MongoDB Comprehensive Guide

Database and Collection/Schema

Concept	MongoDB	SQL	Explanation
Database	Database	Database	Both refer to a logical container for collections or tables. A MongoDB instance can host multiple databases, similar to an SQL server.
Row	Document	Row (or Record)	A document in MongoDB is a set of key-value pairs, equivalent to a row in SQL, which consists of columns.
Table	Collection	Table	In MongoDB, a collection is a group of documents. In SQL, a table is a group of rows. Both serve as the primary storage unit.
Column	Field	Column	Fields in MongoDB documents are similar to columns in SQL tables, representing individual data attributes.
Primary Key	`_id` field	Primary Key	In MongoDB, each document has a unique `_id` field by default. In SQL, a primary key uniquely identifies each row.

Data Operations

Operation	MongoDB	SQL	Explanation
Insert Data	`insertOne`, `insertMany`	`INSERT INTO`	MongoDB uses `insertOne` and `insertMany` to add documents. SQL uses `INSERT INTO` to add rows to a table.
Read Data	`find`, `findOne`	`SELECT`	MongoDB uses `find` and `findOne` to retrieve documents. SQL uses `SELECT` to retrieve rows.
Update Data	`updateOne`, `updateMany`	`UPDATE`	MongoDB uses `updateOne` and `updateMany` to modify documents. SQL uses `UPDATE` to modify rows.
Delete Data	`deleteOne`, `deleteMany`	`DELETE`	MongoDB uses `deleteOne` and `deleteMany` to remove documents. SQL uses `DELETE` to remove rows.

Connecting to the Database

SQL:

```
-- Connect to the SQL server and select the database
-- Assuming using SQL commands in a script or via a client interface
USE my_database;
```

MongoDB:

```
from pymongo import MongoClient

# Connect to the MongoDB server
client = MongoClient("mongodb://localhost:27017/")

# Select the database
db = client.my_database

# Select the collection
collection = db.my_collection
```

Inserting Data

SQL:

```
-- Insert data into the table
INSERT INTO my_collection (name, age, city) VALUES ('Alice', 25, 'New York');
```

```
INSERT INTO my_collection (name, age, city) VALUES ('Bob', 30, 'San Francisco');
INSERT INTO my_collection (name, age, city) VALUES ('Charlie', 35, 'Los Angeles');
```

MongoDB:

```
# Insert documents
collection.insert_one({"name": "Alice", "age": 25, "city": "New York"})
collection.insert_many([
    {"name": "Bob", "age": 30, "city": "San Francisco"},
    {"name": "Charlie", "age": 35, "city": "Los Angeles"}
])
```

Finding Data

SQL:

```
-- Select all documents
SELECT * FROM my_collection;

-- Select documents with age > 25
SELECT * FROM my_collection WHERE age > 25;

-- Select single document with name 'Alice'
SELECT * FROM my_collection WHERE name = 'Alice' LIMIT 1;
```

MongoDB:

```
# Find documents
print("All documents:")
for doc in collection.find():
    print(doc)

print("Documents with age > 25:")
for doc in collection.find({ "age": { "$gt": 25 } }):
    print(doc)

print("Single document with name Alice:")
print(collection.find_one({ "name": "Alice" }))
```

MongoDB Query Operators

- `$lt` stands for "less than."
- `$gt` stands for "greater than."
- `gte` stands for "greater than or equal to".
- `lte` stands for "less than or equal to".

These operators are used within query documents to specify conditions for selecting documents from a collection.

SQL Equivalents of `gte` and `lte`

In SQL, `>=` and `<=` are used to filter rows based on comparison conditions:

- `>=` means "greater than or equal to".
- `<=` means "less than or equal to".

Using `gte` and `lte` in MongoDB

Example: Find Rows with Age Greater Than or Equal to 30

SQL Query:

```
SELECT * FROM my_collection WHERE age >= 30;
```

Explanation:

- `WHERE age >= 30`: This condition selects rows where the `age` column is greater than or equal to 30. The `>=` operator is used to specify this condition.

