Take-Home Questions: Operators and Projections

Instructions

- Complete all 4 questions using Python and PyMongo.
- Include proper error handling and comments in your code.
- Test your solutions with the provided sample data.
- **Use the `faker` library to generate 10,000 sample employee documents matching the structure below.**
- Submit both your code and sample output.
- **Submit your Jupyter notebook containing all code, generated data, and results.**
- Explain your approach for complex queries.
- **Upload your Jupyter notebook to a public GitHub repository and include the link in your submission.**
- **Submit your GitHub repository link and the notebook file through the provided Google Form submission link.**

Sample Employee Document

```
{
    "_id": ObjectId("..."),
    "employee_id": "EMP001",
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@company.com",
    "department": "Engineering",
    "position": "Senior Developer",
    "salary": 95000,
    "years_experience": 8,
    "performance_rating": 4.2,
    "skills": ["Python", "JavaScript", "MongoDB", "Docker"],
    "hire_date": ISODate("2020-03-15"),
    "last_promotion": ISODate("2022-06-01"),
    "is_remote": True,
    "address": {
        "city": "San Francisco",
        "state": "CA",
        "zip_code": "94102"
    }
}
```

Tasks

Write Python functions using PyMongo to solve the following:

1. **High Performers Query:** Find all employees with performance rating >= 4.0 **AND** salary > 80,000. Return only their name, department, salary, and performance rating.
2. **Experience-Based Filtering:** Find employees with 5-10 years of experience (inclusive) who earn between $70,000 and 120,000. Project only essential contact information (name, email, department).
3. **Salary Range Analysis:** Find employees whose salary is **NOT** in the range of $60,000 - 100,000. Show their full name (concatenated), current salary, and years of experience.
4. **Recent Hires:** Find employees hired in the last 2 years with performance rating > 3.5. Return custom fields showing "full_name", "tenure_months", and "annual_salary".

Expected Deliverables

- Python functions with proper error handling.
- Sample output showing at least 3 results for each query.
- Comments explaining your operator choices.
- **Jupyter notebook file containing all code, generated data, and results.**
- **GitHub repository link containing your notebook.**

```
!pip install pymongo
```

```
Collecting pymongo
  Downloading pymongo-4.14.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Downloading pymongo-4.14.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 28.0 MB/s eta 0:00:00
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 20.5 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.14.0
```

```
!pip install faker
```

```
Collecting faker
  Downloading faker-37.5.3-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: tzdata in /usr/local/lib/python3.11/dist-packages (from faker) (2025.2)
Downloading faker-37.5.3-py3-none-any.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 29.0 MB/s eta 0:00:00
Installing collected packages: faker
Successfully installed faker-37.5.3
```

```
from pymongo import MongoClient
from datetime import datetime, timedelta
from bson.objectid import ObjectId
from faker import Faker
import random
import pprint as pp
```

```
client = MongoClient("mongodb+srv://Abisola_lufadeju:Oluwafikayomibaby123.@cluster0.yy4xw2j.mongodb.net/")
db = client["Employees_db"]
collection = db["Employees_collection"]

fake = Faker()
```

```
# Function to generate a single employee document
def generate_employee():
    hire_date = fake.date_time_between(start_date="-5y", end_date="now")
    last_promotion = fake.date_time_between(start_date=hire_date, end_date="now") if random.random() > 0.3 else None
    return {
        "_id": fake.uuid4(),  # Unique ID
        "employee_id": f"EMP{fake.random_number(digits=5, fix_len=True)}",
        "first_name": fake.first_name(),
        "last_name": fake.last_name(),
        "email": fake.email(),
        "department": random.choice(["Engineering", "HR", "Marketing", "Sales"]),
        "position": random.choice(["Senior Developer", "Manager", "Analyst"]),
        "salary": random.randint(50000, 150000),
        "years_experience": random.randint(1, 15),
        "performance_rating": round(random.uniform(2.0, 5.0), 1),
        "skills": random.sample(["Python", "JavaScript", "MongoDB", "Docker", "AWS"], k=random.randint(2, 4)),
        "hire_date": hire_date,
        "last_promotion": last_promotion,
        "is_remote": random.choice([True, False]),
        "address": {
            "city": fake.city(),
            "state": fake.state_abbr(),
            "zip_code": fake.zipcode()
        }
    }

# Generate and insert 10,000 employee documents
try:
    employees = [generate_employee() for _ in range(10000)]
    result = collection.insert_many(employees, ordered=False)  # ordered=False to handle duplicates
    print(f"Inserted {len(result.inserted_ids)} employee documents.")
except OperationFailure as e:
    print(f"Insertion failed: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Inserted 10000 employee documents.
```

