```
import random
from itertools import product

population_factor = 300

def combinator(first_names, last_names, separator = " ", prefix="", sufix=""):
    return  map("".join,product([prefix], first_names, [separator], last_names, [sufix]))+\
            map("".join,product([prefix], last_names, [separator], first_names, [sufix]))+\
            map("".join,product([prefix], first_names, [separator], map(str,range(population_factor)),
[sufix]))+\
            map("".join,product([prefix], map(str,range(population_factor)), [separator], last_names,
[sufix]))

def generate_numbers(number, start=1, type="int"):
    numbers = []
    if type == "float":
        start = float(start)
    for i in range(number):
        numbers.append(start)
        start += 1
    return numbers

def generate_people():
    first_names = ["Joao", "Maria", "Dedeu", "Diguliu", "Michael", "Daniela", "Kratus", "Chuck",
"Johnny", "Ze", "Conf", "Joca"]
    last_names = ["Silva", "Oliveira", "Alcantra", "Souto", "Jackson", "Norris", "Santos", "Neto",
"Braga", "Correa", "Depp", "Jordan" ]

    people_names = combinator(first_names, last_names)
    people_emails = combinator(map(str.lower,first_names), map(str.lower,last_names), "@", "", ".com")
    people_addresses = combinator(first_names, last_names, " - ", "Street ")
    people_cpfs = generate_numbers(len(people_names),345746345)
    people_cpfs = map(str, people_cpfs)
    people_ages = []
    people_genders = []
    people_birth_dates = []
    people_phones1 = []
    people_phones2 = []
    for i in range(len(people_names)):
        people_genders.append(random.choice(["female","male"]))
        year = random.randint(1930,2010)
        people_ages.append(2010-year)
        people_birth_dates.append(str(year) + "-" + str(random.randint(1,12)) + "-" + str
(random.randint(1,28)))
        people_phones1.append("".join(str(random.randint(0,9)) for x in range(8)))
        people_phones2.append("".join(str(random.randint(0,9)) for x in range(8)))

    return zip(*[people_cpfs, people_names, people_addresses, people_emails,\
                 people_genders, people_birth_dates, people_ages, people_phones1,\
                 people_phones2])

def generate_clients(people, percent):
    clients = []
    for i in range(int(len(people)*percent)):
        login = people[i][1].lower().replace(' ','_')
        clients.append([login, people[i][0], "dfasdj"])
    return clients

def generate_members(people, percent):
    members = []
    for i in range(int(len(people)*percent)):
        members.append([i+1, people[len(people)-1-i][0]])
    return members

def generate_bands():
    first_names = ["The", "Arctic", "Pearl", "Led", "Avenged"]
    last_names = ["Zeppelin", "Monkeys", "Sevenfold", "Jam", "Beatles"]

    band_names = combinator(first_names, last_names)
    logins = combinator(map(str.lower,first_names), map(str.lower,last_names),"_")
    passwords = generate_numbers(len(logins))
    passwords = map(str, passwords)
```

```
    first_style_names = ["Hard", "Black", "Emo", "Heavy", "Folk"]
    last_style_names = ["Metal", "Core", "Punk", "Grunge", "Rock"]
    style_names = combinator(first_style_names, last_style_names)

    return zip(*[logins, band_names, style_names, ["www.all-bands.com"]*len(logins), passwords])

def generate_band_has_members(bands, members):
    relationships = []
    for member in members:
        relationships.append([random.choice(bands)[0], member[0]])
    return relationships

def generate_equipments():
    description = ["Semi-Acoustic", "Eletric", "Acoustic", "Tuned", "Golden"]
    models = ["Les Paul", "Stratocaster", "Flying V", "SG", "Telecaster"]
    types = ["Guitar", "Bass", "Drums", "Microphone", "Amplificator"]

    equipment_types = combinator(types, models)
    equipment_ids = generate_numbers(len(equipment_types))
    internal_prices = generate_numbers(len(equipment_types),20,"float")
    external_prices = generate_numbers(len(equipment_types),50,"float")

    return zip(*[equipment_ids, ["Model"]*len(equipment_types), equipment_types, internal_prices,
external_prices])

def generate_services():
    first_names = ["Record", "Sell", "Rent", "Practice", "Make"]
    last_names = ["Albuns", "Instruments", "Equipments", "Rooms", "Dvds"]

    service_names = combinator(first_names, last_names)
    service_ids = generate_numbers(len(service_names))
    prices = generate_numbers(len(service_names),30,"float")

    return zip(*[service_ids, service_names, prices, ["Description"]*len(service_names)])

def generate_agendas(services, bands, n):
    agendas = []
    year = 2010
    for i in range(n):
        agendas.append([i+1,random.choice(services)[0], random.choice(bands)[0],
                        str(year) + "-" + str(random.randint(1,12)) + "-" + str(random.randint(1,28)),\
                        str(random.randint(0,23)) + ":" + str(random.randint(0,59)) + ":" + str