Example: Find Documents with Age Greater Than or Equal to 30

MongoDB Code:

```
# Find documents with age >= 30
for doc in collection.find({ "age": { "$gte": 30 } }):
    print(doc)
```

Explanation:

- `collection.find({ "age": { "$gte": 30 } })`: This query selects documents where the `age` field is greater than or equal to 30. The `$gte` operator is used within the query document to specify this condition.

Example: Find Rows with Age Less Than or Equal to 30

SQL Query:

```
SELECT * FROM my_collection WHERE age <= 30;
```

Explanation:

- `WHERE age <= 30`: This condition selects rows where the `age` column is less than or equal to 30. The `<=` operator is used to specify this condition.

MongoDB Code:

```
# Find documents with age <= 30
for doc in collection.find({ "age": { "$lte": 30 } }):
    print(doc)
```

Explanation:

- `collection.find({ "age": { "$lte": 30 } })`: This query selects documents where the `age` field is less than or equal to 30. The `$lte` operator is used within the query document to specify this condition.

Updating Data

SQL:

```
-- Update age of the document with name 'Alice'
UPDATE my_collection SET age = 26 WHERE name = 'Alice';

-- Update city for all documents where city is 'New York'
UPDATE my_collection SET city = 'Boston' WHERE city = 'New York';
```

MongoDB:

```
# Update documents
collection.update_one({ "name": "Alice" }, { "$set": { "age": 26 } })
collection.update_many({ "city": "New York" }, { "$set": { "city": "Boston" } })
```

Deleting Data

SQL:

```sql
-- Delete document with name 'Charlie'
DELETE FROM my_collection WHERE name = 'Charlie';

-- Delete documents where age is less than 30
DELETE FROM my_collection WHERE age < 30;
```

MongoDB:

```
# Delete documents
collection.delete_one({ "name": "Charlie" })
collection.delete_many({ "age": { "$lt": 30 } })
```

Additional Find Operations

SQL:

```sql
-- Select documents with age >= 25, only show name and city
SELECT name, city FROM my_collection WHERE age >= 25;

-- Select all documents, sorted by age in descending order
SELECT * FROM my_collection ORDER BY age DESC;

-- Select only 2 documents
SELECT * FROM my_collection LIMIT 2;

-- Select all documents, skip the first one (not all SQL dialects support OFFSET without LIMI
T)
SELECT * FROM my_collection OFFSET 1;
```

MongoDB:

```
# Additional find operations
print("Documents with projection:")
for doc in collection.find({ "age": { "$gte": 25 } }, { "name": 1, "city": 1, "_id": 0 }):
    print(doc)

print("Documents sorted by age:")
for doc in collection.find().sort("age", -1):
    print(doc)

print("Limited documents:")
for doc in collection.find().limit(2):
    print(doc)

print("Skipped first document:")
for doc in collection.find().skip(1):
    print(doc)
```

Summary

- **Connecting to the Database**: MongoDB uses a `MongoClient` object, while SQL typically uses `USE` to select a database.
- **Inserting Data**: MongoDB uses `insert_one` and `insert_many`, whereas SQL uses `INSERT INTO`.
- **Finding Data**: MongoDB uses `find` and `find_one` with query dictionaries, while SQL uses `SELECT` with `WHERE` conditions.
- **Updating Data**: MongoDB uses `update_one` and `update_many` with `$set`, while SQL uses `UPDATE` with `SET`.
- **Deleting Data**: MongoDB uses `delete_one` and `delete_many`, while SQL uses `DELETE FROM`.

- **Additional Find Operations**: MongoDB provides various methods like `sort`, `limit`, and `skip` as part of its `find` operations, while SQL uses `ORDER BY`, `LIMIT`, and `OFFSET`.

Download MongoDB Community Version by clicking this: Download MongoDB Community Server | MongoDB

Install MongoDB following the below Youtube video link (Follow the first 1 minute only for this project):

Install MongoDB on windows 11 in Hindi (Step by step) - YouTube

Install `pymongo` in PyCharm terminal by typing the following:

```
pip install pymongo
```

1. **Connect to MongoDB**:
   Connect to the MongoDB server and create the database and collections.
2. **Insert Data**:
   Insert the data into MongoDB collections.