CRUD_operations

```
# Function to find high performers
def high_performers():
    try:
        query = {
            "performance_rating": {"$gte": 4.0},  # Greater than or equal to 4.0
            "salary": {"$gt": 80000}  # Greater than 80,000
        }
        projection = {"_id": 0, "first_name": 1, "last_name": 1, "salary": 1, "performance_rating": 1}
        results = list(collection.find(query, projection).limit(3))  # Limit to 3 for sample output
        pp.pprint(results)
        return results
    except Exception as e:
        print(f"Error querying high performers: {e}")
        return []

# Test the function
print("Sample High Performers:")
high_performers()
```

```
Sample High Performers:
 [{'first_name': 'Cynthia',
   'last_name': 'Duran',
   'performance_rating': 5.0,
   'salary': 84943},
  {'first_name': 'Nicholas',
   'last_name': 'Brown',
   'performance_rating': 4.9,
   'salary': 123661},
  {'first_name': 'Jay',
   'last_name': 'Skinner',
   'performance_rating': 4.7,
   'salary': 142769}]
 [{'first_name': 'Cynthia',
   'last_name': 'Duran',
   'salary': 84943,
   'performance_rating': 5.0},
  {'first_name': 'Nicholas',
   'last_name': 'Brown',
   'salary': 123661,
   'performance_rating': 4.9},
  {'first_name': 'Jay',
   'last_name': 'Skinner',
   'salary': 142769,
   'performance_rating': 4.7}]
```

##Question 2 - Employees with 5-10 Years Experience and Salary 60,000-100,000

```
# Function to find employees with specific experience and salary range
def experience_salary_range():
    try:
        query = {
            "$and": [
                {"years_experience": {"$gte": 5, "$lte": 10}},  # Between 5 and 10 years
                {"salary": {"$gte": 60000, "$lte": 100000}}  # Between 60,000 and 100,000
            ]
        }
        projection = {"_id": 0, "first_name": 1, "last_name": 1, "years_experience": 1, "salary": 1}
        results = list(collection.find(query, projection).limit(3))
        pp.pprint(results)
        return results
    except Exception as e:
        print(f"Error querying experience and salary range: {e}")
        return []

# Test the function
print("Sample Employees with 5-10 Years Experience and Salary 60,000-100,000:")
experience_salary_range()
```

```
Sample Employees with 5-10 Years Experience and Salary 60,000-100,000:
 [{'first_name': 'Kimberly',
   'last_name': 'Simon',
   'salary': 98000,
   'years_experience': 8},
```

```
       {'first_name': 'Nathan',
        'last_name': 'Vaughn',
        'salary': 68089,
        'years_experience': 5},
       {'first_name': 'Justin',
        'last_name': 'Clark',
        'salary': 79943,
        'years_experience': 10}]
      [{'first_name': 'Kimberly',
        'last_name': 'Simon',
        'salary': 98000,
        'years_experience': 8},
       {'first_name': 'Nathan',
        'last_name': 'Vaughn',
        'salary': 68089,
        'years_experience': 5},
       {'first_name': 'Justin',
        'last_name': 'Clark',
        'salary': 79943,
        'years_experience': 10}]
```

