Document created: 2025-06-01

(V.IMP) $lookup example

Let's use **orders** and **products** data for a detailed walkthrough of a MongoDB `$lookup` aggregation.

Your Sample Data

`orders` Collection

```
[
  { "_id": 1, "customerId": "C1", "productId": 101, "orderDate": "2024-06-10" },
  { "_id": 2, "customerId": "C2", "productId": 104, "orderDate": "2024-06-09" },
  { "_id": 3, "customerId": "C3", "productId": 102, "orderDate": "2024-06-08" },
  { "_id": 4, "customerId": "C2", "productId": 105, "orderDate": "2024-06-08" },
  { "_id": 5, "customerId": "C4", "productId": 106, "orderDate": "2024-06-07" }
]
```

`products` Collection

```
[
  { "_id": 101, "name": "Pencil",   "category": "Stationery" },
  { "_id": 102, "name": "Eraser",   "category": "Stationery" },
  { "_id": 103, "name": "Stapler",  "category": "Office" },
  { "_id": 104, "name": "Notebook", "category": "Stationery" },
  { "_id": 105, "name": "Mouse",    "category": "Electronics" }
]
```

Aggregation Query with `$lookup`

Let's combine product details with each order:

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",         // The collection we are joining from
      localField: "productId",  // Field in orders
      foreignField: "_id",      // Field in products
      as: "productInfo"         // Name of the new array field in the result
    }
  }
])
```

What Happens During the Query?

1. **Take one document from `orders`.**
2. **Look in `products`** for any document where `_id` equals the `orders` document's `productId`.
3. **Add a new field `productInfo`** to the `orders` document:
    - If there is a match, `productInfo` will be an array containing the matching `products` doc(s).
    - If there's no match (like for productId 106), `productInfo` will be an empty array.

Step-by-Step Example

For Order `_id: 1`

Order doc:

```
{ "_id": 1, "customerId": "C1", "productId": 101, "orderDate": "2024-06-10" }
```

Match in products:

```
{ "_id": 101, "name": "Pencil", "category": "Stationery" }
```

Result:

```
{
  "_id": 1,
  "customerId": "C1",
  "productId": 101,
  "orderDate": "2024-06-10",
  "productInfo": [
    { "_id": 101, "name": "Pencil", "category": "Stationery" }
  ]
}
```

For Order `_id: 5`

Order doc:

```
{ "_id": 5, "customerId": "C4", "productId": 106, "orderDate": "2024-06-07" }
```

No match in products (no product with `_id`: 106). Result:

```
{
  "_id": 5,
  "customerId": "C4",
  "productId": 106,
  "orderDate": "2024-06-07",
  "productInfo": []
}
```

Full Result for All Orders

```
[
  {
    "_id": 1,
    "customerId": "C1",
    "productId": 101,
    "orderDate": "2024-06-10",
    "productInfo": [
      { "_id": 101, "name": "Pencil", "category": "Stationery" }
    ]
  },
  {
    "_id": 2,
    "customerId": "C2",
    "productId": 104,
    "orderDate": "2024-06-09",
    "productInfo": [
      { "_id": 104, "name": "Notebook", "category": "Stationery" }
    ]
  },
  {
    "_id": 3,
    "customerId": "C3",
    "productId": 102,
    "orderDate": "2024-06-08",
    "productInfo": [
      { "_id": 102, "name": "Eraser", "category": "Stationery" }
    ]
  },
  {
    "_id": 4,
    "customerId": "C2",
    "productId": 105,
    "orderDate": "2024-06-08",
    "productInfo": [
      { "_id": 105, "name": "Mouse", "category": "Electronics" }
    ]
  },
  {
    "_id": 5,
    "customerId": "C4",
    "productId": 106,
    "orderDate": "2024-06-07",
    "productInfo": []
  }
]
```

What Can You Do Next?

- Want to **flatten results** so each order only includes the product info (not an array)?
  Use `$unwind`:

