TUS

THE TECHNOLOGICAL UNIVERSITY OF THE SHANNON: MIDLANDS MIDWEST

Student Name:	**Prajwal Bharti**
Student Id Number:	**A00325600**
Course:	**Masters in Data Analytics**
Title of Assignment:	**Advanced Databases Project**
Date:	**01 May 2025**

Declaration

I hereby certify that the material, which is submitted in this report towards the award of MSc. in Data Analytics, is entirely my own work and has not been submitted for any academic assessment other than part fulfilment of the above named award.

Signed Prajwal Vithal Bharti Date 01/05/2025

1 Table of Contents

2. Project Overview, Database Setup, Collection Creation and Importation.....5

2.1 Project Overview.....5

2.2 Database Setup & Collection Creation.....5

2.3 Verifying and Exploring the Data.....5

2.4 Arrays and their Values.....6

3. Mongo Shell – Simple Queries.....8

Query 1.....8

Query 2.....10

Query 3.....12

Query 4.....14

Query 5.....16

Query 6.....18

Query 7.....20

Query 8.....22

Query 9.....24

Query 10.....26

Query 11.....28

Query 12.....30

Query 13.....32

Query 14.....34

Query 15.....36

Query 16.....38

Query 17.....40

Query 18.....42

Query 19.....44

Query 20.....46

4. Mongo Shell – Aggregation Framework.....48

Query 1.....48

Query 2.....50

Query 3.....52
Query 4.....54
Query 5.....56
Query 6.....58
Query 7.....60
Query 8.....62
Query 9.....64
5. MongoDB Compass – Overview and Simple Queries.....66
5.1 Overview.....66
5.2 Connecting to MongoDB.....67
5.3 Simple Queries in Compass.....68
5.4 Aggreagtion Queries in Compass.....69
5.5 Other features explored in Compass.....71
6.Appendix.....74
6.1 YouTube Links.....74
6.2 Reference Links.....74

2 Project Overview, Database Setup, Collection Creation and Data Importation

2.1 Project Overview

This project explores concert data using MongoDB. The core goal of this project is to query, analyze, and manipulate a dataset that contains concert events using MongoDB's shell and graphical tools (Compass/Atlas). The dataset includes details like concert year, location, performer's information, audience demographics, sponsorship details, merchandise, ticket pricing, crowd statistics, and more. A total of 1000+ documents have been imported into a MongoDB collection named Concert.

2.2 Database Setup & Collection Creation

The MongoDB environment was prepared locally and also accessed using MongoDB Compass and Atlas. The main collection **Concert** was created under the database named **Concert_Db**.

Database Creation Command:

```
test> use Concert_Db
switched to db Concert_Db
Concert_Db>
```

Data Importation:

A .json file containing over 1000 documents was imported using the MongoDB import tool:

```
C:\Users\Prajwal\OneDrive - TUS MM\Documents\MS_PB_Advanced_Databases\MongoDb Project>mongoimport --db Concert_Db --collection Concert
 --file Concert.json --jsonArray --drop
2025-04-13T01:36:37.402+0100    connected to: mongodb://localhost/
2025-04-13T01:36:37.403+0100    dropping: Concert_Db.Concert
2025-04-13T01:36:37.443+0100    1000 document(s) imported successfully. 0 document(s) failed to import.
```

After import checking and verifying the document count in MongoDb Shell

```
Concert_Db> db.Concert.countDocuments()
1000
```

2.3 Verifying and Exploring the Data

To confirm the structure and successfully imported data, the first document was retrieved and displayed:

```
Concert_Db> db.Concert.find({_id : 1}).pretty()
[
  {
    _id: 1,
    Concert_Year: 2010,
    Concert_Month: 5,
    Duration: 90,
    Expected_Crowd: 19900,
    Actual_Crowd: 18500,
    Cheapest_Ticket_Price: 25,
    Dearest_Ticket_Price: 65,
    Food_Sales: 52000,
    Drink_Sales: 69100,
    Atmosphere: 'Poor',
    Police_Cost: 15000,
    Performer: {
      Name: 'Torin McGrath',
      Music_Type: 'Hip-Hop',
      Services: [ 'Fan Engagement', 'Photo Shoots ', 'Meet and Greet' ]
    },
    Sponsor: {
      Name: 'Jane Dolan',
      Amount: 160000,
      Services: [ 'Ticket Sales', 'Posters', 'News Papers' ]
    },
    Stadium: {
      Name: 'Semple Stadium',
      City: 'Thurles',
      Type: 'GAA',
      Capacity: 45690,
      Acoustics: 'Fair'
    },
    Merchandise_Type: [ 'CD's', 'Umbrellas ' ],
    Audience_Type: [ 'Third Level Students', 'Retired ' ]
  }
]
```

2.4 Arrays and their values

Performer.Services:

- "TV Interviews"
- "Photo Shoots"
- "Live Rehearsals"

Sponsor.Services:

- "Radio Interviews"
- "Posters"
- "Online Ads"

Merchandise_Type:

- "CD's"
- "Photos"
- "Tee Shirts"
- "Umbrellas"

Audience_Type:

- "Teenagers"
- "Third Level Students"
- "Middle Aged"
- "Senior Citizens"

3 Mongo Shell – Simple Queries

In this section I have written basic queries tested using the MongoDB Shell. Each query begins with a question for the query, followed by the actual command used to retrieve the data. Where appropriate, countDocuments() is included to verify the number of matches, and projection is used to display only relevant fields. Outputs for the query were manually verified using the shell.

Query 1:

Question:
Find concerts held in 2022, with an audience size over 20,000, where the atmosphere was "Excellent", and the music type was "Rock".

```
db.Concert.countDocuments({
  Concert_Year: 2022,
  Actual_Crowd: { $gt: 20000 },
  Atmosphere: "Excellent",
  "Performer.Music_Type": "Rock"
})
db.Concert.find({
    Concert_Year: 2022,
    Actual_Crowd: { $gt: 20000 },
    Atmosphere: "Excellent",
    "Performer.Music_Type": "Rock"
 })
```

```
Concert_Db> db.Concert.countDocuments({
...   Concert_Year: 2022,
...   Actual_Crowd: { $gt: 20000 },
...   Atmosphere: "Excellent",
...   "Performer.Music_Type": "Rock"
... })
...
2
Concert_Db>
```

```
Concert_Db> db.Concert.find({
...     Concert_Year: 2022,
...     Actual_Crowd: { $gt: 20000 },
...     Atmosphere: "Excellent",
...     "Performer.Music_Type": "Rock"
...  })
...
[
  {
    _id: 741,
    Concert_Year: 2022,
    Concert_Month: 4,
    Duration: 150,
    Expected_Crowd: 30400,
    Actual_Crowd: 32600,
    Cheapest_Ticket_Price: 35,
    Dearest_Ticket_Price: 170,
    Food_Sales: 22200,
    Drink_Sales: 100200,
    Atmosphere: 'Excellent',
    Police_Cost: 7400,
    Performer: {
      Name: 'Aoibhinn O'Leary',
      Music_Type: 'Rock',
      Services: [
        'Meet and Greet',
        'TV Interviews',
        'Fan Engagement',
        'Photo Shoots '
      ]
    },
    Sponsor: {
      Name: 'Pete  Ralph',
      Amount: 27000,
      Services: [ 'Ticket Sales', 'TV Interviews' ]
    },
    Stadium: {
      Name: 'Kingspan Stadium',
      City: 'Belfast',
      Type: 'Rugby',
      Capacity: 18000,
      Acoustics: 'Fair'
    },
    Merchandise_Type: [ 'Umbrellas ' ],
    Audience_Type: [ 'Teenagers', 'Third Level Students' ]
  },
  {
    _id: 927,
    Concert_Year: 2022,
    Concert_Month: 8,
    Duration: 105,
    Expected_Crowd: 30800,
    Actual_Crowd: 29700,
    Cheapest_Ticket_Price: 50,
    Dearest_Ticket_Price: 135,
    Food_Sales: 95600,
    Drink_Sales: 15800,
    Atmosphere: 'Excellent',
    Police_Cost: 40200,
    Performer: {
      Name: 'Nessa O'Dwyer',
      Music_Type: 'Rock',
      Services: [ 'Live Rehearsals', 'Fan Engagement', 'Photo Shoots ' ]
    },
    Sponsor: { Name: 'Lyn Nagle', Amount: 40000, Services: [ 'TV Interviews' ] },
    Stadium: {
      Name: 'Bord Gais Energy Theatre',
      City: 'Dublin',
      Type: 'Basketball',
      Capacity: 2100,
      Acoustics: 'Good'
    },
    Merchandise_Type: [ 'Tee Shirts', 'Umbrellas ' ],
    Audience_Type: [ 'Teenagers', 'Third Level Students' ]
  }
]
Concert_Db>
```