(random.randint(0,59)),\
                        random.randint(1,5), random.randint(1,3), random.choice(["reserved",
"canceled", "done"])])
        year+=1
    return agendas

def generate_agenda_has_equipments(agendas, equipments):
    relationships = []
    for agenda in agendas:
        for i in range(random.randint(0,3)):
            relationships.append([agenda[0], random.choice(equipments)[0], random.randint(1,5), agenda
[4]])
    return relationships

def generate_client_rents_equipments(equipments, clients):
    relationships = []
    i = 1
    year = 2010
    for client in clients:
        for x in range(random.randint(0,2)):
            equipment = random.choice(equipments)
            duration = random.randint(1,5)
            relationships.append([i, equipment[0], client[0], duration, \
                                  str(year) + "-" + str(random.randint(1,12)) + "-" + str
(random.randint(1,28)),\
                                  str(random.randint(0,23)) + ":" + str(random.randint(0,59)) + ":" +
str(random.randint(0,59)),\
                                  equipment[4]*duration, random.choice(["reserved", "canceled",
"done"])])
```

```
            i += 1
            year += 1
    return relationships


def generate_insert_sql(attributes, entities, table_name):
    colums = ""
    sql_code = ""
    for attribute in attributes:
        colums += ',' + attribute
    colums = colums[1:]
    for entity in entities:
        values = ""
        for att in entity:
            if type(att).__name__ != "int" and type(att).__name__ != "float":
                att = "\'{0}\'".format(att)
            values += ',' + str(att)
        values = values[1:]
        sql_code += "INSERT INTO %(table_name)s (%(colums)s)\nVALUES (%(values)s);" % \
                {   'table_name'    : table_name,\
                    'colums'        : colums,\
                    'values'        : values}
        sql_code += "\n\n"
    return sql_code


person_attributes                   = ["Cpf", "Name", "Address", "Email", "Gender", "Birth_Date",
"Age", "Phone1", "Phone2"]
client_attributes                   = ["Login", "Person_Cpf", "Pass"]
member_attributes                   = ["Member_ID", "Person_Cpf"]
band_attributes                     = ["Login", "Name", "Style", "HomePage", "Pass"]
band_has_member_attributes          = ["Band_Login", "Member_ID"]
equipment_attributes                = ["Equipment_ID", "Model", "Equipment_Type", "InternalPrice",
"ExternalPrice"]
service_attributes                  = ["Service_ID", "Name", "Price", "Description"]
agenda_attributes                   = ["Agenda_ID", "Service_ID", "Band_Login", "Date", "Time",
"Duration", "Room", "Status"]
agenda_has_equipments_attributes    = ["Agenda_ID", "Equipment_ID", "Duration", "Time"]
client_rents_equipments_attributes  = ["Rent_ID", "Equipment_ID", "Client_Login", "Duration", "Date",
"Time", "Total_Price", "Status"]

people = generate_people()
print "People -> " + str(len(people))

clients = generate_clients(people,0.5)
print "Clients -> " + str(len(clients))

members = generate_members(people,0.5)
print "Members -> " + str(len(members))

bands = generate_bands()
print "Bands -> " + str(len(bands))

band_has_members = generate_band_has_members(bands, members)
print "Band has Member (should be the same as Members) -> " + str(len(band_has_members))

equipments = generate_equipments()
print "Equipments -> " + str(len(equipments))

services = generate_services()
print "Services -> " + str(len(services))

agendas = generate_agendas(services, bands, 5000)
print "Agendas -> " + str(len(agendas))

agenda_has_equipments = generate_agenda_has_equipments(agendas, equipments)
print "Agenda has Equipments -> " + str(len(agenda_has_equipments))

client_rents_equipments = generate_client_rents_equipments(equipments, clients)
print "Client rents Equipments -> " + str(len(client_rents_equipments))
```

```
sql_insert_code = generate_insert_sql(person_attributes, people, "Person")
sql_insert_code += generate_insert_sql(band_attributes, bands, "Band")
sql_insert_code += generate_insert_sql(equipment_attributes, equipments, "Equipment")
sql_insert_code += generate_insert_sql(service_attributes, services, "Service")
sql_insert_code += generate_insert_sql(client_attributes, clients, "Client")
sql_insert_code += generate_insert_sql(member_attributes, members, "Member")
sql_insert_code += generate_insert_sql(band_has_member_attributes, band_has_members, "Band_has_Member")
sql_insert_code += generate_insert_sql(agenda_attributes, agendas, "Agenda")
sql_insert_code += generate_insert_sql(agenda_has_equipments_attributes, agenda_has_equipments,
"Agenda_has_Equipment")
sql_insert_code += generate_insert_sql(client_rents_equipments_attributes, client_rents_equipments,
"Client_rents_Equipment")
file = open("sql_population_code.sql","w")
file.write(sql_insert_code)
file.close()
```