```
db.orders.aggregate([
 { $lookup: { from: "products", localField: "productId", foreignField: "_id", as: "productInfo" } },
 { $unwind: "$productInfo" }
```

-])
  - Now, only orders with a matching product remain (those with empty arrays are removed).
  - `productInfo` is now a simple object, not an array.

Summary of What You Learned

- **`$lookup`** in MongoDB is like a **left join** in SQL.
- The output always has your original doc plus an **array** named by `as` (`productInfo` here) containing all matching docs from the foreign collection (or empty if none).
- Orders with no matching products still appear, but their `productInfo` array is empty.
- Add `$unwind` if you want to filter only orders with matches, or flatten the array to an object.

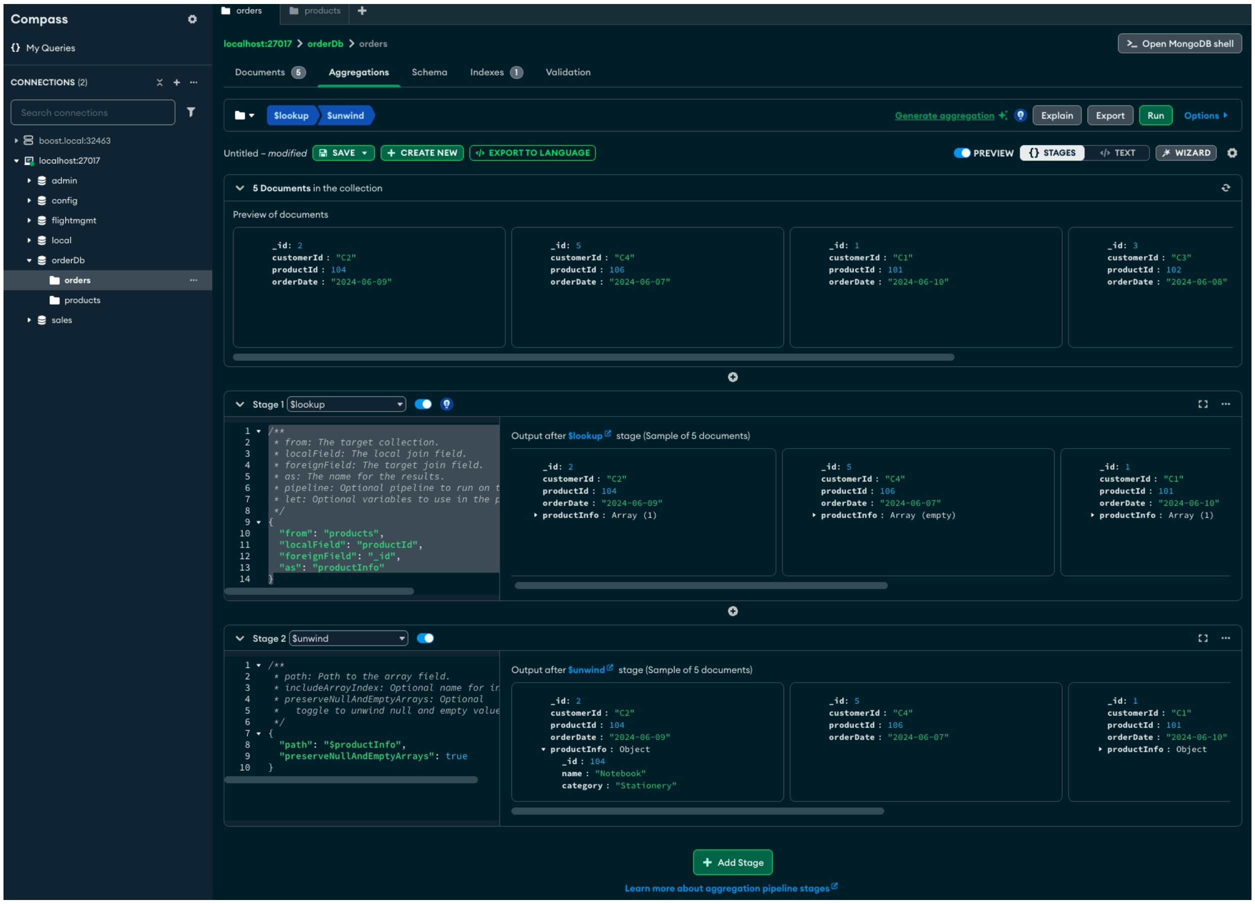
If you want to experiment more, try changing the field names, joining on something else, or see how `$unwind` affects the results. Let me know if you want further examples or practice exercises!

My Trial using MongoDB Atlas

Write the stage as follows(see the screenshot):