Project Example:

```
from pymongo import MongoClient

# Connect to the MongoDB server
client = MongoClient("mongodb://localhost:27017/")

# Create the database
db = client.video_platform

# Create the collections
channels_collection = db.channels
videos_collection = db.videos
playlists_collection = db.playlists
comments_collection = db.comments

# Insert data into Channels collection
channels = [
    {"channel_id": 1, "channel_name": "TechTalk", "description": "Technology and gadget revie
ws", "total_videos": 120, "subscribers": 1300000},
    {"channel_id": 2, "channel_name": "DailyVlogs", "description": "Daily life vlogs and trav
el", "total_videos": 250, "subscribers": 800000},
    {"channel_id": 3, "channel_name": "CookingWithLove", "description": "Delicious recipes an
d cooking tips", "total_videos": 300, "subscribers": 500000},
    {"channel_id": 4, "channel_name": "FitnessFirst", "description": "Workouts and fitness ti
ps", "total_videos": 220, "subscribers": 750000},
    {"channel_id": 5, "channel_name": "TravelWithMe", "description": "Travel guides and tip
s", "total_videos": 180, "subscribers": 650000},
    {"channel_id": 6, "channel_name": "GameOn", "description": "Gaming reviews and walkthroug
hs", "total_videos": 150, "subscribers": 900000},
    {"channel_id": 7, "channel_name": "MusicMania", "description": "Music reviews and tutoria
ls", "total_videos": 110, "subscribers": 400000},
    {"channel_id": 8, "channel_name": "AutoWorld", "description": "Automobile reviews and new
s", "total_videos": 200, "subscribers": 850000},
    {"channel_id": 9, "channel_name": "FashionFrenzy", "description": "Fashion and beauty tip
s", "total_videos": 170, "subscribers": 550000},
    {"channel_id": 10, "channel_name": "HistoryBuff", "description": "Historical documentarie
s and discussions", "total_videos": 95, "subscribers": 300000}
```

```
]
channels_collection.insert_many(channels)

# Insert data into Videos collection
videos = [
    {"video_id": 1, "channel_id": 1, "title": "Latest Smartphone Review", "views": 10000, "li
kes": 500},
    {"video_id": 2, "channel_id": 2, "title": "A Day in My Life", "views": 8000, "likes": 30
0},
    {"video_id": 3, "channel_id": 3, "title": "Easy Pasta Recipe", "views": 5000, "likes": 25
0},
    {"video_id": 4, "channel_id": 4, "title": "Morning Yoga Routine", "views": 12000, "like
s": 800},
    {"video_id": 5, "channel_id": 5, "title": "Exploring Paris", "views": 15000, "likes": 100
0},
    {"video_id": 6, "channel_id": 6, "title": "Epic Game Review", "views": 20000, "likes": 15
00},
    {"video_id": 7, "channel_id": 7, "title": "Guitar Tutorial", "views": 9000, "likes": 60
0},
    {"video_id": 8, "channel_id": 8, "title": "Latest Car Models", "views": 18000, "likes": 1
300},
    {"video_id": 9, "channel_id": 9, "title": "Summer Fashion Trends", "views": 7000, "like
s": 400},
    {"video_id": 10, "channel_id": 10, "title": "World War II Documentary", "views": 22000,
"likes": 1900},
    {"video_id": 11, "channel_id": 1, "title": "Laptop Review", "views": 11000, "likes": 55
0},
    {"video_id": 12, "channel_id": 2, "title": "Camping Trip Vlog", "views": 9000, "likes": 3
50},
    {"video_id": 13, "channel_id": 3, "title": "Healthy Salad Recipe", "views": 4000, "like
s": 220},
    {"video_id": 14, "channel_id": 4, "title": "Cardio Workout", "views": 14000, "likes": 85
0},
    {"video_id": 15, "channel_id": 5, "title": "Visiting London", "views": 16000, "likes": 11
00},
    {"video_id": 16, "channel_id": 6, "title": "New Game Trailer", "views": 25000, "likes": 2
000},
    {"video_id": 17, "channel_id": 7, "title": "Piano Tutorial", "views": 9500, "likes": 62
0},
    {"video_id": 18, "channel_id": 8, "title": "Classic Car Review", "views": 19000, "likes":
1350},
    {"video_id": 19, "channel_id": 9, "title": "Winter Fashion Trends", "views": 8000, "like
s": 450},
    {"video_id": 20, "channel_id": 10, "title": "Ancient Egypt Documentary", "views": 23000,
"likes": 1950},
