPR regulations. This included secure handling of personal information and ensuring that all data processing activities were transparent and lawful.

By automating the ETL process using PostgreSQL and Windows Task Scheduler, the NYC Payroll data warehouse is efficiently updated, ensuring data accuracy and timeliness. This automation reduces manual effort, minimizes errors, and adheres to data protection standards, making it a robust solution for managing large

Data Quality Assurance

To ensure the integrity and quality of our data, we've implemented robust data validation checks throughout the ETL (Extract, Transform, Load) process. These checks meticulously validate the data against predefined rules, including data types, constraints, and business logic.

During the execution of the ETL pipeline, we meticulously handle errors and exceptions that may arise, ensuring smooth operation even in the face of unexpected issues. Our error handling mechanisms are designed to gracefully manage any errors encountered, allowing the pipeline to continue processing data without disruption.

In addition, we've implemented PostgreSQL-specific troubleshooting mechanisms to facilitate error analysis. These mechanisms provide detailed information about the execution of the ETL process, including any errors encountered, data transformations applied, and successful data loads. By maintaining comprehensive records, we ensure transparency and facilitate efficient debugging when necessary.

User Access Control

To ensure secure and controlled access to our data, we've implemented a comprehensive system of roles and privileges within PostgreSQL. This system allows us to manage user permissions effectively, granting appropriate access levels to different types of users. As part of this strategy, we established a public user role with restricted access to sensitive data. This role is carefully configured to provide the necessary access to non-sensitive information while safeguarding confidential data from unauthorized access. Furthermore, we have enabled public access to specific datasets, ensuring that essential information is available to the public. This approach balances transparency with security, allowing us to share valuable insights with the community while maintaining stringent data protection standards.