```
/**
 * from: The target collection.
 * localField: The local join field.
 * foreignField: The target join field.
 * as: The name for the results.
 * pipeline: Optional pipeline to run on the foreign collection.
 * let: Optional variables to use in the pipeline field stages.
 */
{
  "from": "products",
  "localField": "productId",
  "foreignField": "_id",
  "as": "productInfo"
}
```

```
/**
 * path: Path to the array field.
 * includeArrayIndex: Optional name for index.
 * preserveNullAndEmptyArrays: Optional
 *   toggle to unwind null and empty values.
 */
{
  "path": "$productInfo",
  "preserveNullAndEmptyArrays": true
}
```



5 Documents in the collection

Preview of documents

```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"

_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"

_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10"

_id: 3
customerId : "C3"
productId : 102
orderDate : "2024-06-08"
```

Stage 1 $lookup

```
/**
 * from: The target collection.
 * localField: The local join field.
 * foreignField: The target join field.
 * as: The name for the results.
 * pipeline: Optional pipeline to run on t
 * let: Optional variables to use in the p
 */
{
  "from": "products",
  "localField": "productId",
  "foreignField": "_id",
  "as": "productInfo"
}
```

Output after $lookup stage (Sample of 5 documents)

```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"
▸ productInfo : Array (1)

_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"
▸ productInfo : Array (empty)

_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10"
▸ productInfo : Array (1)
```

Stage 2 $unwind

```
/**
 * path: Path to the array field.
 * includeArrayIndex: Optional name for ir
 * preserveNullAndEmptyArrays: Optional
 *   toggle to unwind null and empty value
 */
{
  "path": "$productInfo",
  "preserveNullAndEmptyArrays": true
}
```

Output after $unwind stage (Sample of 5 documents)

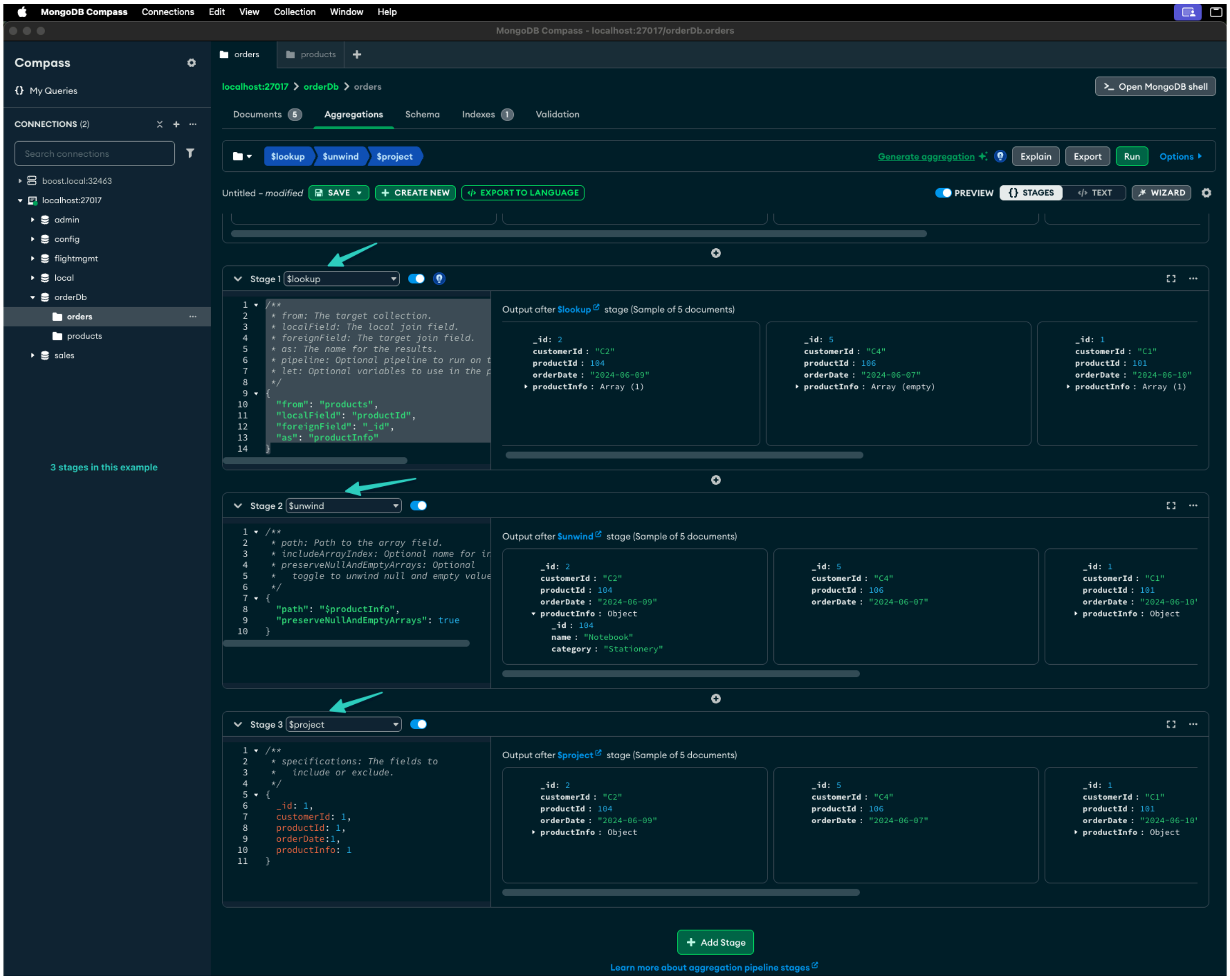
```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"
▾ productInfo : Object
  _id : 104
  name : "Notebook"
  category : "Stationery"