    {"video_id": 21, "channel_id": 1, "title": "Tablet Review", "views": 10500, "likes": 530}
]
videos_collection.insert_many(videos)

# Insert data into Playlists collection
playlists = [
    {"playlist_id": 1, "channel_id": 1, "title": "Smartphone Reviews"},
    {"playlist_id": 2, "channel_id": 2, "title": "Travel Vlogs"},
    {"playlist_id": 3, "channel_id": 3, "title": "Quick Recipes"},
    {"playlist_id": 4, "channel_id": 4, "title": "Fitness Routines"},
    {"playlist_id": 5, "channel_id": 5, "title": "Travel Guides"},
    {"playlist_id": 6, "channel_id": 6, "title": "Game Walkthroughs"},
    {"playlist_id": 7, "channel_id": 7, "title": "Music Tutorials"},
```

```
    {"playlist_id": 8, "channel_id": 8, "title": "Car Reviews"},
    {"playlist_id": 9, "channel_id": 9, "title": "Beauty Tips"},
    {"playlist_id": 10, "channel_id": 10, "title": "History Lessons"},
    {"playlist_id": 11, "channel_id": 1, "title": "Laptop Reviews"},
    {"playlist_id": 12, "channel_id": 2, "title": "Nature Vlogs"},
    {"playlist_id": 13, "channel_id": 3, "title": "Healthy Recipes"},
    {"playlist_id": 14, "channel_id": 4, "title": "Cardio Workouts"},
    {"playlist_id": 15, "channel_id": 5, "title": "City Guides"},
    {"playlist_id": 16, "channel_id": 6, "title": "Game Reviews"},
    {"playlist_id": 17, "channel_id": 7, "title": "Piano Tutorials"},
    {"playlist_id": 18, "channel_id": 8, "title": "Classic Car Reviews"},
    {"playlist_id": 19, "channel_id": 9, "title": "Winter Fashion Tips"},
    {"playlist_id": 20, "channel_id": 10, "title": "Ancient History"}
]
playlists_collection.insert_many(playlists)

# Insert data into Comments collection
comments = [
    {"comment_id": 1, "video_id": 1, "user_name": "TechLover", "comment_text": "Great revie
w!"},
    {"comment_id": 2, "video_id": 2, "user_name": "VlogFan", "comment_text": "Loved this vide
o!"},
    {"comment_id": 3, "video_id": 3, "user_name": "ChefMaster", "comment_text": "Thanks for t
he recipe!"},
    {"comment_id": 4, "video_id": 4, "user_name": "YogaEnthusiast", "comment_text": "This rou
tine is amazing!"},
    {"comment_id": 5, "video_id": 5, "user_name": "Traveler123", "comment_text": "Paris is be
autiful!"},
    {"comment_id": 6, "video_id": 6, "user_name": "GamerDude", "comment_text": "This game is
awesome!"},
    {"comment_id": 7, "video_id": 7, "user_name": "MusicLover", "comment_text": "Great tutori
al!"},
    {"comment_id": 8, "video_id": 8, "user_name": "CarFanatic", "comment_text": "Loved the ca
r review!"},
    {"comment_id": 9, "video_id": 9, "user_name": "Fashionista", "comment_text": "Great fashi
on tips!"},
    {"comment_id": 10, "video_id": 10, "user_name": "HistoryBuff", "comment_text": "Very info
rmative!"},
    {"comment_id": 11, "video_id": 11, "user_name": "TechFan", "comment_text": "Loved the lap
top review!"},
    {"comment_id": 12, "video_id": 12, "user_name": "NatureLover", "comment_text": "Great cam
ping trip vlog!"},
    {"comment_id": 13, "video_id": 13, "user_name": "HealthyEater", "comment_text": "Awesome
salad recipe!"},
    {"comment_id": 14, "video_id": 14, "user_name": "FitnessGuru", "comment_text": "Great car
dio workout!"},
    {"comment_id": 15, "video_id": 15, "user_name": "Traveler456", "comment_text": "London lo
oks amazing!"},
    {"comment_id": 16, "video_id": 16, "user_name": "GamerGirl", "comment_text": "Can’t wait
for this game!"},
    {"comment_id": 17, "video_id": 17, "user_name": "PianoPro", "comment_text": "Very helpful
piano tutorial!"},
    {"comment_id": 18, "video_id": 18, "user_name": "CarEnthusiast", "comment_text": "Loved t
he classic car review!"},
    {"comment_id": 19, "video_id": 19, "user_name": "FashionGuru", "comment_text": "Great win
ter fashion tips!"},
    {"comment_id": 20, "video_id": 20, "user_name": "HistoryBuff2", "comment_text": "Fascinat
ing documentary on Egypt!"},
```

```
    {"comment_id": 21, "video_id": 21, "user_name": "TechLover2", "comment_text": "Tablet rev
iew was spot on!"}
]
comments_collection.insert_many(comments)
```

