Real Estate Company Database

Rita Kurban

CS162

This project is an implementation of a database system for a large franchised real estate company in SQLAlchemy.

1. The company has many offices located all over the country.
2. Each office is responsible for selling houses in a particular area.
3. An estate agent can be associated with one or more offices.

Data Normalization

The database has seven tables: Agents, Buyers, Sellers, Offices, Houses, Sales, Commissions.

Sales and Commissions are populated from the entries from other tables with the help of the sell_house().

I followed the database normalization rules to make sure that there is no data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies.

First Normal Form

Columns have only one attribute per cell, and each column has a unique name: 'ID,' 'price,' 'email,' etc.

I restricted all values in a single column to have the same type: Text, Integer, or DateTime.

The order of the data entries doesn't matter.

Second Normal Form

I made sure that all columns are only dependent on the primary key which is unique for every table. This property is called partial dependency where an attribute in a table depends on only a part of the primary key and not on the whole key. In my case, I didn't have any composite keys. Therefore, the second form was not violated.

Third Normal Form

I made sure that there is no transitive dependency. For example, I didn't include agent names to the Houses table because they would depend on the agent ID. Instead, I created a separate table called Agents where I store all the information about the agents.

Fourth Normal Form

For a dependency A → B, if for a single value of A, multiple values of B exists, then the table may have multi-valued dependency. For example, if a table has students who take different courses and also have various hobbies that are independent of each other, such an issue can arise. In my database, I made sure to separate all tables to make sure that this is not the case.

Indices

An index is an ordered data structure which can be used to find an entry or its primary key in O(log n) time. I didn't explicitly add indices to my tables since SQLite automatically maintains an ordered list of the data within the index's columns as well as their records' primary key values.

The joins that I used in the queries are based on foreign key constraints that are specified in the tables using the ForeignKey keyword.

SQLite is a highly optimized database (https://www.sqlite.org/optoverview.html (https://www.sqlite.org/optoverview.html)). Therefore, I don't think that adding any other type of indices would have a significant effect on the performance.

Transactions

A transaction is a set of tasks put into a single execution unit. It begins with a specific task and ends when all tasks are completed. If any of the tasks fail, the entire transaction fails. Therefore, a transaction has only two results: success or failure.

All transactions should satisfy the ACID properties. Atomicity guarantees that each transaction is treated as a single "unit," which either succeeds or fails. Consistency ensures that a transaction can only bring the database from one valid state to another. Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure — source: Wikipedia.

To implement transactions, I used sessionmaker of SQLAlchemy. All the tasks are added with the help of session.add() and executed using session.commit(). I do it to add entries to all the tables to ensure that no information is getting corrupted or lost.

In [1]:

```
# Install all necessary packages
import sqlalchemy
from sqlalchemy import create_engine, Column, Text, Integer, ForeignKey, DateTim
e, DECIMAL, VARCHAR, func, and_, text, case, Date, cast
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker
import time
from datetime import date
from sqlalchemy.ext.hybrid import hybrid_property, hybrid_method
import pandas as pd
import numpy as np
from datetime import datetime
```

In [2]:

```
# Create and connect the engine

engine = create_engine('sqlite:///database.db')
engine.connect()

Base = declarative_base()
```