_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"

_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10"
▸ productInfo : Object
```

My Trial using Mongo shell

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "productId",
      foreignField: "_id",
      as: "productInfo"
    }
  },
  {
    $unwind: {
      path: "$productInfo",
      preserveNullAndEmptyArrays: true
    }
  }
])
```

Output in mongo shell:

```
[
  {
    _id: 2,
    customerId: 'C2',
    productId: 104,
    orderDate: '2024-06-09',
    productInfo: { _id: 104, name: 'Notebook', category: 'Stationery' }
  },
  { _id: 5, customerId: 'C4', productId: 106, orderDate: '2024-06-07' },
  {
    _id: 1,
    customerId: 'C1',
    productId: 101,
    orderDate: '2024-06-10',
    productInfo: { _id: 101, name: 'Pencil', category: 'Stationery' }
  },
  {
    _id: 3,
    customerId: 'C3',
    productId: 102,
    orderDate: '2024-06-08',
    productInfo: { _id: 102, name: 'Eraser', category: 'Stationery' }
  },
  {
    _id: 4,
    customerId: 'C2',
    productId: 105,
    orderDate: '2024-06-08',
    productInfo: { _id: 105, name: 'Mouse', category: 'Electronics' }
  }
]
```



Stage 1 $lookup

```
/**
 * from: The target collection.
 * localField: The local join field.
 * foreignField: The target join field.
 * as: The name for the results.
 * pipeline: Optional pipeline to run on t
 * let: Optional variables to use in the p
 */
{
  "from": "products",
  "localField": "productId",
  "foreignField": "_id",
  "as": "productInfo"
}
```

Output after $lookup stage (Sample of 5 documents)

```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"
productInfo : Array (1)
```

```
_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"
productInfo : Array (empty)
```

```
_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10"
productInfo : Array (1)
```

Stage 2 $unwind

```
/**
 * path: Path to the array field.
 * includeArrayIndex: Optional name for ir
 * preserveNullAndEmptyArrays: Optional
 *   toggle to unwind null and empty value
 */
{
  "path": "$productInfo",
  "preserveNullAndEmptyArrays": true
}
```