This query filters out only the big, energetic rock concerts from the year 2022 that had a large crowd and were enjoyed by the audience size over 20000. I used both a filter and a count to see how many such concerts there were. In this first query, I didn't use projection I wanted to see the full document structure to understand the data better. After this in all queries, I've used projections to only show relevant fields.

Query 2

Question:
Find concerts held in Dublin in 2023, where police cost is under €15,000, and sponsor amount is over €120,000.

```
db.Concert.countDocuments({
  "Stadium.City": "Dublin",
  Concert_Year: 2023,
  Police_Cost: { $lt: 15000 },
  "Sponsor.Amount": { $gt: 120000 }
})
db.Concert.find(
  {
    "Stadium.City": "Dublin",
    Concert_Year: 2023,
    Police_Cost: { $lt: 15000 },
    "Sponsor.Amount": { $gt: 120000 }
  },
  {
    Concert_Year: 1,
    Police_Cost: 1,
    "Sponsor.Amount": 1,
    "Stadium.Name": 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   "Stadium.City": "Dublin",
...   Concert_Year: 2023,
...   Police_Cost: { $lt: 15000 },
...   "Sponsor.Amount": { $gt: 120000 }
... })
...
8
Concert_Db>
```

```
Concert_Db> db.Concert.find(
...   {
...     "Stadium.City": "Dublin",
...     Concert_Year: 2023,
...     Police_Cost: { $lt: 15000 },
...     "Sponsor.Amount": { $gt: 120000 }
...   },
...   {
...     Concert_Year: 1,
...     Police_Cost: 1,
...     "Sponsor.Amount": 1,
...     "Stadium.Name": 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2023,
    Police_Cost: 13300,
    Sponsor: { Amount: 405000 },
    Stadium: { Name: 'Aviva Stadium' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 8600,
    Sponsor: { Amount: 324000 },
    Stadium: { Name: 'Olympia Theatre' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 8400,
    Sponsor: { Amount: 431000 },
    Stadium: { Name: 'Bord Gais Energy Theatre' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 1600,
    Sponsor: { Amount: 304000 },
    Stadium: { Name: 'Aviva Stadium' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 8000,
    Sponsor: { Amount: 481000 },
    Stadium: { Name: 'Olympia Theatre' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 3200,
    Sponsor: { Amount: 207000 },
    Stadium: { Name: 'National Concert Hall' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 10300,
    Sponsor: { Amount: 350000 },
    Stadium: { Name: 'Croke Park' }
  },
  {
    Concert_Year: 2023,
    Police_Cost: 9200,
    Sponsor: { Amount: 178000 },
    Stadium: { Name: 'Tallaght Stadium' }
  }
]
Concert_Db>
```

Here, I have tried to identify well-sponsored concerts in Dublin that managed to keep their police security costs low. This could be useful for figuring out which events were efficient in terms of security spending. I also used projection to only show the fields related to the question like the year, costs, and sponsor information.

Query 3

Question:
Find concerts where the performer offers only "TV Interviews", Drink Sales are over €90,000, and the Stadium Acoustics is “Good”.

```
db.Concert.countDocuments({
                "Performer.Services": ["TV Interviews"],
                 Drink_Sales: {$gt: 90000},
                "Stadium.Acoustics": "Good"
})
db.Concert.find(
        {
                "Performer.Services": ["TV Interviews"],
                 Drink_Sales: {$gt: 90000},
                "Stadium.Acoustics": "Good"
        },
        {
                "Performer.Name": 1,
                  Drink_Sales: 1,
                  "Stadium.Acoustics": 1,
                _id : 0
        }
)
```

```
Concert_Db> db.Concert.countDocuments({
... "Performer.Services": ["TV Interviews"],
...  Drink_Sales: {$gt: 90000},
... "Stadium.Acoustics": "Good"
... })
...
9
Concert_Db> db.Concert.find(
... {
... "Performer.Services": ["TV Interviews"],
...  Drink_Sales: {$gt: 90000},
... "Stadium.Acoustics": "Good"
... },
... {
... "Performer.Name": 1,
...   Drink_Sales: 1,
...   "Stadium.Acoustics": 1,
... _id : 0
... }
... )
...
[
  {
    Drink_Sales: 101400,
    Performer: { Name: 'Eoin O'Farrell' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 110900,
    Performer: { Name: 'Senan O'Brien' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 104600,
    Performer: { Name: 'Seamus Donovan' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 94100,
    Performer: { Name: 'Conal Foley' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 106600,
    Performer: { Name: 'Clodagh Gallagher' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 117400,
    Performer: { Name: 'Clodagh Moore' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 112700,
    Performer: { Name: 'Conal Lynch' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 95700,
    Performer: { Name: 'Fionn Doyle' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    Drink_Sales: 106500,
    Performer: { Name: 'Eoin O'Farrell' },
    Stadium: { Acoustics: 'Good' }
  }
]
Concert_Db>
```

In this query, I am showing concerts where the performers focused purely on media coverage through TV interviews and where the drinks were selling well possibly lively events. Also, good acoustics might hint at higher-quality venues. I used a projection to show just the performer's name, drink sales, and acoustics the parts directly related to the question. I used an exact match array for "Performer.Services" because the requirement of the query was that the performer only offers TV interviews, and no other service.

Query 4

Question:

Find concerts with Duration ≥ 150 minutes, held in Cork, and Sponsor Amount ≥ €200,000.

```
db.Concert.countDocuments({
  Duration: { $gte: 150 },
  "Stadium.City": "Cork",
  "Sponsor.Amount": { $gte: 200000 }
})
db.Concert.find(
  {
    Duration: { $gte: 150 },
    "Stadium.City": "Cork",
    "Sponsor.Amount": { $gte: 200000 }
  },
  {
    Duration: 1,
    "Stadium.City": 1,
    "Sponsor.Amount": 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Duration: { $gte: 150 },
...   "Stadium.City": "Cork",
...   "Sponsor.Amount": { $gte: 200000 }
... })
...
11
Concert_Db>
```

```
Concert_Db> db.Concert.find(
...   {
...     Duration: { $gte: 150 },
...     "Stadium.City": "Cork",
...     "Sponsor.Amount": { $gte: 200000 }
...   },
...   {
...     Duration: 1,
...     "Stadium.City": 1,
...     "Sponsor.Amount": 1,
...     _id: 0
...   }
... )
...
[
  {
    Duration: 170,
    Sponsor: { Amount: 320000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 155,
    Sponsor: { Amount: 379000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 175,
    Sponsor: { Amount: 486000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 160,
    Sponsor: { Amount: 430000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 180,
    Sponsor: { Amount: 335000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 150,
    Sponsor: { Amount: 315000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 155,
    Sponsor: { Amount: 235000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 170,
    Sponsor: { Amount: 203000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 160,
    Sponsor: { Amount: 286000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 170,
    Sponsor: { Amount: 205000 },
    Stadium: { City: 'Cork' }
  },
  {
    Duration: 170,
    Sponsor: { Amount: 359000 },
    Stadium: { City: 'Cork' }
  }
]
```

In This query I have showed concerts over 2.5 hours particularly in the city of Cork that also had strong financial support. I wanted to see if bigger sponsor amounts were associated with longer events. I included only the duration, city name, and sponsor amount in the output using projection. The use of $gte ensures that concerts lasting exactly 150 minutes are included as well.