Question 3 - Employees NOT in Salary Range 70,000-90,000

```
# Function to find employees outside salary range 70,000-90,000
def outside_salary_range():
    try:
        query = {
            "salary": {"$not": {"$gte": 70000, "$lte": 90000}}  # Not between 70,000 and 90,000
        }
        projection = {"_id": 0, "first_name": 1, "last_name": 1, "salary": 1}
        results = list(collection.find(query, projection).limit(3))
        pp.pprint(results)
        return results
    except Exception as e:
        print(f"Error querying outside salary range: {e}")
        return []

# Test the function
print("Sample Employees NOT in Salary Range 70,000-90,000:")
outside_salary_range()
```

```
Sample Employees NOT in Salary Range 70,000-90,000:
 [{'first_name': 'Christopher', 'last_name': 'Lewis', 'salary': 132834},
  {'first_name': 'Antonio', 'last_name': 'Cole', 'salary': 114857},
  {'first_name': 'Shelby', 'last_name': 'Santiago', 'salary': 129536}]
 [{'first_name': 'Christopher', 'last_name': 'Lewis', 'salary': 132834},
  {'first_name': 'Antonio', 'last_name': 'Cole', 'salary': 114857},
  {'first_name': 'Shelby', 'last_name': 'Santiago', 'salary': 129536}]
```

Question 4 - Hired in Last 2 Years with Performance Rating > 4.0

```
# Function to find employees hired in last 2 years with high performance
def recent_high_performers():
    try:
        two_years_ago = datetime.now() - timedelta(days=2*365)
        query = {
            "hire_date": {"$gt": two_years_ago},  # Hired after 2 years ago
            "performance_rating": {"$gt": 4.0}  # Performance > 4.0
        }
        projection = {"_id": 0, "first_name": 1, "last_name": 1, "hire_date": 1, "performance_rating": 1}
        results = list(collection.find(query, projection).limit(3))
        pp.pprint(results)
        return results
    except Exception as e:
        print(f"Error querying recent high performers: {e}")
        return []

# Test the function
print("Sample Employees Hired in Last 2 Years with Performance > 4.0:")
recent_high_performers()
```

```
Sample Employees Hired in Last 2 Years with Performance > 4.0:
 [{'first_name': 'Cynthia',
   'hire_date': datetime.datetime(2024, 2, 7, 6, 23, 44, 542000),
   'last_name': 'Duran',
```

```
  'performance_rating': 5.0},
 {'first_name': 'Rebecca',
  'hire_date': datetime.datetime(2023, 9, 10, 4, 45, 39, 120000),
  'last_name': 'Mccullough',
  'performance_rating': 4.4},
 {'first_name': 'Benjamin',
  'hire_date': datetime.datetime(2024, 12, 21, 14, 31, 44, 314000),
  'last_name': 'Baldwin',
  'performance_rating': 4.3}]
[{'first_name': 'Cynthia',
  'last_name': 'Duran',
  'performance_rating': 5.0,
  'hire_date': datetime.datetime(2024, 2, 7, 6, 23, 44, 542000)},
 {'first_name': 'Rebecca',
  'last_name': 'Mccullough',
  'performance_rating': 4.4,
  'hire_date': datetime.datetime(2023, 9, 10, 4, 45, 39, 120000)},
 {'first_name': 'Benjamin',
  'last_name': 'Baldwin',
  'performance_rating': 4.3,
  'hire_date': datetime.datetime(2024, 12, 21, 14, 31, 44, 314000)}]
```

Explanation of Approach for Complex Queries

Explanation Cell (Markdown)

"""

Approach for Complex Queries

1. **High Performers Query**: Used `$gte` for performance_rating and `$gt` for salary to filter employees. The implicit `$and` operator combines these conditions since they are in the same query object. Limited to 3 results for readability.
2. **Experience and Salary Range Query**: Employed `$and` explicitly to combine `$gte` and `$lte` conditions for both years_experience and salary, ensuring the range is inclusive. Projection excludes unnecessary fields.
3. **Outside Salary Range Query**: Utilized `$not` with `$gte` and `$lte` to exclude the 70,000-90,000 salary range, leveraging MongoDB's negation operator for efficient filtering.
4. **Recent High Performers Query**: Calculated the date 2 years ago using `timedelta` and used `$gt` to filter hire_dates. Combined with `$gt` for performance_rating using implicit `$and`. Dates are handled as Python `datetime` objects, compatible with PyMongo.

Error Handling: Each function includes try-except blocks to catch `ConnectionError`, `OperationFailure`, and general exceptions, providing meaningful error messages.

Performance Consideration: For 10,000 documents, indexes on `performance_rating`, `salary`, `years_experience`, and `hire_date` could optimize queries, though not implemented here due to assignment scope. """