Output after $unwind stage (Sample of 5 documents)

```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"
productInfo : Object
  _id : 104
  name : "Notebook"
  category : "Stationery"
```

```
_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"
```

```
_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10'
productInfo : Object
```

Stage 3 $project

```
/**
 * specifications: The fields to
 *   include or exclude.
 */
{
  _id: 1,
  customerId: 1,
  productId: 1,
  orderDate:1,
  productInfo: 1
}
```

Output after $project stage (Sample of 5 documents)

```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"
productInfo : Object
```

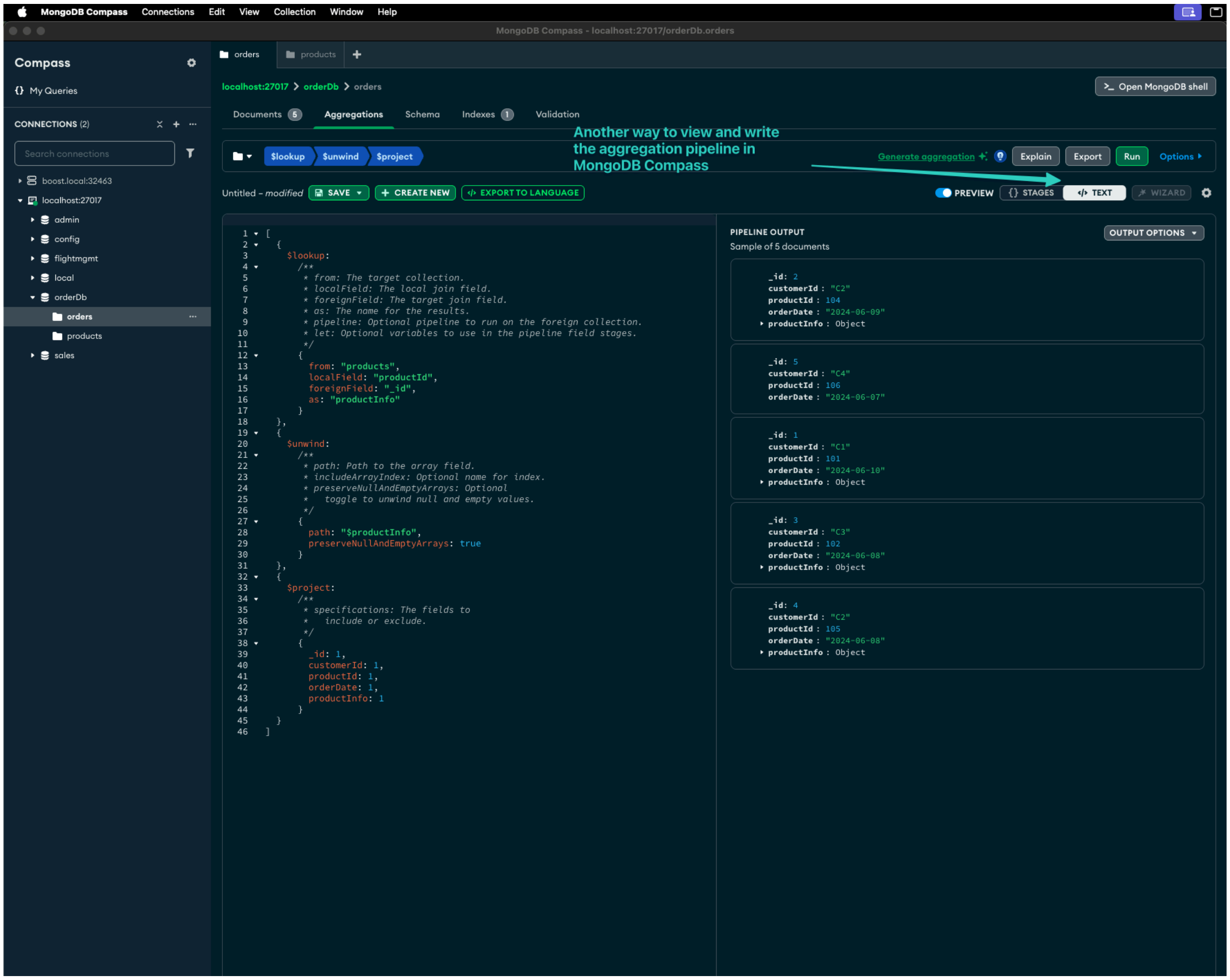
```
_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"
```

```
_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10'
productInfo : Object
```

```
/**
 * specifications: The fields to
 *   include or exclude.
 */
{
  _id: 1,
  customerId: 1,
  productId: 1,
  orderDate:1,
  productInfo: 1
}
```

Using projection in this example. See above.