Query 5

Question:

List concerts where the performer only offers *TV Interviews*, the *Police_Cost* is between 40000 and 50000, and the *Stadium Type* where the performance is "Basketball". Display performer name, police cost, and stadium type.

```
db.Concert.countDocuments({
   "Performer.Services": ["TV Interviews"],
   Police_Cost: { $gte: 40000, $lte: 50000 },
   "Stadium.Type": "Basketball"
})
```

```
db.Concert.find(
  {
   "Performer.Services": ["TV Interviews"],
   Police_Cost: { $gte: 40000, $lte: 50000 },
   "Stadium.Type": "Basketball"
  },
  {
   "Performer.Name": 1,
   Police_Cost: 1,
   "Stadium.Type": 1,
   _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...     "Performer.Services": ["TV Interviews"],
...     Police_Cost: { $gte: 40000, $lte: 50000 },
...     "Stadium.Type": "Basketball"
... })
...
5
Concert_Db> db.Concert.find(
...   {
...     "Performer.Services": ["TV Interviews"],
...     Police_Cost: { $gte: 40000, $lte: 50000 },
...     "Stadium.Type": "Basketball"
...   },
...   {
...     "Performer.Name": 1,
...     Police_Cost: 1,
...     "Stadium.Type": 1,
...     _id: 0
...   }
... )
...
[
  {
    Police_Cost: 47900,
    Performer: { Name: 'Diarmuid Walsh' },
    Stadium: { Type: 'Basketball' }
  },
  {
    Police_Cost: 50000,
    Performer: { Name: 'Conchur Duffy' },
    Stadium: { Type: 'Basketball' }
  },
  {
    Police_Cost: 40900,
    Performer: { Name: 'Deirbhile Ward' },
    Stadium: { Type: 'Basketball' }
  },
  {
    Police_Cost: 44800,
    Performer: { Name: 'Ultan O'Mahony' },
    Stadium: { Type: 'Basketball' }
  },
  {
    Police_Cost: 47900,
    Performer: { Name: 'Darragh MacDonald' },
    Stadium: { Type: 'Basketball' }
  }
]
Concert_Db>
```

Here, In this query I'm looking for concerts that fall into a very specific combination: limited performer services (just TV interviews), decent-sized police budgets (not too low or too high), and held in basketball stadiums. These kinds of events might be mid-scale but still tightly managed. $gte means "greater than or equal to" and $lte means "less than or equal to". Together they allow us to get result in a range for the police cost with including those exact values.

Query 6

Question:

Display concerts where *Merchandise_Type* includes "CD's" and the *Sponsor Amount* is below 70000, and the concert was held in the month of May (5). Display sponsor name, amount, and merchandise types.

```
db.Concert.countDocuments({
   Merchandise_Type: "CD's",
   "Sponsor.Amount": { $lt: 70000 },
   Concert_Month: 5
})
db.Concert.find(
  {
   Merchandise_Type: "CD's",
   "Sponsor.Amount": { $lt: 70000 },
   Concert_Month: 5
  },
  {
   "Sponsor.Name": 1,
   "Sponsor.Amount": 1,
   Merchandise_Type: 1
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...     Merchandise_Type: "CD's",
...     "Sponsor.Amount": { $lt: 70000 },
...     Concert_Month: 5
... })
...
5
Concert_Db> db.Concert.find(
...   {
...     Merchandise_Type: "CD's",
...     "Sponsor.Amount": { $lt: 70000 },
...     Concert_Month: 5
...   },
...   {
...     "Sponsor.Name": 1,
...     "Sponsor.Amount": 1,
...     Merchandise_Type: 1
...   }
... )
...
[
  {
    _id: 302,
    Sponsor: { Name: 'Tina Prone', Amount: 55000 },
    Merchandise_Type: [ 'CD's', 'Tee Shirts' ]
  },
  {
    _id: 376,
    Sponsor: { Name: 'Eric Duffy', Amount: 49000 },
    Merchandise_Type: [ 'CD's' ]
  },
  {
    _id: 496,
    Sponsor: { Name: 'Ann Breen', Amount: 45000 },
    Merchandise_Type: [ 'CD's', 'Photos', 'Tee Shirts', 'Umbrellas ' ]
  },
  {
    _id: 542,
    Sponsor: { Name: 'Tara Noone', Amount: 61000 },
    Merchandise_Type: [ 'CD's', 'Tee Shirts' ]
  },
  {
    _id: 660,
    Sponsor: { Name: 'Matt Roche', Amount: 27000 },
    Merchandise_Type: [ 'CD's', 'Tee Shirts' ]
  }
]
Concert_Db>
```

In this query I am focusing on showing smaller-budget events happening in the month of May where CDs were part of the merchandise maybe smaller or indie-style concerts. I wanted to get a sense of which sponsors were still supporting these kinds of shows. Here I am directly matching Merchandise_Type to CD's we don't need to explicilty use $in operator. MongoDB checks if "CD's" is present anywhere in the array.

Query 7

Question:
Find concerts where the Performer Services include either "Photo Shoots" or "Live Rehearsals", the Expected Crowd is below 5000, and the Cheapest Ticket Price is above 55. Show performer name, ticket price, and crowd.

```
db.Concert.countDocuments({
  $or: [
    { "Performer.Services": "Photo Shoots " },
    { "Performer.Services": "Live Rehearsals" }
  ],
  Expected_Crowd: { $lt: 5000 },
  Cheapest_Ticket_Price: { $gt: 55 }
})
```

```
db.Concert.find(
  {
    $or: [
      { "Performer.Services": "Photo Shoots " },
      { "Performer.Services": "Live Rehearsals" }
    ],
    Expected_Crowd: { $lt: 5000 },
    Cheapest_Ticket_Price: { $gt: 55 }
  },
  {
    "Performer.Name": 1,
     Expected_Crowd: 1,
    Cheapest_Ticket_Price: 1
  }
)
```

In this query, I am trying to find smaller concerts with fewer people, where the ticket price was still a bit high. Also, I wanted to check if the performer is offering something extra like photo shoots or rehearsals not just singing. These might be more special events. I used $or because I want either "Photo Shoots" or "Live Rehearsals" to be in the services not both, just at least one of them.

```
Concert_Db> db.Concert.countDocuments({
...   $or: [
...     { "Performer.Services": "Photo Shoots " },
...     { "Performer.Services": "Live Rehearsals" }
...   ],
...   Expected_Crowd: { $lt: 5000 },
...   Cheapest_Ticket_Price: { $gt: 55 }
... })
6
Concert_Db> db.Concert.find(
...   {
...     $or: [
...       { "Performer.Services": "Photo Shoots " },
...       { "Performer.Services": "Live Rehearsals" }
...     ],
...     Expected_Crowd: { $lt: 5000 },
...     Cheapest_Ticket_Price: { $gt: 55 }
...   },
...   {
...     "Performer.Name": 1,
...      Expected_Crowd: 1,
...     Cheapest_Ticket_Price: 1
...   }
... )
...
[
  {
    _id: 184,
    Expected_Crowd: 2500,
    Cheapest_Ticket_Price: 60,
    Performer: { Name: 'Aine Thompson' }
  },
  {
    _id: 243,
    Expected_Crowd: 3600,
    Cheapest_Ticket_Price: 60,
    Performer: { Name: 'Meabh MacNamara' }
  },
  {
    _id: 412,
    Expected_Crowd: 3200,
    Cheapest_Ticket_Price: 60,
    Performer: { Name: 'Dearbhla Sheehan' }
  },
  {
    _id: 454,
    Expected_Crowd: 4300,
    Cheapest_Ticket_Price: 60,
    Performer: { Name: 'Ultan Fitzgerald' }
  },
  {
    _id: 689,
    Expected_Crowd: 4600,
    Cheapest_Ticket_Price: 60,
    Performer: { Name: 'Cian MacKenna' }
  },
  {
    _id: 724,
    Expected_Crowd: 1300,
    Cheapest_Ticket_Price: 60,
    Performer: { Name: 'Naomhan McLoughlin' }
  }
]
Concert_Db>
```

