In [ ]:

```python
# Ameya Gaitonde

# Install pymongo if applicable

!python -m pip install "pymongo[srv]"
```

```
Collecting pymongo[srv]
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
WARNING: pymongo 4.10.1 does not provide the extra 'srv'
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo[srv])
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 6.1 MB/s eta 0:00:00
Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 28.1 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.10.1
```

In [ ]:

```python
# setup the client. Make sure to remove your password while submitting

import pymongo
from bson.json_util import dumps

# Password obfuscated
#uri = "my-Atlas-instance-uri"
client = pymongo.MongoClient(uri)
db = client['zotmusic']
```

In [ ]:

```python
# Use Python comments to answer Q1 below
# 1.A: In records collection, "released_by" is of "Document" datatype.

# 1.B: Yes, there is a newly added field.
#    i. It is called "_id", and does not exist in records.json.

#    ii. Its datatype is "ObjectId".

#    iii. It serves as a unique identifier for each document in the collection
#         and is automatically generated by the MongoDB driver. The "_id" field
#         may contain values of any BSON data type other than array, regex,
#         or undefined.

#    iv. Yes, _id is unique for all documents in the collection. It serves as a
#        primary key for documents.

# 1.C: In the reviews collection, you can link the "user_id" nested field within
#      the "posted_by" field with the users collection to retrieve the name of
#     the review's creator.


# 1.D: The subscriptions field is of the "String" datatype, and its values are
#      "monthly", "free", and "yearly".


# 1.E The genres field is an Array of strings, and a user maintains an
# average of 4.62 genres.
```

In [ ]:

```python
# 2A
document = db.records.find( { 'title': 'Big worry' } )

# find() returns a cursor to the document
for doc in document:
    print(doc)
```

```
{'_id': ObjectId('671d3752ec3776a702e215ca'), 'record_id': 'record_HPJtcrZW', 'title': 'Big worry', 'genre': 'Reggae', 'released_by': {'artist_u
ser_id': 'user_YEGFl37i', 'release_date': '2020-11-15'}, 'is_album': False, 'is_single': True, 'video_url': 'http://walker-sanchez.org/', 'song
s': [{'track_number': 1, 'title': 'Big worry', 'length': 435, 'bpm': 142, 'mood': 'people'}]}
```

In [ ]:

```python
# 2B
# Get the 3 most recently released albums
documents = db.records.find().sort("released_by.release_date", -1).limit(3)

for doc in documents:
    title = doc.get('title')
    release_date = doc.get('released_by', {}).get('release_date')
    print(f"Title: {title}, Release Date: {release_date}")
```

```
Title: Training citizen many, Release Date: 2023-12-30
Title: Position option hair, Release Date: 2023-12-29
Title: Whose hear, Release Date: 2023-12-28
```

In [ ]:

```python
# 2C

count_users = db.users.count_documents({
    "is_listener": True,
    "is_artist": True
})

print(f"There are {count_users} who are both listeners and artists.")
```

```
There are 100 who are both listeners and artists.
```

In [ ]:

```python
# 2D
conditions_query = {
    "$or": [
        {
            "is_single": True,
            "genre": "Pop",
            "released_by.release_date": {"$gt": "2023-06-30"}
        },
        {
            "is_album": True,
            "genre": "Jazz",
            "released_by.release_date": {"$lt": "2023-01-01"}
        }
    ]
}

# Count the documents that match the query
count_docs = db.records.count_documents(conditions_query)

print(f"There are {count_docs} records that match the given conditions.")
```

```
There are 24 records that match the given conditions.
```

In [ ]:

```python
# 2E


users_query = {
    "is_listener": True,
    "$expr": {
        "$gte": [{"$size": "$genres"}, 9] # Check that len(genres) >= 9
    },
    "address.street": {"$exists": True, "$ne": ""},
    "address.city": {"$exists": True, "$ne": ""},
    "address.state": {"$exists": True, "$ne": ""},
}




# Project the email and formatted full name
projection = {
    "email": 1,
    "full_name": {
        "$concat": [
            {"$ifNull": ["$real_name.last_name", ""]},
            ", ",
            {"$ifNull": ["$real_name.first_name", ""]}
        ]
    }
}

# Find the documents that match the query and project the necessary fields
documents = db.users.find(users_query, projection)

for doc in documents:
    print(f"email: {doc['email']}, full name: {doc['full_name']}")
```

```
email: courtney36@protonmail.com, full name: Sherman, Jason
email: ryanmorgan@icloud.com, full name: Wood, Kimberly
email: gomezbrittany@foxmail.com, full name: Woodward, Holly
```

