Structure of the travel agency database

Supervisor

employee_id – unique identifier of a supervisor – primary key

name – name of the supervisor, has to be separated by a space

address – address of a supervisor, supposedly in the format <Street, Street Number, Zip Code, Cityname>

email – will always be in this format: firstname.lastname@hammertrips.com

salary – cannot be lower than 8000

nationality – assumed to be a string of an existing country

role – will always be set to supervisor by default

relationships

account – one to one relationship with User table

teammembers – one to many relationship with TravelAgent

messages – one to many relationship with Message

```
__tablename__ = "supervisor"
employee_id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(50),nullable=False)
address = db.Column(db.String(100),nullable=False)
email = db.Column(db.String(100),nullable=False, unique=True)
salary = db.Column(db.Integer,nullable=False)
nationality = db.Column(db.String(20),nullable=False)
role = db.Column(db.String(20),nullable=False, default='supervisor')

account = db.relationship('User', backref='Supervisor', uselist=False)
teammembers = db.relationship('TravelAgent', backref='Supervisor')
messages = db.relationship('Message', backref='Supervisor')
```

User

id – unique identifier of this user – primary key

username – username chosen by supervisor, it has to be unique

password hash – password of the user in encoded form

relationships

manager_id – one to one relationship with Supervisor

```
__tablename__ = "user"
id = db.Column(db.Integer, primary_key=True)
username = db.Column(db.String(50), unique=True)
password_hash = db.Column(db.String(100))

manager_id = db.Column(db.Integer, db.ForeignKey('supervisor.employee_id'))
```

Message

message_id – unique identifier of the inbox – primary key

message – type of request, can be either “raise” or “discount”

offer_id – set to 0 if there is no request for a discount, otherwise contains the offer_id

percentage – set to 0 if there is no request for a discount, otherwise contains recommended percentage from the TravelAgent for the decrease of the total price of an offer

relationships

supervisor_id – one to one relationship with Supervisor

agent_id – one to one relationship with TravelAgent

```
__tablename__ = "message"
message_id = db.Column(db.Integer, primary_key=True)
message = db.Column(db.String(50), nullable=False)
offer_id = db.Column(db.Integer, nullable=False, default=0)
percentage = db.Column(db.Integer, nullable=False, default=0)

supervisor_id = db.Column(db.Integer,
db.ForeignKey('supervisor.employee_id'),nullable=False)
agent_id = db.Column(db.Integer,
db.ForeignKey('travel_agent.employee_id'),nullable=False)
```

TravelAgent

employee_id – unique identifier of a travelAgent – primary key

name – name of the supervisor, has to be separated by a space

address – address of a supervisor, supposedly in the format <Street, Street Number, Zip Code, Cityname>

email – will always be in this format: firstname.lastname@hammertrips.com

salary – the minimum by default is set to 2000 and the maximum is 4000

nationality – assumed to be a string of an existing country

role – will always be set to travelAgent by default

relationships

supervisor_id – one to one relationship with Supervisor

stats – one to one relationship with AgentStats

countries – many to many relationship with Country

customers – one to many relationship with Customer

offers – one to many relationship with Offer

messages – one to one relationship with Message

```
__tablename__ = "travel_agent"
employee_id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(50),nullable=False)
address = db.Column(db.String(100),nullable=False)
email = db.Column(db.String(100),nullable=False, unique=True)
salary = db.Column(db.Integer,nullable=False)
nationality = db.Column(db.String(20),nullable=False)
role = db.Column(db.String(20),nullable=False, default='travelAgent')

supervisor_id = db.Column(db.Integer,
db.ForeignKey('supervisor.employee_id'), nullable=False)
stats = db.relationship('AgentStats', backref='TravelAgent', uselist=False)
countries = db.relationship('Country', secondary='agent_country',
back_populates='agents')
customers = db.relationship('Customer', backref='TravelAgent')
offers = db.relationship('Offer', backref='TravelAgent')
messages = db.relationship('Message', backref='TravelAgent')
```

Agent_Country

joined table to represent the many to many relationship of TravelAgent and Country

country_id from Country and employee_id from TravelAgent as foreign keys

```
'agent_country',
db.Column('country_id', db.Integer, db.ForeignKey('country.country_id')),
db.Column('employee_id', db.Integer,
db.ForeignKey('travel_agent.employee_id'))
```