In [3]:

```
# Build tables

class Agents(Base):
    __tablename__ = 'agents'
    ID = Column(Integer, primary_key=True, index=True)
    name = Column(Text)
    email = Column(Text)

    def __repr__(self):
        return "<Agent(id={0}, name={1} email={2}>".format(self.ID, self.name, s
elf.email)

class Buyers(Base):
    __tablename__ = 'buyers'
    ID = Column(Integer, primary_key=True, index=True)
    name = Column(Text)
    email = Column(Text)

    def __repr__(self):
        return "<Buyer(id={0}, name={1}, email={2}>".format(self.ID, self.name,
self.email)

class Sellers(Base):
    __tablename__ = 'sellers'
    ID = Column(Integer, primary_key=True, index=True)
    name = Column(Text)
    email = Column(Text)

    def __repr__(self):
        return "<Seller(id={0}, name={1}, email={2}>".format(self.ID,self.name,
self.email)

class Offices(Base):
    __tablename__ = 'offices'
    ID = Column(Integer, primary_key=True, index=True)
    name = Column(Text)

    def __repr__(self):
        return "<Seller(id={0}, name={1}>".format(self.ID, self.name)


class Houses(Base):
    __tablename__ = 'houses'
    ID = Column(Integer, primary_key = True, index=True)
    seller_id = Column(Integer, ForeignKey('sellers.ID'))
    n_bedrooms = Column(Integer)
    n_bathrooms = Column(Integer)
    price = Column(Integer)
    zip_code = Column(Integer) #should it be ID?
    date = Column(DateTime)
    agent_id = Column(Integer, ForeignKey('agents.ID'))
    office_id = Column(Integer, ForeignKey('offices.ID')) #potentially combine w
ith zips
    status =Column(Integer)



    def __repr__(self):
        return "<House(id={0}, seller_id={1}, n_bedrooms={2},\
```

```
        n_bathrooms={3}, price={4}, zip_code={5}, date={6}, \
        agent_id={7}, office_id={8}, status={9})>".format(self.ID, self.seller_i
d, self.n_bedrooms,
                                                         self.n_bathrooms, self
.price, self.zip_code,
                                                         self.date, self.agent_
id, self.office_id, self.status)

class Sales(Base):
    __tablename__ = 'sales'
    ID = Column(Integer, primary_key = True, index=True)
    house_id = Column(Integer, ForeignKey('houses.ID'))
    buyer_id = Column(Integer, ForeignKey('buyers.ID'))
    sale_price = Column(Integer)
    date = Column(DateTime)

    def __repr__(self):
        return "<Sale(id={0}, house_id={1}, buyer_id={2}, sale_price={3}, date=
{4}>".format(self.ID,

self.house_id,

self.buyer_id,

self.sale_price,

self.date)



class Commissions(Base):
    __tablename__ = 'commissions'
    ID = Column(Integer, primary_key = True, index=True)
    sale_id = Column(Integer, ForeignKey('sales.ID'))
    amount = Column(Integer)
    agent_id = Column(Text)


    def __repr__(self):
        return "<Commissions(id = {0}, sale_id={1}, amount={2}, agent_id={3}>".f
ormat(self.ID,

self.sale_id,

self.amount,

self.agent_id)
```

In [4]:

```
# Generate entries

office_keys = ['ID', 'name']
office_values = [
    [1,"California"],
    [2,"New York"],
    [3,"Massachusetts"],
    [4,"Washington"],
    [5,"Nevada"]]


house_keys = ['ID', 'seller_id', 'n_bedrooms', 'n_bathrooms', 'price', 'zip_cod
e',
              'date', 'agent_id', 'office_id', 'status']
house_values = [
    [1,1,2,1,200000,94102,date(2018,4,25), 1,1,0],
    [2,2,1,1,100000,94102,date(2018,5,20), 2,1,0],
    [3,3,4,3,700000,94102,date(2018,8,18), 4,1,0],
    [4,4,4,3,600000,94103,date(2018,8,16), 5,2,0],
    [5,5,3,2,500000,94103,date(2018,9,5), 1,2,0],
    [6,1,2,2,300000,94103,date(2018,11,4), 2,2,0],
    [7,2,5,3,800000,94104,date(2018,2,25), 3,3,0],
    [8,3,1,1,200000,94104,date(2018,6,27), 2,3,0],
    [9,4,2,2,300000,94104,date(2018,7,11), 1,3,0],
    [10,5,3,2,400000,94105,date(2018,9,16), 2,4,0],
    [11,1,4,2,800000,94105,date(2018,10,5), 3,4,0],
    [12,2,2,1,400000,94105,date(2018,11,8), 4,4,0],
    [13,3,1,1,100000,94106,date(2018,12,1), 3,5,0],
    [14,4,3,2,600000,94106,date(2018,1,31), 5,5,0],
    [15,5,4,2,1000000,94106,date(2018,3,18), 5,5,0],
    [16,5,3,1,300000,94104,date(2019,1,27), 1,3,0],
    [17,5,2,1,150000,94104,date(2019,2,18), 2,3,0],
    [18,4,3,3,800000,94102,date(2019,3,4), 3,1,0],
    [19,2,6,3,1100000,94103,date(2019,4,6), 4,2,0],
    [20,3,4,2,700000,94103,date(2019,5,15), 5,2,0],
    [21,3,3,2,500000,94103,date(2019,6,14), 2,2,0],
    [22,1,6,3,700000,94104,date(2019,7,5), 3,3,0]]


agent_keys = ['ID', 'name', 'email']
agent_values = [
    [1,'Gregory Watson','watson@gmail.com'],
    [2,'Alice Gander','gander_alice@gmail.com'],
    [3,'Adam Walferd','walferd_ben@gmail.com'],
    [4,'Denis Cremen','cremen_denis@gmail.com'],
    [5,'Fenni Hoasner','hoasner@gmail.com']]

seller_keys = ['ID', 'name','email']
seller_values = [
    [1,'David Kahn','kahl@gmail.com'],
    [2,'Nancy Dawn','nancy_dawn@gmail.com'],
    [3,'Sherrill Mann','mann_sherrill@gmail.com'],
    [4,'Vera Benster','vera_benster@gmail.com'],
    [5,'Georg Miller','miller@gmail.com']]


buyer_keys = ['ID','name', 'email']
buyer_values = [
    [1,'Slava Kochitz','konchitz@gmail.com'],
```

```
    [2,'Pasha Sevkov','pasha_sevkov@gmail.com'],
    [3,'Jared Smith','jared_smith@gmail.com'],
    [4,'Mary Puffindor','mary_puff@gmail.com'],
    [5,'Kate Chervotkin','chechotkin@gmail.com']]
```

