INFORME CARGA DE DATOS

Para la carga de datos se escogió la metodología de hacer un script en Python el cual generara las sentencias SQL que contenían los datos que se iban a meter dentro de la base de datos de Oracle. El script de Python se escogió debido a que podemos tener control sobre los datos que generamos y la consistencia entre los datos que generamos, esto debido a que hay varias reglas de negocio que deben seguir los datos al entrar dentro de la base de datos.

El proceso para generar los datos empieza por crear los datos en tablas que no dependen de otras tablas, es decir, que no tienen FKs. Por ejemplo, se crearon los datos para la tabla de "TipoUsuarios", "TiposH", "Planes" y "Servicios". Estas tablas no dependen de ninguna otra tabla por lo cual podemos crear los datos iniciales de forma manual y meterlos dentro de la base de datos. De aquí en adelante, la información que se creó para el resto de las tablas fue de manera pseudoaleatorio. Por ejemplo, se creó un numero arbitrario de habitaciones escogiendo aleatoriamente que tipo de habitación iban a hacer asignándoles una llave aleatoria que estuviera en "TiposH". Para empleados y usuarios se siguió la misma metodología, pero ateniéndonos a las reglas de negocios como que un usuario solo puede ser cliente mientras un empleado puede tener otro rol dentro del hotel. Para la tabla de "PlanesServicios" se asignaron los servicios correspondientes al plan y algunos más.

Por otro lado, para crear las reservas del hotel, se cogieron llaves primarias de las tablas de "Habitaciones", "Usuarios" y "Planes" para crear aleatoriamente datos sobre reservas hechas en el hotel, pero se debieron seguir algunas restricciones. Fue necesario asegurarse de que la fecha de salida registrada a una reserva fuera mayor a su fecha de entrada para evitar inconsistencias lógicas dentro del sistema. Además, fue necesario asegurarnos de que una habitación no tuviera dos reservas que sobrelapen en la estadía. Esto igual se verifica en la aplicación, pero cuando se meten datos generados pseudoaleatoriamente es necesario revisar porque los datos no se están metiendo por la aplicación y para evitar errores en la base de datos. Para tablas como "Consumos" y "ReservasServicios" se siguió un proceso similar al de la aleatoriedad, pero de igual manera fue necesario implementar algunos chequeos. Fue necesario revisar que una reserva de servicio asignada a una reserva dentro del hotel estuviera hecha dentro de las fechas de estadía establecidas por la reserva, esto con el fin de que la información dentro de la base de datos fuera consistente.

El proceso de generación de datos con Python facilito la generación de las sentencias SQL para después ponerlas a correr en batch para subirlas a la base de datos. Para garantizar la consistencia de los datos, como por ejemplo que los datos que tuvieras FKs tuvieran una FK valida, localmente se guardaron las PKs de las tablas necesarias para poder asignar a las tuplas que lo necesitaran valores existentes sin necesidad de pedírselo a la base de datos. Además de las PKs, también se guardó otra información relevante que facilito la creación de datos. La decisión de guardar información localmente se hizo con el propósito de evitar peticiones a la base de datos mientras se crean los datos. Aunque las peticiones a la base de datos no tomen más de unos segundos por mucho, para la creación de casi un millón de datos el tiempo necesario hubiera sido mayor a un día por lo cual se decidió guardar toda la información localmente y después subirla toda en batch a la base de datos.

Cantidad de datos:

Tipos de habitaciones = 5
Tipos de usuarios = 5
Servicios = 23
Planes = 4
Servicios en planes = 40
Habitaciones = 10,000
Empleados = 50,000
Usuarios = 120,000
Consumos = 100,000
Reservas = 300,000
Reservas de servicios = 200,000

Total = 780,077 tuplas en la base de datos

Fragmentos de código

En el siguiente fragmento de código se evidencia como a cada una de las habitaciones se les asigna un tipo de habitación aleatoriamente de las PKs de tipo de habitación y se guarda su sentencia para subir a la base de datos.

```
# Habitaciones

habitaciones_pk = range(0, num_habitaciones)

with open("habitaciones.txt", "w") as file:
    for id_habitacion in habitaciones_pk:
        file.write(f"INSERT INTO habitaciones VALUES ({id_habitacion}, '{random.choice(tipos_habitacion_pk)}');\n")
```