Query 8

Question:
Find concerts where Food Sales are above €98,000, the Stadium Acoustics is "Good", and the Performer does not offer "TV Interviews".

```
db.Concert.countDocuments({
  Food_Sales: { $gt: 98000 },
  "Stadium.Acoustics": "Good",
  "Performer.Services": { $not: { $in: ["TV Interviews"] } }
})
```

```
db.Concert.find(
  {
    Food_Sales: { $gt: 98000 },
    "Stadium.Acoustics": "Good",
    "Performer.Services": { $not: { $in: ["TV Interviews"] } }
  },
  {
    "Performer.Name": 1,
    "Stadium.Acoustics": 1,
    Food_Sales: 1
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Food_Sales: { $gt: 98000 },
...   "Stadium.Acoustics": "Good",
...   "Performer.Services": { $not: { $in: ["TV Interviews"] } }
... })
...
5
Concert_Db> db.Concert.find(
...   {
...     Food_Sales: { $gt: 98000 },
...     "Stadium.Acoustics": "Good",
...     "Performer.Services": { $not: { $in: ["TV Interviews"] } }
...   },
...   {
...     "Performer.Name": 1,
...     "Stadium.Acoustics": 1,
...     Food_Sales: 1
...   }
... )
...
[
  {
    _id: 561,
    Food_Sales: 98100,
    Performer: { Name: 'Nessa O'Donnell' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    _id: 660,
    Food_Sales: 98600,
    Performer: { Name: 'Clodagh Burns' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    _id: 704,
    Food_Sales: 99300,
    Performer: { Name: 'Bébinn Smith' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    _id: 790,
    Food_Sales: 98800,
    Performer: { Name: 'Daithi Donnelly' },
    Stadium: { Acoustics: 'Good' }
  },
  {
    _id: 884,
    Food_Sales: 98200,
    Performer: { Name: 'Cathair Fitzgerald' },
    Stadium: { Acoustics: 'Good' }
  }
]
Concert_Db>
```

Here, In this query I was looking for concerts where food sold really well, the stadium sound was good, and the performer wasn't doing TV interviews. Basically, I wanted to see events that didn't rely on media like TV interviews but still had a strong vibe and crowd. This means: the performer's list of services should not include "TV Interviews". MongoDB checks if "TV Interviews" is inside the array, and we tell it not to match it using $not function. I have used $in function to directly match the TV interviews in the performer services array.

Query 9

Question:
Find concerts where Audience_Type includes both "Teenagers" and "Third Level Students", and Sponsor Amount is strictly between €112,000 and €122,000. Display sponsor name, amount, and audience type.

```
db.Concert.countDocuments({
  Audience_Type: { $all: ["Teenagers", "Third Level Students"] },
  "Sponsor.Amount": { $gt: 112000, $lt: 122000 }
})
```

```
db.Concert.find(
  {
    Audience_Type: { $all: ["Teenagers", "Third Level Students"] },
    "Sponsor.Amount": { $gt: 112000, $lt: 122000 }
  },
  {
    "Sponsor.Name": 1,
    "Sponsor.Amount": 1,
    Audience_Type: 1,
    _id : 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Audience_Type: { $all: ["Teenagers", "Third Level Students"] },
...   "Sponsor.Amount": { $gt: 112000, $lt: 122000 }
... })
...
4
Concert_Db> db.Concert.find(
...   {
...     Audience_Type: { $all: ["Teenagers", "Third Level Students"] },
...     "Sponsor.Amount": { $gt: 112000, $lt: 122000 }
...   },
...   {
...     "Sponsor.Name": 1,
...     "Sponsor.Amount": 1,
...     Audience_Type: 1,
...     _id : 0
...   }
... )
...
[
  {
    Sponsor: { Name: 'Brian Oates', Amount: 115000 },
    Audience_Type: [ 'Teenagers', 'Third Level Students', 'Young Working Adults' ]
  },
  {
    Sponsor: { Name: 'Joey Webb', Amount: 121000 },
    Audience_Type: [ 'Teenagers', 'Third Level Students', 'Young Working Adults' ]
  },
  {
    Sponsor: { Name: 'Pat Feehy', Amount: 117000 },
    Audience_Type: [
      'Teenagers',
      'Third Level Students',
      'Young Working Adults',
      'Retired '
    ]
  },
  {
    Sponsor: { Name: 'Mags Crowe', Amount: 116000 },
    Audience_Type: [
      'Teenagers',
      'Third Level Students',
      'Young Working Adults',
      'Middle Aged'
    ]
  }
]
Concert_Db>
```

In this query, I wanted to find concerts where the target audience was both teenagers and students, and were backed by sponsors giving a medium-to-high budget. It's useful to see which shows were aimed at younger crowds and how much money the sponsors were putting in this concerts. I used $all to check that both "Teenagers" and "Third Level Students" are in the array.
So MongoDB will only match documents that include both audience types not just one.

Query 10 – Limit Use Case

Question:
Show any 3 concerts that occurred in March, had "CD's" in their merchandise, and were held in stadiums with "Good" acoustics. Display stadium name, merchandise type, and concert month.

```
db.Concert.countDocuments({
  "Stadium.Acoustics": "Good",
  Concert_Month: 3,
  Merchandise_Type: "CD's"
})
```

```
db.Concert.find(
  {
    "Stadium.Acoustics": "Good",
    Concert_Month: 3,
    Merchandise_Type: "CD's"
  },
  {
    "Stadium.Name": 1,
    Merchandise_Type: 1,
    Concert_Month: 1,
    _id: 0
  }
).limit(3)
```

```
Concert_Db> db.Concert.countDocuments({
...   "Stadium.Acoustics": "Good",
...   Concert_Month: 3,
...   Merchandise_Type: "CD's"
... })
...
23
Concert_Db> db.Concert.find(
...   {
...     "Stadium.Acoustics": "Good",
...     Concert_Month: 3,
...     Merchandise_Type: "CD's"
...   },
...   {
...     "Stadium.Name": 1,
...     Merchandise_Type: 1,
...     Concert_Month: 1,
...     _id: 0
...   }
... ).limit(3)
...
[
  {
    Concert_Month: 3,
    Stadium: { Name: '3Arena' },
    Merchandise_Type: [ 'CD's' ]
  },
  {
    Concert_Month: 3,
    Stadium: { Name: 'Bord Gais Energy Theatre' },
    Merchandise_Type: [ 'CD's', 'Photos', 'Tee Shirts' ]
  },
  {
    Concert_Month: 3,
    Stadium: { Name: 'Bord Gais Energy Theatre' },
    Merchandise_Type: [ 'CD's' ]
  }
]
Concert_Db> S
```

In this query I wanted to show the usecase of **limit** function. Limit function limits the query output documents upto given number. In this query you can see the 23 documents are returned by the query but by using limit upto 3 query returns the first 3 documents.

This query was more like a quick check just to grab any 3 concerts that meet the conditions. I wanted to see if there were any well-sounding March concerts that sold CDs. This helps spot common trends in a certain time of year.