In [5]:

```
# Add entries to the session
Base.metadata.drop_all(bind=engine)

Base.metadata.create_all(bind=engine)

Session = sessionmaker(bind=engine)
session = Session()

keys = [office_keys, house_keys, agent_keys, seller_keys, buyer_keys]
values = [office_values, house_values, agent_values, seller_values, buyer_values
]
tables = [Offices, Houses, Agents, Sellers, Buyers]


def add_entries(keys, values, tables):
    '''
    Add entries to the table by combining keys, values,
    and table names.
    '''
    list_of_dict = []
    for value in values:
        item_dict = dict(zip(keys, value))
        list_of_dict.append(item_dict)

    for data_entry in list_of_dict:
        entry = tables(**data_entry)
        session.add(entry)


for i in range(len(tables)):
    add_entries(keys[i], values[i], tables[i])

session.commit()
session.close()
```

In [6]:

```
def sell_house(house_id, buyer_id, date):
    '''
    Add a transaction to the sales table.
    Update the status of the house from 0 to 1.
    Calculate commissions and populate the Commissions table.
    '''

    # Change the house status
    sold = session.query(Houses).filter(Houses.ID == house_id)
    sold.update({Houses.status: 1})

    # Calculate listing price, commission and total price
    listing_price = session.query(Houses.price).filter(Houses.ID == house_id).fi
rst()[0]
    agent_id = session.query(Houses.agent_id).filter(Houses.ID == house_id).firs
t()[0]
    coef = case([(Houses.price < 100000, 0.01),
           (Houses.price < 200000, 0.075),
           (Houses.price < 500000, 0.06),
           (Houses.price < 1000000, 0.05),
           (Houses.price >= 1000000, 0.04),])
    commission = session.query(Houses.price*coef).filter(Houses.ID == house_id).
first()[0]
    sale_price = listing_price + commission

    # Add Sales entry
    session.add(Sales(
        house_id = house_id,
        buyer_id = buyer_id,
        sale_price = sale_price,
        date = date))


    # Add Commissions entry
    sale = session.query(Sales.ID).filter(Houses.ID == house_id).first()[0]

    session.add(Commissions(
        sale_id = sale,
        amount = commission,
        agent_id = agent_id))

    session.commit()
```

In [7]:

```
# Insert new data
sale_values = [
    [1,5,date(2018,12,10)],
    [2,4,date(2018,6,8)],
    [3,3,date(2018,12,17)],
    [4,2,date(2018,11,23)],
    [5,1,date(2019,1,17)],
    [6,5,date(2019,4,8)],
    [7,4,date(2019,3,13)],
    [8,3,date(2018,12,27)],
    [9,2,date(2018,9,1)],
    [10,1,date(2019,4,23)],
    [11,5,date(2018,11,10)],
    [12,4,date(2019,4,10)],
    [13,3,date(2019,3,12)],
    [14,2,date(2018,9,18)],
    [15,1,date(2018,11,16)]]

for value in sale_values:
    sell_house(*value)
```

In [8]:

```
# Query 1: Find the top 5 offices with the most sales for that month.

month = 11
top_offices = session.query(Offices.name,
                            func.sum(Houses.price),
                            func.extract('month', Sales.date)). \
                            join(Houses, and_(Offices.ID == Houses.office_id)).
 \
                            join(Sales, and_(Sales.house_id == Houses.ID)). \
                            group_by(Offices.ID).order_by(func.sum(Houses.price)
. \
                            desc()).filter(func.extract('month', Sales.date) ==
month). \
                            limit(5).statement

top_offices = pd.read_sql(top_offices, session.bind)
top_offices.columns=['Office', 'Sales', 'Month']
top_offices
```

Out[8]:

	Office	Sales	Month
0	Nevada	1000000	11
1	Washington	800000	11
2	New York	600000	11