Explanation

1. **Connecting to MongoDB**: The `MongoClient` is used to connect to a MongoDB instance running locally on the default port.
2. **Creating Database and Collections**: The database `video_platform` is created along with four collections: `channels`, `videos`, `playlists`, and `comments`.

This code effectively mirrors the SQL database structure and data in MongoDB, preserving relationships by using references (e.g., `channel_id`, `video_id`).

Fetching Data From MongoDB to Python:

Full Python Program:

Combining all the parts, we get the complete program:

```
from pymongo import MongoClient

# Connect to the MongoDB server
client = MongoClient("mongodb://localhost:27017/")

# Select the database
db = client.video_platform

# Collections
collections = {
    "Channels": db.channels,
    "Videos": db.videos,
    "Playlists": db.playlists,
    "Comments": db.comments
}

# Count of rows and sample documents from each collection
for name, collection in collections.items():
    count = collection.count_documents({})
    print(f"Count of rows in {name} collection: {count}")

    print(f"Sample documents from {name} collection:")
    sample = collection.find().limit(3)
    for doc in sample:
        print(doc)
    print("\\n")

# Perform a join between Channels and Videos using local and foreign key
pipeline = [
    {
        '$lookup': {
            'from': "videos",
            'localField': "channel_id",
            'foreignField': "channel_id",
            'as': 'videos'
        }
    },
    {
```

```
        '$limit': 3  # Show only 3 joined results for brevity
    }
]

joined_result = db.channels.aggregate(pipeline)

print("Joined data (Channels and Videos):")
for doc in joined_result:
    print(doc)
    print("\\n")
```

This program will:

1. Connect to the MongoDB server.
2. Select the `video_platform` database.
3. Loop through each collection to count the number of documents and show a sample of 3 documents.
4. Perform a join between the `channels` and `videos` collections on the `channel_id` field and display the results.

Explanation:

Part 1: Connecting to MongoDB

First, we need to connect to the MongoDB server.

```
from pymongo import MongoClient

# Connect to the MongoDB server
client = MongoClient("mongodb://localhost:27017/")
```

- `from pymongo import MongoClient`: Imports the `MongoClient` class from the `pymongo` library.
- `client = MongoClient("mongodb://localhost:27017/")`: Creates a connection to the MongoDB server running on `localhost` at the default port `27017`.

Part 2: Selecting the Database and Collections

Next, we select the database and the collections we will work with.

```
# Select the database
db = client.video_platform

# Collections
collections = {
    "Channels": db.channels,
    "Videos": db.videos,
    "Playlists": db.playlists,
    "Comments": db.comments
}
```

- `db = client.video_platform`: Selects the `video_platform` database.
- `collections`: A dictionary that maps collection names (as keys) to their corresponding collection objects in the database (as values).