Query 11

Question:
List concerts from 2023 where the Atmosphere is neither "Excellent" nor the Stadium Type is "Basketball", but Drink Sales exceeded €100,000.

```
db.Concert.countDocuments({
  $nor: [
    { Atmosphere: "Excellent" },
    { "Stadium.Type": "Basketball" }
  ],
  Drink_Sales: { $gt: 100000 },
  Concert_Year: 2023
})
```

```
db.Concert.find(
  {
    $nor: [
      { Atmosphere: "Excellent" },
      { "Stadium.Type": "Basketball" }
    ],
    Drink_Sales: { $gt: 100000 },
    Concert_Year: 2023
  },
  {
    Atmosphere: 1,
    "Stadium.Type": 1,
    Drink_Sales: 1,
    Concert_Year: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   $nor: [
...     { Atmosphere: "Excellent" },
...     { "Stadium.Type": "Basketball" }
...   ],
...   Drink_Sales: { $gt: 100000 },
...   Concert_Year: 2023
... })
...
3
Concert_Db> db.Concert.find(
...   {
...     $nor: [
...       { Atmosphere: "Excellent" },
...       { "Stadium.Type": "Basketball" }
...     ],
...     Drink_Sales: { $gt: 100000 },
...     Concert_Year: 2023
...   },
...   {
...     Atmosphere: 1,
...     "Stadium.Type": 1,
...     Drink_Sales: 1,
...     Concert_Year: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2023,
    Drink_Sales: 101300,
    Atmosphere: 'Good',
    Stadium: { Type: 'GAA' }
  },
  {
    Concert_Year: 2023,
    Drink_Sales: 101900,
    Atmosphere: 'Good',
    Stadium: { Type: 'GAA' }
  },
  {
    Concert_Year: 2023,
    Drink_Sales: 109300,
    Atmosphere: 'Fair',
    Stadium: { Type: 'GAA' }
  }
]
Concert_Db>
```

In this query, I am checking the concerts that still had strong drink sales, even though they didn't happen in basketball stadiums and didn't have the best-rated atmosphere. It helps find events that performed well even without perfect conditions.

Here I have used **$nor** function to neglect both the conditions nor excellent atmosphere neither in the basketball stadium.

Query 12

Question:
Find concerts from 2023 where Drink Sales exceed €70,000, the Stadium Type is neither "Basketball" nor "Soccer", and the Atmosphere is "Good".

```
db.Concert.countDocuments({
  Concert_Year: 2023,
  Drink_Sales: { $gt: 70000 },
  "Stadium.Type": { $nin: ["Basketball", "Soccer"] },
  Atmosphere: "Good"
})
db.Concert.find(
  {
    Concert_Year: 2023,
    Drink_Sales: { $gt: 70000 },
    "Stadium.Type": { $nin: ["Basketball", "Soccer"] },
    Atmosphere: "Good"
  },
  {
    Concert_Year: 1,
    Drink_Sales: 1,
    "Stadium.Type": 1,
    Atmosphere: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Concert_Year: 2023,
...   Drink_Sales: { $gt: 70000 },
...   "Stadium.Type": { $nin: ["Basketball", "Soccer"] },
...   Atmosphere: "Good"
... })
...
4
Concert_Db> db.Concert.find(
...   {
...     Concert_Year: 2023,
...     Drink_Sales: { $gt: 70000 },
...     "Stadium.Type": { $nin: ["Basketball", "Soccer"] },
...     Atmosphere: "Good"
...   },
...   {
...     Concert_Year: 1,
...     Drink_Sales: 1,
...     "Stadium.Type": 1,
...     Atmosphere: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2023,
    Drink_Sales: 101300,
    Atmosphere: 'Good',
    Stadium: { Type: 'GAA' }
  },
  {
    Concert_Year: 2023,
    Drink_Sales: 101900,
    Atmosphere: 'Good',
    Stadium: { Type: 'GAA' }
  },
  {
    Concert_Year: 2023,
    Drink_Sales: 97500,
    Atmosphere: 'Good',
    Stadium: { Type: 'Rugby' }
  },
  {
    Concert_Year: 2023,
    Drink_Sales: 98300,
    Atmosphere: 'Good',
    Stadium: { Type: 'GAA' }
  }
]
Concert_Db>
```

In This query I was looking for good-quality concerts in 2023 that weren't held in particular stadium types like soccer or basketball. I also wanted to see if drink sales were still high in those less typical venues.

For this query I have used new operator **$nin** which is similar to not in it just don't count those array elements when $nin is used.

Query 13

Question:
Find concerts held in October, with Atmosphere exactly "Fair", hosted in Dublin, where "CD's" were sold, and the Performer Music Type is "Folk".

```
db.Concert.countDocuments({
  Atmosphere: { $eq: "Fair" },
  Concert_Month: { $eq: 10 },
  "Stadium.City": { $eq: "Dublin" },
  Merchandise_Type: "CD's",
  "Performer.Music_Type": "Folk"
})
```

```
db.Concert.find(
  {
    Atmosphere: { $eq: "Fair" },
    Concert_Month: { $eq: 10 },
    "Stadium.City": { $eq: "Dublin" },
    Merchandise_Type: "CD's",
    "Performer.Music_Type": "Folk"
  },
  {
    "Stadium.City": 1,
    Atmosphere: 1,
    Concert_Month: 1,
    "Performer.Music_Type": 1,
    Merchandise_Type: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Atmosphere: { $eq: "Fair" },
...   Concert_Month: { $eq: 10 },
...   "Stadium.City": { $eq: "Dublin" },
...   Merchandise_Type: "CD's",
...   "Performer.Music_Type": "Folk"
... })
...
2
Concert_Db> db.Concert.find(
...   {
...     Atmosphere: { $eq: "Fair" },
...     Concert_Month: { $eq: 10 },
...     "Stadium.City": { $eq: "Dublin" },
...     Merchandise_Type: "CD's",
...     "Performer.Music_Type": "Folk"
...   },
...   {
...     "Stadium.City": 1,
...     Atmosphere: 1,
...     Concert_Month: 1,
...     "Performer.Music_Type": 1,
...     Merchandise_Type: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Month: 10,
    Atmosphere: 'Fair',
    Performer: { Music_Type: 'Folk' },
    Stadium: { City: 'Dublin' },
    Merchandise_Type: [ 'CD's' ]
  },
  {
    Concert_Month: 10,
    Atmosphere: 'Fair',
    Performer: { Music_Type: 'Folk' },
    Stadium: { City: 'Dublin' },
    Merchandise_Type: [ 'CD's', 'Photos', 'Tee Shirts', 'Hoodies' ]
  }
]
Concert_Db>
```

In this query I am focusing on a very particular type of concert small or average events in Dublin, which has folk music and had sold CD's specifically. It's useful to see how common or rare these calm, folk-style shows were during that time.
In this one, I've used **$eq** function which means "equal to". It is used to check if a field is exactly equal to a specific value. In this query I wanted atmosphere exactly equal to fair and no other options included.