In [ ]:

```python
# 2F

agg_pipeline = [
    {
        "$match": {
            "joined_date": {"$regex": "^2022"},  # joined_date begins with "2022"
            "is_listener": True,
            "subscription": "yearly"
        }
    },
    {
        "$count": "total_listeners"
    }
]

agg_result = list(db.users.aggregate(agg_pipeline))

print(agg_result[0]['total_listeners'], "listeners joined in 2022 with a yearly subscription.")
```

```
11 listeners joined in 2022 with a yearly subscription.
```

In [ ]:

```python
# 2G

pipeline = [
    {
        # Calculate the total reviews
        "$group": {
            "_id": "$posted_by.user_id",
            "total_reviews": {"$sum": 1}
        }
    },
    {
        # Sort by total reviews in descending order
        "$sort": {
            "total_reviews": -1
        }
    },
    {
        "$group": {
            "_id": None,
            "users": {
                "$push": {
                    "user_id": "$_id",
                    "total_reviews": "$total_reviews"
                }
            }
        }
    },
    {
        # Creates separate documents for each item in users array
        "$unwind": "$users"
    },
    {
        # Append the new field of dense_rank to the document
        "$setWindowFields": {
            "sortBy": {"users.total_reviews": -1},
            "output": {
                "dense_rank": {"$denseRank": {}}
            }
        }
    },
    {   # Keep top 4 users by dense rank
        "$match": {
            "dense_rank": {"$lte": 4}
        }
    },
    {
        # Project relevant information
        "$project": {
            "_id": 0,
            "user_id": "$users.user_id",
            "total_reviews": "$users.total_reviews",
            "dense_rank": "$dense_rank"
        }
    }
]


result = list(db.reviews.aggregate(pipeline))


for item in result:
    print(f"User ID: {item['user_id']}, Total Reviews: {item['total_reviews']}, Dense Rank: {item['dense_rank']}")
```

```
Top 4 Users with Dense Rank: [{'user_id': 'user_ux9_qRS-', 'total_reviews': 74, 'dense_rank': 1}, {'user_id': 'user_vu4RI4dq', 'total_reviews':
71, 'dense_rank': 2}, {'user_id': 'user_Z217vp0p', 'total_reviews': 69, 'dense_rank': 3}, {'user_id': 'user_ODWOljEV', 'total_reviews': 68, 'den
se_rank': 4}, {'user_id': 'user_V3iJ0OgE', 'total_reviews': 68, 'dense_rank': 4}]
User ID: user_ux9_qRS-, Total Reviews: 74, Dense Rank: 1
User ID: user_vu4RI4dq, Total Reviews: 71, Dense Rank: 2
User ID: user_Z217vp0p, Total Reviews: 69, Dense Rank: 3
User ID: user_ODWOljEV, Total Reviews: 68, Dense Rank: 4
User ID: user_V3iJ0OgE, Total Reviews: 68, Dense Rank: 4
```

In [ ]:

```python
# 2H

# Find user_id values where record_id is 'record_qGHboIDL' and rating >= 4
query = {
    "record_id": "record_qGHboIDL",
    "rating": {"$gte": 4}  # rating greater than or equal to 4
}


result = db.reviews.find(query, {"posted_by.user_id": 1, "_id": 0})

# Retrieve relevant user_id values
user_ids = [review["posted_by"]["user_id"] for review in result]
#print("user IDs:", user_ids)



# Find users using the relevant user_id values from above.
query = {
    "user_id": {"$in": user_ids}
}


users_result = db.users.find(query, {
    "user_id": 1,
    "nickname": 1,
    "address.street": 1,
    "address.city": 1,
    "address.state": 1,
    "address.zip": 1,
    "_id": 0
})


# Print the information from each user
for user in users_result:
    print("user ID:", user.get("user_id"))
    print("nickname:", user.get("nickname", "N/A"))
    print("address:", {
        "street": user.get("address", {}).get("street", "N/A"),
        "city": user.get("address", {}).get("city", "N/A"),
        "state": user.get("address", {}).get("state", "N/A"),
        "zip": user.get("address", {}).get("zip", "N/A")
    })
    print()
```

```
user ID: user_BerqqJn3
nickname: davidchad
address: {'street': '4174 Michele Gardens Suite 245', 'city': 'Ortizbury', 'state': 'Delaware', 'zip': '53945'}

user ID: user_p4EheoPI
nickname: ellengibson
address: {'street': '9400 Wu Court', 'city': 'New Andre', 'state': 'New Hampshire', 'zip': '95877'}

user ID: user_IJys3jPu
nickname: paynedavid
address: {'street': '3446 Cynthia Unions Apt. 809', 'city': 'Christietown', 'state': 'Colorado', 'zip': 'N/A'}
```