Part 3: Counting Rows/documents and Showing Sample Documents

We loop through each collection to count the number of documents and show a sample of 3 documents.

```
# Count of rows and sample documents from each collection
for name, collection in collections.items():
```

```
    count = collection.count_documents({})
    print(f"Count of rows in {name} collection: {count}")

    print(f"Sample documents from {name} collection:")
    sample = collection.find().limit(3)
    for doc in sample:
        print(doc)
    print("\\n")
```

- `for name, collection in collections.items()`: Iterates over the `collections` dictionary.

To understand this part, let's assume that `collections` is a dictionary where the keys are the names of the collections and the values are the corresponding collection objects from the MongoDB database.

Example of the `collections` Dictionary

```
collections = {
    "Channels": db.Channels,
    "Videos": db.Videos,
    "Playlists": db.Playlists,
    "Comments": db.Comments
}
```

In this dictionary:

- `"Channels"`, `"Videos"`, `"Playlists"`, and `"Comments"` are the keys (names of the collections).
- `db.Channels`, `db.Videos`, `db.Playlists`, and `db.Comments` are the values (MongoDB collection objects).

The `for` Loop

Syntax:

```
for name, collection in collections.items():
```

- `collections.items()` returns a view object that displays a list of a dictionary's key-value tuple pairs.
- `name` will be assigned the key from each key-value pair (e.g., "Channels", "Videos").
- `collection` will be assigned the corresponding value (e.g., `db.Channels`, `db.Videos`).

Explanation:

1. **First Iteration:**
    - `name` will be `"Channels"`.
    - `collection` will be `db.Channels`.
2. **Second Iteration:**
    - `name` will be `"Videos"`.
    - `collection` will be `db.Videos`.

- `count = collection.count_documents({})`: Counts the number of documents in the current collection.
- `print(f"Count of rows in {name} collection: {count}")`: Prints the count of documents in the current collection.
- `sample = collection.find().limit(3)`: Retrieves a sample of 3 documents from the current collection.
- `for doc in sample`: Iterates over the sample documents and prints each document.

Part 4: Performing a Join Between Collections

We perform a join between the `channels` and `videos` collections using the `channel_id` field.

```
# Perform a join between Channels and Videos using local and foreign key
pipeline = [
    {
        '$lookup': {
            'from': "videos",
            'localField': "channel_id",
            'foreignField': "channel_id",
            'as': 'videos info'
        }
    },
    {
        '$limit': 3  # Show only 3 joined results for brevity
    }
]

joined_result = db.channels.aggregate(pipeline)

print("Joined data (Channels and Videos):")
for doc in joined_result:
    print(doc)
    print("\\n")
```

- `pipeline`: Defines the aggregation pipeline for the join operation.
  - `'$lookup'`: Performs the join.
    - `'from': "videos"`: Specifies the collection to join with (`videos`). Foreign table is referenced.
    - `'localField': "channel_id"`: The field from the `channels` collection.
    - `'foreignField': "channel_id"`: The field from the `videos` collection.
    - `'as': 'videos info'`: The name of the array field in the output documents that will contain the joined documents.
  - `'$limit': 3`: Limits the number of documents in the result to 3 for brevity.
- `joined_result = db.channels.aggregate(pipeline)`: Runs the aggregation pipeline on the `channels` collection.
- `for doc in joined_result`: Iterates over the joined result documents and prints each document.

Understanding Join in MongoDB

In SQL, a join combines columns from two or more tables based on a related column between them. For example, an `INNER JOIN` between the `Channels` and `Videos` tables would combine rows from both tables where there is a match on the `channel_id` column.

MongoDB `$lookup` Stage

In MongoDB, the `$lookup` stage in an aggregation pipeline performs a left outer join to combine documents from two collections. The fields specified in `localField` and `foreignField` are used to match documents from the input collection and the joined collection.