(Imp) Yet another way to write the stage in MongoDB Compass



Another way to view and write the aggregation pipeline in MongoDB Compass

```
[
  {
    $lookup:
      /**
       * from: The target collection.
       * localField: The local join field.
       * foreignField: The target join field.
       * as: The name for the results.
       * pipeline: Optional pipeline to run on the foreign collection.
       * let: Optional variables to use in the pipeline field stages.
       */
      {
        from: "products",
        localField: "productId",
        foreignField: "_id",
        as: "productInfo"
      }
  },
  {
    $unwind:
      /**
       * path: Path to the array field.
       * includeArrayIndex: Optional name for index.
       * preserveNullAndEmptyArrays: Optional
       *   toggle to unwind null and empty values.
       */
      {
        path: "$productInfo",
        preserveNullAndEmptyArrays: true
      }
  },
  {
    $project:
      /**
       * specifications: The fields to
       *   include or exclude.
       */
      {
        _id: 1,
        customerId: 1,
        productId: 1,
        orderDate: 1,
        productInfo: 1
      }
  }
]
```

```
_id: 2
customerId : "C2"
productId : 104
orderDate : "2024-06-09"
productInfo : Object
```

```
_id: 5
customerId : "C4"
productId : 106
orderDate : "2024-06-07"
```

```
_id: 1
customerId : "C1"
productId : 101
orderDate : "2024-06-10"
productInfo : Object
```

```
_id: 3
customerId : "C3"
productId : 102
orderDate : "2024-06-08"
productInfo : Object
```

```
_id: 4
customerId : "C2"
productId : 105
orderDate : "2024-06-08"
productInfo : Object
```

In Mongo shell it would be like this:

```
db.orders.aggregate([
  { $lookup: {
      from: "products",
      localField: "productId",
      foreignField: "_id",
      as: "productInfo"
    }
  },
  { $unwind: {
      path: "$productInfo",
      preserveNullAndEmptyArrays: true
    }
  },
  { $project: {
      _id: 1,
      customerId: 1,
      productId: 1,
      orderDate:1,
      productInfo: 1
    }
  }
])
```

Below is a practical example of an aggregation pipeline using MongoDB that employs the `$unwind`, `$group`, `$match`, `$project`, and `$sort` stages. I'll first provide example JSON objects and then walk you through the aggregation process along with the expected results.

Example JSON Objects

Let's create a collection named `orders` which holds customer orders, each having a list of items purchased:

```
[
  {
    "orderId": 1,
    "customer": "Alice",
    "items": [
      {
        "name": "Laptop",
        "quantity": 1
      },
      {
        "name": "Mouse",
        "quantity": 2
      }
    ],
    "totalPrice": 1200,
    "status": "shipped"
  },
  {
    "orderId": 2,
    "customer": "Bob",
    "items": [
      {
        "name": "Keyboard",
        "quantity": 2
      },
      {
        "name": "Monitor",
        "quantity": 1
      }
    ],
    "totalPrice": 400,
    "status": "processing"
  },
  {
    "orderId": 3,
    "customer": "Alice",
    "items": [
      {
        "name": "Monitor",
        "quantity": 1
      },
      {
        "name": "Mouse",
        "quantity": 2
      }
    ],
    "totalPrice": 350,
    "status": "shipped"
  },
  {
    "orderId": 4,
    "customer": "Charlie",
    "items": [
      {
        "name": "Laptop",
        "quantity": 2
      }
    ],
    "totalPrice": 2400,
    "status": "canceled"
  },
  {
    "orderId": 5,
    "customer": "Bob",
    "items": [
      {
        "name": "Mouse",
        "quantity": 3
      },
      {
        "name": "Keyboard",
        "quantity": 2
      }
    ],
    "totalPrice": 150,
    "status": "shipped"
  }
]
```