In [9]:

```
# Query 5: For all houses that were sold that month, calculate the average selli
ng price

# Change the month variable above and run both cells to see results for other mon
ths.
if len(top_offices) >= 1:
    print("The average selling price for the {0}th month is ${1}".format(month,
int(top_offices.mean()[0])))
else:
    print("No houses were sold in the {0}th month".format(month))
```

```
The average selling price for the 11th month is $800000
```

In [10]:

```
# Query 2: Find the top 5 estate agents who have sold the most (include their co
ntact
# details and their sales details so that it is easy contact them and congratula
te them).

# Names and contact info
top_agents = session.query(Agents.name,
                           Agents.email,
                           func.count(Houses.price),
                           func.sum(Houses.price)). \
                            join(Houses, and_(Agents.ID == Houses.agent_id)). \
                            group_by(Agents.name). \
                            order_by(func.sum(Houses.price). \
                            desc()).limit(5).statement
top_agents = pd.read_sql(top_agents, session.bind)
top_agents.columns=['Agent', 'Email', 'Houses Sold','Sales']
top_agents
```

Out[10]:

	Agent	Email	Houses Sold	Sales
0	Adam Walferd	walferd_ben@gmail.com	5	3200000
1	Fenni Hoasner	hoasner@gmail.com	4	2900000
2	Denis Cremen	cremen_denis@gmail.com	3	2200000
3	Alice Gander	gander_alice@gmail.com	6	1650000
4	Gregory Watson	watson@gmail.com	4	1300000

In [11]:

```
#Sales details
agent = "Fenni Hoasner"

sales= session.query(Sales.ID,
                     Buyers.name,
                     Sellers.name,
                     Sales.date,
                     Houses.price). \
                    join(Houses, and_(Sales.house_id == Houses.ID)). \
                    join(Buyers, and_(Buyers.ID == Sales.buyer_id)). \
                    join(Agents, and_(Houses.agent_id == Agents.ID)). \
                    join(Sellers, and_(Houses.seller_id == Sellers.ID)). \
                    order_by(Sales.date).filter(Agents.name == agent).limit(5).s
tatement
sales = pd.read_sql(sales, session.bind)
sales.columns=['Sale ID', 'Buyer', 'Seller','Date',"Price"]
sales
```

Out[11]:

	Sale ID	Buyer	Seller	Date	Price
0	14	Pasha Sevkov	Vera Benster	2018-09-18	600000
1	15	Slava Kochitz	Georg Miller	2018-11-16	1000000
2	4	Pasha Sevkov	Vera Benster	2018-11-23	600000

In [12]:

```
# Query 3: Calculate the commission that each estate agent must receive and stor
e the results in a separate table.

commissions_per_agent = session.query(Agents.name,
                                      func.sum(Commissions.amount),
                                      func.count(Commissions.amount)). \
                                    join(Commissions, and_(Commissions.agent_id
== Agents.ID)). \
                                    group_by(Agents.name). \
                                    order_by(func.sum(Commissions.amount). \
                                    desc()).statement
commissions_per_agent = pd.read_sql(commissions_per_agent, session.bind)
commissions_per_agent.columns=['Agent', 'Total Commission', 'Number of Sales']
commissions_per_agent
```

Out[12]:

	Agent	Total Commission	Number of Sales
0	Fenni Hoasner	100000	3
1	Adam Walferd	87500	3
2	Alice Gander	61500	4
3	Denis Cremen	59000	2
4	Gregory Watson	55000	3

In [13]:

```
# Query 4: For all houses that were sold that month, calculate the average numbe
r of days that the house was on the market.

month = 4
dates = session.query(Houses.ID, Sales.date, Houses.date).join(Sales, and_(Sales
.house_id == Houses.ID)).all()

n_days = []
for date in dates:
    n_days.append(date[1]-date[2])
avg_days = np.mean(n_days).days

print("Average number of days that the house was on the market is {0}.".format(a
vg_days))
```

```
Average number of days that the house was on the market is 157.
```

In [14]:

```
# Query 6: Find the zip codes with the top 5 average sales prices

zip_codes = session.query(Houses.zip_code,
                          func.sum(Houses.price)). \
                        group_by(Houses.zip_code). \
                        order_by(func.sum(Houses.price). \
                        desc()).limit(5).statement

zip_codes = pd.read_sql(zip_codes, session.bind)
zip_codes.columns=['Zip Code', 'Average Price']
zip_codes
```

Out[14]:

	Zip Code	Average Price
0	94103	3700000
1	94104	2450000
2	94102	1800000
3	94106	1700000
4	94105	1600000

In [15]:

```
# Cleanup
session.close()
Base.metadata.drop_all(bind=engine)
```