Example Collections

Let's assume you have the following documents in your `Channels` and `Videos` collections:

Channels Collection:

```
[
    { "channel_id": 1, "channel_name": "TechTalk", "description": "Technology and gadget revi
ews", "total_videos": 120, "subscribers": 1300000 },
    { "channel_id": 2, "channel_name": "DailyVlogs", "description": "Daily life vlogs and tra
vel", "total_videos": 250, "subscribers": 800000 },
    { "channel_id": 3, "channel_name": "CookingWithLove", "description": "Delicious recipes a
```

```
nd cooking tips", "total_videos": 300, "subscribers": 500000 }
]
```

Videos Collection:

```
[
    { "video_id": 1, "channel_id": 1, "title": "Latest Smartphone Review", "views": 10000, "l
ikes": 500 },
    { "video_id": 2, "channel_id": 2, "title": "A Day in My Life", "views": 8000, "likes": 30
0 },
    { "video_id": 3, "channel_id": 1, "title": "Laptop Review", "views": 11000, "likes": 550
}
]
```

Joining Collections with `$lookup`

Let's perform a join between `Channels` and `Videos` where the `channel_id` field matches.

Performing the Join Operation

We use the following MongoDB aggregation pipeline to join the `Channels` collection with the `Videos` collection:

```
from pymongo import MongoClient

# Connect to the MongoDB server
client = MongoClient("mongodb://localhost:27017/")

# Select the database
db = client.my_database

# Define the pipeline for the join
pipeline = [
    {
        '$lookup': {
            'from': "videos",          # Name of the collection to join
            'localField': "channel_id",# Field from the Channels collection
            'foreignField': "channel_id",# Field from the Videos collection
            'as': 'videos_info'        # Name for the array to store joined documents
        }
    }
]

# Execute the aggregation pipeline
joined_result = db.channels.aggregate(pipeline)

# Print the joined data
print("Joined data (Channels and Videos):")
for doc in joined_result:
    print(doc)
    print("\\n")
```

Explanation of the Pipeline

1. `$lookup` **Stage:**
    - `from`: Specifies the collection to join with (`videos` collection).
    - `localField`: Specifies the field from the input collection (`channels`) to match (`channel_id`).
    - `foreignField`: Specifies the field from the joined collection (`videos`) to match (`channel_id`).

- `as`: Specifies the name of the new array field to add to the input documents. This field will contain the matching documents from the joined collection.

2. `$limit` **Stage:**
   - Limits the result to 3 documents for brevity. This is optional and can be removed if you want to see all results.

Sample Output

Here is the sample output after performing the join operation:

```
{
    "_id": ObjectId("..."),
    "channel_id": 1,
    "channel_name": "TechTalk",
    "description": "Technology and gadget reviews",
    "total_videos": 120,
    "subscribers": 1300000,
    "videos_info": [
        { "video_id": 1, "channel_id": 1, "title": "Latest Smartphone Review", "views": 1000
0, "likes": 500 },
        { "video_id": 3, "channel_id": 1, "title": "Laptop Review", "views": 11000, "likes":
550 }
    ]
}
```

```
{
    "_id": ObjectId("..."),
    "channel_id": 2,
    "channel_name": "DailyVlogs",
    "description": "Daily life vlogs and travel",
    "total_videos": 250,
    "subscribers": 800000,
    "videos_info": [
        { "video_id": 2, "channel_id": 2, "title": "A Day in My Life", "views": 8000, "like
s": 300 }
    ]
}
```

```
{
    "_id": ObjectId("..."),
    "channel_id": 3,
    "channel_name": "CookingWithLove",
    "description": "Delicious recipes and cooking tips",
    "total_videos": 300,
    "subscribers": 500000,
    "videos_info": []
}
```

Explanation of the Sample Output

1. **Channel 1 (TechTalk):**
   - Has two videos associated with it: "Latest Smartphone Review" and "Laptop Review".
   - The `videos_info` array contains these two video documents.
2. **Channel 2 (DailyVlogs):**
   - Has one video associated with it: "A Day in My Life".

- The `videos_info` array contains this single video document.

3. **Channel 3 (CookingWithLove):**
   - Has no videos associated with it in the provided `Videos` collection.
   - The `videos_info` array is empty.

Summary

- The `$lookup` stage in the aggregation pipeline joins documents from the `Channels` collection with matching documents from the `Videos` collection based on the `channel_id` field.
- The resulting documents from the `Channels` collection include an additional `videos_info` array containing the related video documents.
- This joined output provides a way to see all videos associated with each channel in a single document, similar to how you would achieve this with a join in SQL.