Aggregation Pipeline

Objective

Find out how many units of each product have been shipped and sort the results in descending order of quantity.

```
db.orders.aggregate([
  {
    $unwind: "$items"
  },
  {
    $match: { "status": "shipped" }
  },
  {
    $group: {
      _id: "$items.name",
      totalQuantity: { $sum: "$items.quantity" }
    }
  },
  {
    $sort: { totalQuantity: -1 }
  },
  {
    $project: {
      _id: 0,
      productName: "$_id",
      totalQuantity: 1
    }
  }
]);
```

Explanation

1. **`$unwind`**: Deconstructs the `items` array field, with each item becoming a separate document for processing.

2. **`$match`**: Filters the orders to include only those with a `status` of `"shipped"`.

3. **`$group`**: Groups documents by the `name` of each item, accumulating the `totalQuantity` for each grouped item.

4. **`$sort`**: Sorts the grouped results in descending order based on `totalQuantity`.

5. **`$project`**: Reshapes the documents to include only the `productName` and `totalQuantity`, renaming the `_id` field to `productName` for clarity.

Expected Results

Based on the aggregation pipeline, the expected output will look like this:

```
[
  { "productName": "Mouse", "totalQuantity": 4 },
  { "productName": "Laptop", "totalQuantity": 1 },
  { "productName": "Monitor", "totalQuantity": 1 }
]
```

Explanation of Results

- **Mouse**: 2 units from order 1 and 2 units from order 3, total = 4.
- **Laptop**: 1 unit from order 1, total = 1.
- **Monitor**: 1 unit from order 3, total = 1.

This example demonstrates combining various aggregation operations to perform a common data analysis task, which includes filtering, transforming, and summarizing your dataset. If you need further details or adjustments, feel free to ask!

(IMP) But when you unwind then you might get multiple results with same value from the left side

Exactly! You've identified a key behavior of `$unwind` in MongoDB.

What Happens When You Unwind After `$lookup`?

When using `$lookup`, the resulting field (e.g., `productInfo`) is **always an array**, even if it contains 0, 1, or multiple matched documents.

When you use `$unwind` on that array:

- If the array has **multiple elements**, you get **one output document for each element** in the array.
- **The left-side (original) fields are duplicated** for each matching document from the "right" (foreign) collection.

Example

Suppose after `$lookup` you get:

```
{
  "_id": 1,
  "customerId": "C1",
  "productId": 101,
  "productInfo": [
    { "_id": 101, "name": "Pencil" },
    { "_id": 101, "name": "Pencil (Updated)" }
  ]
}
```

After `$unwind: "$productInfo"`:

```
{
  "_id": 1,
  "customerId": "C1",
  "productId": 101,
  "productInfo": { "_id": 101, "name": "Pencil" }
}
{
  "_id": 1,
  "customerId": "C1",
  "productId": 101,
  "productInfo": { "_id": 101, "name": "Pencil (Updated)" }
}
```

- **One original document, two joined matches ⇒ two output docs, both with the same left-side values.**

Why is this useful or important?

- This is how **MongoDB achieves a “1:N” join** (one-to-many).
  If your foreign collection has multiple matches for a left doc’s key, every match leads to a new result document.

If You Want Only One Joined Row Per Left-Side Document

- **If you expect only one or zero match:**
  - `$unwind` works fine (at most one result per left doc).
- **If you want only the first match:**
  - Use `$unwind` **with option** `{ "preserveNullAndEmptyArrays": true }` and perhaps `$project` to keep only the first element, *or*
  - Use the `$arrayElemAt` operator after lookup to select just the first element from the array (without using `$unwind`).

Example:

```
{
  "$project": {
    "customerId": 1,
    "productId": 1,
    "productInfo": { "$arrayElemAt": ["$productInfo", 0] }
  }
}
```

Summary

- `$unwind` “explodes” an array into multiple docs, duplicating the rest of the fields.
- If multiple matches exist in the joined collection, you get multiple output rows per original document.
- If you want only one, use further aggregation stages (like `$project` + `$arrayElemAt`).