In [ ]:

```python
# 2I

# Pipeline to find record_id where avg rating = 4.0
first_pipeline = [
    {
        # Group by record_id and find mean rating for each record_id
        '$group': {
            '_id': '$record_id',
            'average_rating': { '$avg': '$rating' }
        }
    },
    {
        # Match only record_id with average_rating = 4.0
        '$match': {
            'average_rating': 4.0
        }
    },
    {
        # Project only the record_id
        '$project': {
            'record_id': '$_id',
            '_id': 0
        }
    }
]

first_result = list(db.reviews.aggregate(first_pipeline))

recordIDValues = [result['record_id'] for result in first_result]

#print(recordIDValues)



# Now, using the above record_id values, we must use the record collection to
# retrieve songs matching the above recordIDValues, where bpm < 63


second_pipeline = [
    {
        # First record_id values above
        '$match': {
            'record_id': { '$in': recordIDValues },
        }
    },
    {
        # Flatten the songs array for easier filtering
        '$unwind': '$songs'
    },
    {
        # Only match songs.bpm < 63
        '$match': { 'songs.bpm': { '$lt': 63 } }
    },
    {
        # Randomly sample 5 documents
        '$sample': { 'size': 5 }
    },
    {
        # Project relevant fields
        '$project': {
            'record_id': 1,
            'title': 1,
            'track_number': '$songs.track_number',
            'song_title': '$songs.title'
        }
    }
]

# Execute the aggregation pipeline
random_sample = list(db.records.aggregate(second_pipeline))

# Print the relevant information from the random sample
for document in random_sample:
    print("Record_id:", document['record_id'])
    print("Title:", document['title'])
    print("Track Number:", document['track_number'])
    print("Song Title:", document['song_title'])
    print()
```

```
Record_id: record_D0VTjwgg
Title: Material season
Track Number: 8
Song Title: Well peace

Record_id: record_ZBXcXXcq
Title: Forward score matter
Track Number: 1
Song Title: Note career

Record_id: record_PzgNFItj
Title: Develop between prove
Track Number: 9
Song Title: System oil

Record_id: record_AxTzPFAx
Title: Gas risk
Track Number: 9
Song Title: Throw fall

Record_id: record_g6asf3ux
Title: Garden approach
Track Number: 1
Song Title: Try both whole
```

In [ ]:

```python
# 2J

# Not sure about this one

final_pipeline = [
    {
        # Joining the relevant collections on user_id
        # and record_id.
        '$lookup': {
            'from': 'sessions',
            'localField': 'user_id',
            'foreignField': 'user_id',
            'as': 'sessions'
        }
    },
    {
        '$unwind': '$sessions'
    },
    {
        '$lookup': {
            'from': 'records',
            'localField': 'sessions.record_id',
            'foreignField': 'record_id',
            'as': 'records'
        }
    },
    {
        '$unwind': '$records'
    },
    {
        # Matching conditions
        '$match': {
            'records.release_date': {'$gt': '2023-01-01'},
            'sessions.replay_count': {'$gt': 3},
            'sessions.device': 'mobile-app'
        }
    },
    {
        # GROUP BY equivalent
        '$group': {
            '_id': {
                'user_id': '$user_id',
                'nickname': '$nickname',
                'email': '$email',
                'joined_date': '$joined_date'
            },
            # $addToSet distinct genres
            'distinct_genres': {'$addToSet': '$records.genre'}
        }
    },
    {
        # Sort by joined_date in descending order.
        '$sort': {
            '_id.joined_date': -1
        }
    },
    {
        # Project the nickname and email
        '$project': {
            'nickname': '$_id.nickname',
            'email': '$_id.email'
        }
    },
    {
        # Limit to only top 2 results
        '$limit': 2
    }
]

documents = db.users.aggregate(final_pipeline)


for document in documents:
    print(document)
```