Query 14

Question:
Find concerts from 2021 where the Atmosphere was not "Excellent", the Sponsor contributed less than €80,000, the Audience included "Middle Aged" individuals, and the Cheapest Ticket Price was over €30.

```
db.Concert.countDocuments({
  Concert_Year: 2021,
  Atmosphere: { $ne: "Excellent" },
  "Sponsor.Amount": { $lt: 80000 },
  Audience_Type: "Middle Aged",
  Cheapest_Ticket_Price: { $gt: 30 }
})
```

```
db.Concert.find(
  {
    Concert_Year: 2021,
    Atmosphere: { $ne: "Excellent" },
    "Sponsor.Amount": { $lt: 80000 },
    Audience_Type: "Middle Aged",
    Cheapest_Ticket_Price: { $gt: 30 }
  },
  {
    Concert_Year: 1,
    Atmosphere: 1,
    "Sponsor.Name": 1,
    "Sponsor.Amount": 1,
    Audience_Type: 1,
    Cheapest_Ticket_Price: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Concert_Year: 2021,
...   Atmosphere: { $ne: "Excellent" },
...   "Sponsor.Amount": { $lt: 80000 },
...   Audience_Type: "Middle Aged",
...   Cheapest_Ticket_Price: { $gt: 30 }
... })
...
4
Concert_Db> db.Concert.find(
...   {
...     Concert_Year: 2021,
...     Atmosphere: { $ne: "Excellent" },
...     "Sponsor.Amount": { $lt: 80000 },
...     Audience_Type: "Middle Aged",
...     Cheapest_Ticket_Price: { $gt: 30 }
...   },
...   {
...     Concert_Year: 1,
...     Atmosphere: 1,
...     "Sponsor.Name": 1,
...     "Sponsor.Amount": 1,
...     Audience_Type: 1,
...     Cheapest_Ticket_Price: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2021,
    Cheapest_Ticket_Price: 60,
    Atmosphere: 'Fair',
    Sponsor: { Name: 'Sean Green', Amount: 60000 },
    Audience_Type: [ 'Middle Aged', 'Retired ', 'Teenagers', 'Third Level Students' ]
  },
  {
    Concert_Year: 2021,
    Cheapest_Ticket_Price: 45,
    Atmosphere: 'Good',
    Sponsor: { Name: 'Lorna Dowd', Amount: 70000 },
    Audience_Type: [
      'Young Working Adults',
      'Middle Aged',
      'Retired ',
      'Third Level Students'
    ]
  },
  {
    Concert_Year: 2021,
    Cheapest_Ticket_Price: 45,
    Atmosphere: 'Poor',
    Sponsor: { Name: 'Susan Barry', Amount: 49000 },
    Audience_Type: [ 'Middle Aged', 'Retired ' ]
  },
  {
    Concert_Year: 2021,
    Cheapest_Ticket_Price: 50,
    Atmosphere: 'Fair',
    Sponsor: { Name: 'Matt Boyle', Amount: 12000 },
    Audience_Type: [ 'Young Working Adults', 'Third Level Students', 'Middle Aged' ]
  }
]
Concert_Db>
```

In this query I introduced new function as **$ne** which is basically opposite to the **$eq** function it checks the given values and dosen't include that values for eg, It will select all the documents where atmosphere is not excellent.

In this one, I wanted to check concerts that were aimed at middle-aged people but didn't have a top-rated vibe or a big sponsor. Still, the ticket price was over €30, so maybe it wasn't a low-budget event. This mix helps spot mid-range concerts.

Query 15

Question:
Find concerts held in 2023, where:

- Both Food Sales and Drink Sales fields exist
- Police Cost is greater than €20,000, Audience includes "Third Level Students"
- The Stadium Acoustics is "Good"

```
db.Concert.countDocuments({
  Concert_Year: 2023,
  Food_Sales: { $exists: true },
  Drink_Sales: { $exists: true },
  Police_Cost: { $gt: 20000 },
  "Stadium.Acoustics": "Good",
  Audience_Type: "Third Level Students"
})
db.Concert.find(
  {
    Concert_Year: 2023,
    Food_Sales: { $exists: true },
    Drink_Sales: { $exists: true },
    Police_Cost: { $gt: 20000 },
    "Stadium.Acoustics": "Good",
    Audience_Type: "Third Level Students"
  },
  {
    Concert_Year: 1,
    Food_Sales: 1,
    Drink_Sales: 1,
    Police_Cost: 1,
    "Stadium.Acoustics": 1,
    Audience_Type: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Concert_Year: 2023,
...   Food_Sales: { $exists: true },
...   Drink_Sales: { $exists: true },
...   Police_Cost: { $gt: 20000 },
...   "Stadium.Acoustics": "Good",
...   Audience_Type: "Third Level Students"
... })
...
5
Concert_Db> db.Concert.find(
...   {
...     Concert_Year: 2023,
...     Food_Sales: { $exists: true },
...     Drink_Sales: { $exists: true },
...     Police_Cost: { $gt: 20000 },
...     "Stadium.Acoustics": "Good",
...     Audience_Type: "Third Level Students"
...   },
...   {
...     Concert_Year: 1,
...     Food_Sales: 1,
...     Drink_Sales: 1,
...     Police_Cost: 1,
...     "Stadium.Acoustics": 1,
...     Audience_Type: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2023,
    Food_Sales: 63200,
    Drink_Sales: 9000,
    Police_Cost: 28300,
    Stadium: { Acoustics: 'Good' },
    Audience_Type: [ 'Teenagers', 'Third Level Students' ]
  },
  {
    Concert_Year: 2023,
    Food_Sales: 29300,
    Drink_Sales: 111500,
    Police_Cost: 27800,
    Stadium: { Acoustics: 'Good' },
    Audience_Type: [ 'Middle Aged', 'Retired ', 'Teenagers', 'Third Level Students' ]
  },
  {
    Concert_Year: 2023,
    Food_Sales: 61800,
    Drink_Sales: 99800,
    Police_Cost: 47200,
    Stadium: { Acoustics: 'Good' },
    Audience_Type: [
      'Teenagers',
      'Third Level Students',
      'Young Working Adults',
      'Retired '
    ]
  },
  {
    Concert_Year: 2023,
    Food_Sales: 12600,
    Drink_Sales: 86700,
    Police_Cost: 45100,
    Stadium: { Acoustics: 'Good' },
    Audience_Type: [ 'Third Level Students', 'Retired ' ]
  },
  {
    Concert_Year: 2023,
    Food_Sales: 24400,
    Drink_Sales: 19700,
    Police_Cost: 46000,
    Stadium: { Acoustics: 'Good' },
    Audience_Type: [ 'Teenagers', 'Third Level Students', 'Retired ' ]
  }
]
Concert_Db>
```

In this query I introduced new function as **$exists,** $exists: true checks if a field is present in the document. Food_Sales: { $exists: true } means: the concert must have food sales data recorded. $exists helps in situation where we don't try to use a field that's missing from some documents.

In This query I was checks for concerts that were fully recorded (with both food and drink sales), had decent crowd control (police cost), concerts that were attended by students, and held in a stadium with good sound. Basically, I was checking for well-managed and properly logged events with a young audience.

Query 16

Question:
Find concerts from the year 2022 where the sponsor's services field is stored as an array, the drink sales are stored as a numeric type, the merchandise includes either "Photos" or "Tee Shirts", the stadium is located in Cork, Galway, or Thurles, and the cheapest ticket price is greater than €30.

```
db.Concert.countDocuments({
  Concert_Year: 2022,
  "Sponsor.Services": { $type: "array" },
  Drink_Sales: { $type: "number" },
  Merchandise_Type: { $in: ["Photos", "Tee Shirts"] },
  "Stadium.City": { $in: ["Cork", "Galway", "Thurles"] },
  Cheapest_Ticket_Price: { $gt: 30 }
})
```

```
db.Concert.find(
  {
    Concert_Year: 2022,
    "Sponsor.Services": { $type: "array" },
    Drink_Sales: { $type: "number" },
    Merchandise_Type: { $in: ["Photos", "Tee Shirts"] },
    "Stadium.City": { $in: ["Cork", "Galway", "Thurles"] },
    Cheapest_Ticket_Price: { $gt: 30 }
  },
  {
    Concert_Year: 1,
    "Sponsor.Services": 1,
    Drink_Sales: 1,
    Merchandise_Type: 1,
    "Stadium.City": 1,
    Cheapest_Ticket_Price: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   Concert_Year: 2022,
...   "Sponsor.Services": { $type: "array" },
...   Drink_Sales: { $type: "number" },
...   Merchandise_Type: { $in: ["Photos", "Tee Shirts"] },
...   "Stadium.City": { $in: ["Cork", "Galway", "Thurles"] },
...   Cheapest_Ticket_Price: { $gt: 30 }
... })
...
3
Concert_Db> db.Concert.find(
...   {
...     Concert_Year: 2022,
...     "Sponsor.Services": { $type: "array" },
...     Drink_Sales: { $type: "number" },
...     Merchandise_Type: { $in: ["Photos", "Tee Shirts"] },
...     "Stadium.City": { $in: ["Cork", "Galway", "Thurles"] },
...     Cheapest_Ticket_Price: { $gt: 30 }
...   },
...   {
...     Concert_Year: 1,
...     "Sponsor.Services": 1,
...     Drink_Sales: 1,
...     Merchandise_Type: 1,
...     "Stadium.City": 1,
...     Cheapest_Ticket_Price: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2022,
    Cheapest_Ticket_Price: 35,
    Drink_Sales: 60700,
    Sponsor: { Services: [ 'Ticket Sales', 'TV Interviews' ] },
    Stadium: { City: 'Thurles' },
    Merchandise_Type: [ 'CD's', 'Photos', 'Tee Shirts' ]
  },
  {
    Concert_Year: 2022,
    Cheapest_Ticket_Price: 35,
    Drink_Sales: 24500,
    Sponsor: {
      Services: [ 'TV Interviews', 'Posters', 'News Papers', 'Ticket Sales' ]
    },
    Stadium: { City: 'Galway' },
    Merchandise_Type: [ 'Hoodies', 'Umbrellas ', 'Photos' ]
  },
  {
    Concert_Year: 2022,
    Cheapest_Ticket_Price: 35,
    Drink_Sales: 46200,
    Sponsor: { Services: [ 'Radio Interviews' ] },
    Stadium: { City: 'Thurles' },
    Merchandise_Type: [ 'Photos', 'Umbrellas ' ]
  }
]
Concert_Db>
```

