ENSF607 Assignment 3

Balkarn Gill - 30202219 - balkarn.gill1@ucalgary.ca
Yajur Vashist - 30200252
Satchytan Karalasingham - 30222555
Momin Muhammad - 30033100

SQL Scripts for table

createdaatabase.sql:

```
CREATE DATABASE ServiceTickets;
USE ServiceTickets;

CREATE TABLE EventActivity (
  ID INT AUTO_INCREMENT PRIMARY KEY,
  Activityname VARCHAR(20)
);

CREATE TABLE EventOrigin (
   ID INT AUTO_INCREMENT PRIMARY KEY,
   Origin VARCHAR(20)
);

CREATE TABLE EventStatus (
   ID INT AUTO_INCREMENT PRIMARY KEY,
   Status VARCHAR(20)
);

CREATE TABLE EventClass (
   ID INT AUTO_INCREMENT PRIMARY KEY,
   Class VARCHAR(20)
);

CREATE TABLE EventLog (
   ID INT AUTO_INCREMENT PRIMARY KEY,
   CaseID VARCHAR(20) UNIQUE,
   Activity VARCHAR(20),
   Urgency VARCHAR(1),
   Impact VARCHAR(1),
   Priority VARCHAR(1),
   StartDate DATE,
   EndDate DATE,
   TicketStatus VARCHAR(20),
   UpdateDateTime DATETIME,
   Duration INT,
   Origin VARCHAR(20),
   Class VARCHAR(20)
);
```

authentication.sql

```
CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';
GRANT ALL PRIVILEGES ON servicetickets.* TO 'myuser'@'localhost';
FLUSH PRIVILEGES;
```

Source code for generator program

generator.py:

```
import mysql.connector
from mysql.connector import Error
import random
from datetime import datetime, timedelta

# Sample data from your database tables
event_activities = ['Design', 'Construction', 'Test', 'Password Reset']
event_origins = ['Joe S.', 'Bill B.', 'George E.', 'Achmed M.', 'Rona E.']
event_statuses = ['Open', 'On Hold', 'In Process', 'Deployed', 'Deployed Failed']
event_classes = ['Change', 'Incident', 'Problem', 'SR']

# Input parameters
num_tickets = 100
time_window_start = datetime(2023, 1, 1)
time_window_end = datetime(2023, 6, 30)

unique_random_integers = random.sample(range(1, num_tickets + 1), num_tickets)
i = 0

# Function to generate a random datetime within the specified time window
def random_datetime(start, end):
    return start + timedelta(
        seconds=random.randint(0, int((end - start).total_seconds()))
    )

# Function to calculate priority from urgency and impact
def calculate_priority(urgency, impact):
    if urgency == 'H' and impact == 'H':
        return 'H'
    elif urgency == 'H' and impact == 'M':
        return 'M'
    elif urgency == 'M' and impact == 'H':
        return 'M'
    elif urgency == 'H' and impact == 'L':
        return 'M'
    elif urgency == 'L' and impact == 'H':
        return 'M'
    else:
        return 'L'

# Function to insert randomly generated ticket into SQL database
def insert_ticket(case_id, activity, origin, status, ticket_class, start_date,
end_date, urgency, impact, priority, duration, update_datetime):
```

```
    # Define SQL statements for each table
    sql_event_log = "INSERT INTO EventLog (Caseid, Activity, Origin, TicketStatus,
Class, StartDate, EndDate, Urgency, Impact, Priority, Duration, UpdateDateTime) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
    sql_event_activity = "INSERT INTO EventActivity (Activityname) VALUES (%s)"
    sql_event_origin = "INSERT INTO EventOrigin (Origin) VALUES (%s)"
    sql_event_status = "INSERT INTO EventStatus (Status) VALUES (%s)"
    sql_event_class = "INSERT INTO EventClass (Class) VALUES (%s)"

    values_event_log = (case_id, activity, origin, status, ticket_class, start_date,
end_date, urgency, impact, priority, duration, update_datetime)
    values_event_activity = (activity,)
    values_event_origin = (origin,)
    values_event_status = (status,)
    values_event_class = (ticket_class,)

    # Start a transaction
    connection.start_transaction()

    try:
        # Insert into EventLog
        cursor.execute(sql_event_log, values_event_log)

        # Insert into EventActivity
        cursor.execute(sql_event_activity, values_event_activity)

        # Insert into EventOrigin
        cursor.execute(sql_event_origin, values_event_origin)

        # Insert into EventStatus
        cursor.execute(sql_event_status, values_event_status)

        # Insert into EventClass
        cursor.execute(sql_event_class, values_event_class)

        # Commit the transaction
        connection.commit()
    except Exception as e:
        # Rollback the transaction if an error occurs
        print("Error:", e)
        connection.rollback()

    connection.commit()
```

```
db_config = {
    'host': 'localhost',
    'user': 'myuser',
    'password': 'mypassword',
    'database': 'ServiceTickets'
}

# Establish a connection to the MySQL database
connection = mysql.connector.connect(**db_config)
cursor = connection.cursor()

# Generate and print tickets
for _ in range(num_tickets):
    activity = random.choice(event_activities)
    origin = random.choice(event_origins)
    status = random.choice(event_statuses)
    event_class = random.choice(event_classes)
    start_date = random_datetime(time_window_start, time_window_end)
    end_date = random_datetime(start_date, time_window_end)
    update_datetime = random_datetime(start_date, end_date)

    # Calculate urgency, impact, and priority (you can define your own logic here)
    urgency = random.choice(['H', 'M', 'L'])  # High, Medium, Low
    impact = random.choice(['H', 'M', 'L'])   # High, Medium, Low
    priority = calculate_priority(urgency, impact)

    id = unique_random_integers[i]
    i += 1
    case_id = f'CS_{id:05d}'  # Unique Case ID
    # case_id = f'CS_{random.randint(1, 99999):05d}'  # Unique Case ID

    # Print or save the generated ticket data
    insert_ticket(case_id, activity, origin, status, event_class, start_date,
end_date, urgency, impact, priority, int((end_date - start_date).total_seconds()),
update_datetime)

cursor.close()
connection.close()
```