Let me know if you'd like an example pipeline that **guarantees only one result per left-side document**, or if you want to handle or aggregate over multiple joined rows!

(IMP) Yes please give an example using Orders and products along with example data and the result

Absolutely!
Let’s walk through the **Orders and Products** example, show how `$lookup` with possible multiple matches works, and how to guarantee only **one result per left-side document** using `$arrayElemAt`.

Sample Data

orders

```
[
  { "_id": 1, "customerId": "C1", "productId": 101 },
  { "_id": 2, "customerId": "C2", "productId": 102 },
  { "_id": 3, "customerId": "C3", "productId": 103 }
]
```

products

```
[
  { "_id": 101, "name": "Pencil", "category": "Stationery" },
  { "_id": 101, "name": "Pencil (Updated)", "category": "Education" },
  { "_id": 102, "name": "Eraser", "category": "Stationery" }
]
```

Notice: There are **two products** with `_id: 101`.

Plain `$lookup` Output

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "productId",
      foreignField: "_id",
      as: "productInfo"
    }
  }
])
```

Result:

```
[
  {
    "_id": 1,
    "customerId": "C1",
    "productId": 101,
    "productInfo": [
      { "_id": 101, "name": "Pencil", "category": "Stationery" },
      { "_id": 101, "name": "Pencil (Updated)", "category": "Education" }
    ]
  },
  {
    "_id": 2,
    "customerId": "C2",
    "productId": 102,
    "productInfo": [
      { "_id": 102, "name": "Eraser", "category": "Stationery" }
    ]
  },
  {
    "_id": 3,
    "customerId": "C3",
    "productId": 103,
    "productInfo": []       // No match!
  }
]
```

Using `$unwind` (Multiple Docs for Multiple Matches)

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "productId",
      foreignField: "_id",
      as: "productInfo"
    }
  },
  { $unwind: { path: "$productInfo", preserveNullAndEmptyArrays: true } }
])
```

Result:

```
[
  {
    "_id": 1,
    "customerId": "C1",
    "productId": 101,
    "productInfo": { "_id": 101, "name": "Pencil", "category": "Stationery" }
  },
  {
    "_id": 1,
    "customerId": "C1",
    "productId": 101,
    "productInfo": { "_id": 101, "name": "Pencil (Updated)", "category": "Education" }
  },
  {
    "_id": 2,
    "customerId": "C2",
    "productId": 102,
    "productInfo": { "_id": 102, "name": "Eraser", "category": "Stationery" }
  },
  {
    "_id": 3,
    "customerId": "C3",
    "productId": 103,
    "productInfo": null     // No match
  }
]
```

Order #1 appears twice, once per product match.

Using `$arrayElemAt` to Guarantee *One Result per Order*

To take **only the first match**:

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "productId",
      foreignField: "_id",
      as: "productInfo"
    }
  },
  {
    $project: {
      customerId: 1,
      productId: 1,
      productInfo: { $arrayElemAt: [ "$productInfo", 0 ] }
    }
  }
])
```

Result:

```
[
  {
    "_id": 1,
    "customerId": "C1",
    "productId": 101,
    "productInfo": { "_id": 101, "name": "Pencil", "category": "Stationery" }
  },
  {
    "_id": 2,
    "customerId": "C2",
    "productId": 102,
    "productInfo": { "_id": 102, "name": "Eraser", "category": "Stationery" }
  },
  {
    "_id": 3,
    "customerId": "C3",
    "productId": 103,
    "productInfo": null   // No match: array was empty
  }
]
```

Now:

- Each order appears **at most once**
- `productInfo` is the **first matching product**, or `null` if no match

Key Takeaways

- `$unwind`: One result for each match (can duplicate left-side docs).
- `$arrayElemAt` after `$lookup`: One result per left-side doc, with the first match (or `null`).
- Use `$project` to reshape and reduce fields.

Let me know if you need further customization, want a random match instead of the first, or need to handle all matches differently!