In this query I introduced a new function called **$type**.
$type checks the data type of a field. For example:"Sponsor.Services": { $type: "array" } means the sponsor services must be saved as an array. Drink_Sales: { $type: "number" } means the drink sales value must be a number (not a string or text).This function is useful when we want to make sure our data is in the right format, especially before running calculations or comparisons.

In this query I was checking concerts from 2022 that had clean and correct data. I wanted Concerts where sponsor services were stored properly as arrays, drink sales were saved as numbers, and they sold either photos or t-shirts. I also focused on concerts in three cities Cork, Galway, and Thurles and filtered records shows where the ticket price was comparitively too low.

Query 17

Question:
Retrieve concerts held in either 2022 or 2023 where the Performer's Music Type is "Classical" or the Stadium Acoustics are rated "Poor", the Cheapest Ticket Price is below €15, and the results should include the concert year, performer details, stadium acoustics, and the cheapest ticket price.

db.Concert.countDocuments(

```
{ $and: [{
    $or: [
      { "Performer.Music_Type": "Classical" },
      { "Stadium.Acoustics": "Poor" }
    ]},
   { Cheapest_Ticket_Price: { $lt: 15 } },
   { Concert_Year: { $in: [2022, 2023] } }
 ]
})
```

db.Concert.find(

```
{
 $and: [{
    $or: [
      { "Performer.Music_Type": "Classical" },
      { "Stadium.Acoustics": "Poor" }
    ]},
   { Cheapest_Ticket_Price: { $lt: 15 } },
   { Concert_Year: { $in: [2022, 2023] } }
 ]
},
{
 "Performer.Name": 1,
 "Performer.Music_Type": 1,
 "Stadium.Acoustics": 1,
 Cheapest_Ticket_Price: 1,
 Concert_Year: 1,
 _id: 0
})
```

```
Concert_Db> db.Concert.countDocuments({
...   $and: [{
...       $or: [
...         { "Performer.Music_Type": "Classical" },
...         { "Stadium.Acoustics": "Poor" }
...       ]},
...     { Cheapest_Ticket_Price: { $lt: 15 } },
...     { Concert_Year: { $in: [2022, 2023] } }
...   ]
... })
...
2
Concert_Db> db.Concert.find(
...   {
...     $and: [{
...         $or: [
...           { "Performer.Music_Type": "Classical" },
...           { "Stadium.Acoustics": "Poor" }
...         ]},
...       { Cheapest_Ticket_Price: { $lt: 15 } },
...       { Concert_Year: { $in: [2022, 2023] } }
...     ]
...   },
...   {
...     "Performer.Name": 1,
...     "Performer.Music_Type": 1,
...     "Stadium.Acoustics": 1,
...     Cheapest_Ticket_Price: 1,
...     Concert_Year: 1,
...     _id: 0
...   }
... )
...
[
  {
    Concert_Year: 2022,
    Cheapest_Ticket_Price: 10,
    Performer: { Name: 'Colm McGrath', Music_Type: 'Classical' },
    Stadium: { Acoustics: 'Fair' }
  },
  {
    Concert_Year: 2023,
    Cheapest_Ticket_Price: 10,
    Performer: { Name: 'Eachann Burke', Music_Type: 'Classical' },
    Stadium: { Acoustics: 'Fair' }
  }
]
Concert_Db>
```

In this query I introduced a new function called $and combined with $or.

- **$or** is used when we want either one of the condition from both to be true
- **$and** is used when we want all the conditions inside it to be true together.

In this query The $or is used to check if the music type is "Classical" or the acoustics is "Poor".Then I have used $and to make sure that those matched documents also had a cheap ticket price (under €15) and were from 2022 or 2023.

In This query I was checking concerts that were either more traditional (Classical) or had poor sound quality, and the tickets were cheap. I limited the results to recent years. I wanted to see what kind of concerts were low-cost and whether they had anything in common.

Query 18

Question:
Find concerts where the Sponsor has a "Name" field, and the name includes “Media” (case-insensitive). Also, the Concert duration should be 120 minutes or more.

```
db.Concert.countDocuments(
  {
    "Sponsor.Name": { $exists: true, $regex: /^Mary|^Ann|Sam/i },
    Duration: { $gte: 100, $lte: 130 },
    Atmosphere: "Excellent"
  }
)
db.Concert.find(
  {
    "Sponsor.Name": { $exists: true, $regex: /^Mary|Ann|Sam/i },
    Duration: { $gte: 100, $lte: 130 },
    Atmosphere: "Excellent"
  },
  {
    "Sponsor.Name": 1,
    Duration: 1,
    Atmosphere: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments(
...   {
...     "Sponsor.Name": { $exists: true, $regex: /^Mary|Ann|Sam/i },
...     Duration: { $gte: 100, $lte: 130 },
...     Atmosphere: "Excellent"
...   }
... )
...
10
Concert_Db> db.Concert.find(
...   {
...     "Sponsor.Name": { $exists: true, $regex: /^Mary|Ann|Sam/i },
...     Duration: { $gte: 100, $lte: 130 },
...     Atmosphere: "Excellent"
...   },
...   {
...     "Sponsor.Name": 1,
...     Duration: 1,
...     Atmosphere: 1,
...     _id: 0
...   }
... )
...
[
  {
    Duration: 115,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Ann Rice' }
  },
  {
    Duration: 100,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Anna Breen' }
  },
  {
    Duration: 130,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Ann  Dowd' }
  },
  {
    Duration: 125,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Ann Rushe' }
  },
  {
    Duration: 125,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Amie Canny' }
  },
  {
    Duration: 100,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Mary Glynn' }
  },
  {
    Duration: 115,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Mary Henry' }
  },
  {
    Duration: 130,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Anne Maher' }
  },
  {
    Duration: 125,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Mary Hussy' }
  },
  {
    Duration: 105,
    Atmosphere: 'Excellent',
    Sponsor: { Name: 'Anne Ward' }
  }
]
Concert_Db>
```

In this query I introduced two new functions: **$regex** and **$options**: "i". $regex lets you search text using patterns. For example: "Sponsor.Name": { $regex: "Mary|Ann|Sam" }means: sponsor name includes Mary, Ann, or Sam anywhere in the string. $options: "i" keeps searching case-insensitively, so it doesn't matter if the letters are capital or small it will match both "Mary" and "mary".

In this query I was looking for concerts that had high-quality atmosphere, medium-length durations, and were sponsored by companies or people whose names included certain keywords (maybe common names). I just wanted to spot any trends related to certain sponsor names.

