PyMongo Insertion and Query Guide

Introduction

PyMongo is a Python library that allows interaction with MongoDB, a NoSQL document-oriented database. This guide covers various ways to insert and query documents using PyMongo, with detailed explanations of operators and methods.

Setup and Connection

Install PyMongo:

```
pip install pymongo
```

Connect to MongoDB:

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['mydatabase']
collection = db['mycollection']
```

Insert Operations

Insert One Document:

```
result = collection.insert_one({'name': 'Alice', 'age': 25})
```

- Inserts a single document into the collection.
- Returns an InsertOneResult with the inserted_id.

Insert Many Documents:

```
docs = [{'name': 'Bob', 'age': 30}, {'name': 'Charlie', 'age': 35}]
result = collection.insert_many(docs)
```

- Inserts multiple documents.
- Returns an InsertManyResult with a list of inserted_ids.

Find Operations

Find One:

```
collection.find_one({'name': 'Alice'})
```

- Returns the first document that matches the filter.

Find All:

```
for doc in collection.find(): print(doc)
```

- Iterates over all documents in the collection.

Comparison Operators

$gt (greater than): {'age': {'$gt': 30}}
$lt (less than): {'age': {'$lt': 30}}
$gte (greater than or equal): {'age': {'$gte': 25}}

PyMongo Insertion and Query Guide

$lte (less than or equal): {'age': {'$lte': 35}}
$eq (equal): {'name': {'$eq': 'Alice'}}
$ne (not equal): {'name': {'$ne': 'Bob'}}
- These operators filter results based on numeric or string comparisons.

Logical Operators

$or: {'$or': [{'name': 'Alice'}, {'age': 30}]}
$and: {'$and': [{'age': {'$gte': 25}}, {'age': {'$lte': 35}}]}
$not: {'age': {'$not': {'$gt': 30}}}
$nor: {'$nor': [{'age': 30}, {'name': 'Bob'}]}
- Combine multiple query expressions for complex filtering.

Projection

Projection controls which fields are returned:
collection.find({}, {'_id': 0, 'name': 1})
- Includes 'name', excludes '_id'.
- A value of 1 includes a field, 0 excludes it.

Sorting and Limiting

Sort:
```
collection.find().sort('age', 1)  # Ascending
collection.find().sort('age', -1)  # Descending
```

Limit/Skip:
```
collection.find().limit(5)
collection.find().skip(5).limit(5)
```
- Use for pagination or controlling output size.

Advanced Query Operators

$in: {'name': {'$in': ['Alice', 'Bob']}} - Matches any listed value
$nin: {'name': {'$nin': ['Charlie']}} - Excludes listed values
$exists: {'email': {'$exists': True}} - Checks if a field exists
$type: {'age': {'$type': 'int'}} - Filters by BSON type (e.g., 'string', 'int')

Counting and Distinct

Count documents matching a filter:
```
collection.count_documents({'age': {'$gt': 25}})
```

Get distinct values from a field:
```
collection.distinct('name')
```

PyMongo Insertion and Query Guide

Regex and Text Search

Regex:

```
collection.find({'name': {'$regex': '^A'}})
```

- Matches strings starting with 'A'.

Text Search:

```
collection.create_index([('name', 'text')])
collection.find({'$text': {'$search': 'Alice'}})
```

- Requires a text index. Performs full-text search.

Indexing and Explain

Create Index:

```
collection.create_index('name')
collection.create_index([('age', 1)])
```

- Improves query performance.

Explain Query Plan:

```
collection.find({'name': 'Alice'}).explain()
```

- Provides execution details for performance tuning.