En el siguiente fragmento de código se evidencia como a cada usuario (tabla “Usuarios”) solo se le puede asignar el rol de “Cliente” y además que su tipo de documento solo puede ser CC, TI o PASAPORTE. Subsecuentemente se procede a guardar la sentencia con sus respectivos datos.

```
# Usuarios

usuarios_pk = range(0, num_usuarios)

with open("usuarios.txt", "w") as file:
    tipo_documento = ["CC", "PASAPORTE", "TI"]
    tipo_usuario = "Cliente"
    for id_usuario in usuarios_pk:
        name = random.choice(names)
        file.write(f"INSERT INTO usuarios VALUES ({id_usuario}, '{random.choice(tipo_documento)}', '{name}', '{name.lower()}@gmail.com', '{tipo_usuario}');\n")
```

En el siguiente fragmento de código se puede ver como se crean datos para una fecha de entrada y una fecha de salida que se asignara a una reserva. El código revisa que la fecha de salida no sea

menor al de entrada ya que estoy no tiene sentido lógico con las reglas de negocio. Si se determina que la fecha de salida es mayor que la fecha de entrada el código sale del while y sigue con sus tareas. En caso de que la fecha de salida sea menor que la fecha de entrada entonces se repite el proceso hasta que ya no sea el caso.

```
while True:
    entry_year = random.randint(2020, 2023)
    entry_month = random.randint(1, 12)
    entry_day = random.randint(1, 28)

    exit_year = random.randint(entry_year, entry_year + 2)
    exit_month = random.randint(1, 12)
    exit_day = random.randint(1, 28)

    if (exit_year < entry_year):
        continue

    if (exit_year == entry_year and exit_month < entry_month):
        continue

    if (exit_year == entry_year and exit_month == entry_month and exit_day < entry_day):
        continue

    break
```

En el siguiente fragmento de código se puede ver como se asegura que una reserva de un servicio hecha asignada a una reserva este dentro de las fechas de entrada y de salida de la reserva. Para lograr esto se coge un día aleatorio entre las fechas de entrada y salida de la reserva y se le asigna a la reserva de servicio.

```
with open("reservas_servicios.txt", "w") as file:
    for id_reserva_servicio in reservas_servicios_pk:
        reserva = random.choice(reservas_pk)
        rango_fecha = reservas_fechas[reserva]
        fecha_entrada = rango_fecha[0]
        fecha_salida = rango_fecha[1]

        diferencia = (fecha_salida[0] - fecha_entrada[0]) * 365 + (fecha_salida[1] - fecha_entrada[1]) * 30 + (fecha_salida[2] - fecha_entrada[2])

        servicio_dias = random.randint(0, diferencia)

        extra_years = servicio_dias // 365
        extra_months = (servicio_dias % 365) // 30
        extra_days = (servicio_dias % 365) % 30

        fecha_servicio = (fecha_entrada[0] + extra_years, fecha_entrada[1] + extra_months, fecha_entrada[2] + extra_days)

        if (fecha_servicio[2] > 28):
            fecha_servicio = (fecha_servicio[0], fecha_servicio[1] + 1, fecha_servicio[2] % 28 if fecha_servicio[2] % 28 > 0 else 1)

        if (fecha_servicio[1] > 12):
            fecha_servicio = (fecha_servicio[0] + 1, fecha_servicio[1] % 12 if fecha_servicio[1] % 12 > 0 else 1, fecha_servicio[2])

        fecha = f"TO_DATE('{fecha_servicio[0]}-{fecha_servicio[1]}-{fecha_servicio[2]}', 'YYYY-MM-DD')"
        servicio = random.choice(servicios_pk)
        empleado = random.choice(empleados_pk)
        file.write(f"INSERT INTO reservasservicios VALUES ({id_reserva_servicio}, {fecha}, {random.randint(1, 7)}, {servicio}, {reserva}, {empleado});\n")
```