AgentStats

stats_id – unique identifier of stats table – primary key

num_customers – number of customers a travelAgent is assigned to

num_trips – number of successful trips he or she sold

total_revenue – total amount of revenue this travelAgent generated

```
__tablename__ = "agent_stats"
stats_id = db.Column(db.Integer, primary_key=True)
num_customers = db.Column(db.Integer, nullable=False)
num_trips = db.Column(db.Integer, nullable=False, default=0)
total_revenue = db.Column(db.Integer, nullable=False, default=0)

agent_id = db.Column(db.Integer,
db.ForeignKey('travel_agent.employee_id'),nullable=False)
```

Offer

offer_id – unique identifier for an offer – primary key

country – name of the country assigned to this trip

total_price – total costs for this trip

status – status of this offer, can either be “pending”, “resend”, “changed”, “budget”, “declined” or “accepted”

relationships

agent_id – one to one relationship with TravelAgent

customer_id – one to one relationship with Customer

activities – many to many relationship with Activity

```
__tablename__ = "offer"
offer_id = db.Column(db.Integer, primary_key=True)
country = db.Column(db.String(50), nullable=False)
total_price = db.Column(db.Integer, nullable=False)
status = db.Column(db.String, nullable=False)

agent_id = db.Column(db.Integer, db.ForeignKey('travel_agent.employee_id'),
nullable=False)
customer_id = db.Column(db.Integer, db.ForeignKey('customer.customer_id'),
nullable=False)
activities = db.relationship('Activity', secondary='offer_activity',
back_populates='offers')
```

Offer_Activity

joined table to represent the many to many relationship between Offer and Activity

offer_id from Offer and activity_id from Activity as foreign keys

```
    'offer_activity',
    db.Column('offer_id', db.Integer, db.ForeignKey('offer.offer_id')),
    db.Column('activity_id', db.Integer,
db.ForeignKey('activity.activity_id'))
)
```

Customer

customer_id – unique identifier of a customer – primary key

name – name of a customer, first and last name have to be separated by a space

address – address of a customer, supposedly in the format <Street, Street Number, Zip Code, Cityname>

email – email address of a customer, without constraints

budget – maximum amount of money a customer wants to spend for a trip

preference – preferred country he or she wants to visit, set to “None” if there is no specific preference

expert – boolean value that checks if a customer requires an expert of this country

relationships

agent_id – one to one relationship with TravelAgent

offers – one to many relationship with Offers

str dunder method to represent the name of the customer if needed

```
__tablename__ = "customer"
customer_id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(50),nullable=False)
address = db.Column(db.String(100),nullable=False)
email = db.Column(db.String(100),nullable=False, unique=True)
budget = db.Column(db.Integer, nullable=False)
preference = db.Column(db.String(20), default='None')
expert = db.Column(db.Boolean, unique=False, default=False)

agent_id = db.Column(db.Integer, db.ForeignKey('travel_agent.employee_id'),
nullable=False)
offers = db.relationship('Offer', backref='Customer')

def __str__(self):
    return self.name
```

Country

country_id – unique identifier of a country – primary key

name – name of a supposedly existing country

relationships

activities – many to many relationship with Activity

agents – many to many relationship with TravelAgent

str dunder method to represent the name of the country if needed

```
__tablename__ = "country"
country_id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(50), nullable=False)

activities = db.relationship('Activity', secondary='country_activity',
back_populates='countries')
agents = db.relationship('TravelAgent', secondary='agent_country',
back_populates='countries')

def __str__(self):
    return self.name
```

Country_Activity

joined table to represent the many to many relationship between Country and Activity

country_id from Country and activity_id from Activity as foreign keys

```
    'country_activity',
    db.Column('country_id', db.Integer,
db.ForeignKey('country.country_id')),
    db.Column('activity_id', db.Integer,
db.ForeignKey('activity.activity_id'))
)
```

Activity

activity_id – unique identifier of an activity – primary key

name – name of the activity

price – price of this activity

relationships

countries – many to many relationship with Country

offers – many to many relationship with Offer

str dunder method to represent the name of the activity if needed

```
__tablename__ = "activity"
activity_id = db.Column(db.Integer, primary_key=True)
name = db.Column(db.String(50), nullable=False)
price = db.Column(db.Integer,nullable=False)

countries = db.relationship('Country', secondary='country_activity',
back_populates='activities')
offers = db.relationship('Offer', secondary='offer_activity',
back_populates='activities')

def __str__(self):
    return self.name
```