Query 19

Question:
Find concerts held in 2022 or 2023, where the actual crowd was less than expected, the sponsor offered "Posters", the stadium acoustics were rated "Fair", the performer's music type was "Jazz", the duration was greater than 120 minutes, and the cheapest ticket price was above €35.

```
db.Concert.countDocuments({
  $and: [
    { $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] } },
    { "Sponsor.Services": "Posters" },
    { "Stadium.Acoustics": "Fair" },
    { "Performer.Music_Type": "Jazz" },
    { Duration: { $gt: 120 } },
    { Cheapest_Ticket_Price: { $gt: 35 } },
    { Concert_Year: { $in: [2022, 2023] } }
  ]
})
db.Concert.find(
  {
    $and: [
      { $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] } },
      { "Sponsor.Services": "Posters" },
      { "Stadium.Acoustics": "Fair" },
      { "Performer.Music_Type": "Jazz" },
      { Duration: { $gt: 120 } },
      { Cheapest_Ticket_Price: { $gt: 35 } },
      { Concert_Year: { $in: [2022, 2023] } }
    ]
  },
  {
    Actual_Crowd: 1,
    Expected_Crowd: 1,
    "Sponsor.Services": 1,
    "Stadium.Acoustics": 1,
    "Performer.Music_Type": 1,
    Duration: 1,
```

```
    Cheapest_Ticket_Price: 1,
    Concert_Year: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   $and: [
...     { $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] } },
...     { "Sponsor.Services": "Posters" },
...     { "Stadium.Acoustics": "Fair" },
...     { "Performer.Music_Type": "Jazz" },
...     { Duration: { $gt: 120 } },
...     { Cheapest_Ticket_Price: { $gt: 35 } },
...     { Concert_Year: { $in: [2022, 2023] } }
...   ]
... })
3
Concert_Db> db.Concert.find(
...   {
...     $and: [
...       { $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] } },
...       { "Sponsor.Services": "Posters" },
...       { "Stadium.Acoustics": "Fair" },
...       { "Performer.Music_Type": "Jazz" },
...       { Duration: { $gt: 120 } },
...       { Cheapest_Ticket_Price: { $gt: 35 } },
...       { Concert_Year: { $in: [2022, 2023] } }
...     ]
...   },
...   {
...     Actual_Crowd: 1,
...     Expected_Crowd: 1,
...     "Sponsor.Services": 1,
...     "Stadium.Acoustics": 1,
...     "Performer.Music_Type": 1,
...     Duration: 1,
...     Cheapest_Ticket_Price: 1,
...     Concert_Year: 1,
...     _id: 0
...   }
... )
[
  {
    Concert_Year: 2023,
    Duration: 175,
    Expected_Crowd: 46400,
    Actual_Crowd: 38700,
    Cheapest_Ticket_Price: 60,
    Performer: { Music_Type: 'Jazz' },
    Sponsor: { Services: [ 'Ticket Sales', 'Radio Interviews', 'Posters' ] },
    Stadium: { Acoustics: 'Fair' }
  },
  {
    Concert_Year: 2022,
    Duration: 160,
    Expected_Crowd: 71900,
    Actual_Crowd: 65000,
    Cheapest_Ticket_Price: 50,
    Performer: { Music_Type: 'Jazz' },
    Sponsor: {
      Services: [
        'Ticket Sales',
        'Radio Interviews',
        'TV Interviews',
        'Posters'
      ]
    },
    Stadium: { Acoustics: 'Fair' }
  },
  {
    Concert_Year: 2022,
    Duration: 180,
    Expected_Crowd: 67200,
    Actual_Crowd: 57700,
    Cheapest_Ticket_Price: 60,
    Performer: { Music_Type: 'Jazz' },
    Sponsor: { Services: [ 'Radio Interviews', 'Posters' ] },
    Stadium: { Acoustics: 'Fair' }
  }
]
Concert_Db>
```

In this query I introduced a new function called **$expr**. $expr is used to compare two fields inside the same document, not just field vs. value. For example:$expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] }means: actual crowd was less than expected crowd this wouldn't be possible with just $lt i.e less than.

In this query I was looking for Jazz concerts that didn't attract as many people as expected. I added other filters like poster sponsorship, fair acoustics, longer show time, and a ticket price that wasn't cheap. It helps find well-planned shows that still underperformed.

Query 20

Question:
Find concerts where the expected crowd exceeded 20,000, but the actual crowd was lower, the merchandise included "Tee Shirts" but not "Umbrellas", the sponsor provided "Radio Interviews", the drink sales exceeded €50,000, the stadium type was "Hockey", and the atmosphere was not "Poor".

```
db.Concert.countDocuments({
  $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] },
  Expected_Crowd: { $gt: 20000 },
  Merchandise_Type: { $all: ["Tee Shirts"], $nin: ["Umbrellas"] },
  "Sponsor.Services": "Radio Interviews",
  Drink_Sales: { $gt: 50000 },
  "Stadium.Type": "Hockey",
  Atmosphere: { $ne: "Poor" }
})
db.Concert.find(
  {
    $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] },
    Expected_Crowd: { $gt: 20000 },
    Merchandise_Type: { $all: ["Tee Shirts"], $nin: ["Umbrellas"] },
    "Sponsor.Services": "Radio Interviews",
    Drink_Sales: { $gt: 50000 },
    "Stadium.Type": "Hockey",
    Atmosphere: { $ne: "Poor" }
  },
  {
    Expected_Crowd: 1,
    Actual_Crowd: 1,
    Merchandise_Type: 1,
    "Sponsor.Services": 1,
    Drink_Sales: 1,
    "Stadium.Type": 1,
    Atmosphere: 1,
    _id: 0
  }
)
```

```
Concert_Db> db.Concert.countDocuments({
...   $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] },
...   Expected_Crowd: { $gt: 20000 },
...   Merchandise_Type: { $all: ["Tee Shirts"], $nin: ["Umbrellas"] },
...   "Sponsor.Services": "Radio Interviews",
...   Drink_Sales: { $gt: 50000 },
...   "Stadium.Type": "Hockey",
...   Atmosphere: { $ne: "Poor" }
... })
4
Concert_Db> db.Concert.find(
...   {
...     $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] },
...     Expected_Crowd: { $gt: 20000 },
...     Merchandise_Type: { $all: ["Tee Shirts"], $nin: ["Umbrellas"] },
...     "Sponsor.Services": "Radio Interviews",
...     Drink_Sales: { $gt: 50000 },
...     "Stadium.Type": "Hockey",
...     Atmosphere: { $ne: "Poor" }
...   },
...   {
...     Expected_Crowd: 1,
...     Actual_Crowd: 1,
...     Merchandise_Type: 1,
...     "Sponsor.Services": 1,
...     Drink_Sales: 1,
...     "Stadium.Type": 1,
...     Atmosphere: 1,
...     _id: 0
...   }
... )
...
[
  {
    Expected_Crowd: 71700,
    Actual_Crowd: 62000,
    Drink_Sales: 62600,
    Atmosphere: 'Good',
    Sponsor: {
      Services: [ 'TV Interviews', 'Posters', 'News Papers', 'Radio Interviews' ]
    },
    Stadium: { Type: 'Hockey' },
    Merchandise_Type: [ 'Tee Shirts', 'Hoodies', 'Umbrellas ', 'Photos' ]
  },
  {
    Expected_Crowd: 47000,
    Actual_Crowd: 42900,
    Drink_Sales: 68800,
    Atmosphere: 'Good',
    Sponsor: { Services: [ 'Radio Interviews', 'News Papers' ] },
    Stadium: { Type: 'Hockey' },
    Merchandise_Type: [ 'Tee Shirts', 'Hoodies', 'Umbrellas ' ]
  },
  {
    Expected_Crowd: 37300,
    Actual_Crowd: 35600,
    Drink_Sales: 54100,
    Atmosphere: 'Fair',
    Sponsor: { Services: [ 'Radio Interviews' ] },
    Stadium: { Type: 'Hockey' },
    Merchandise_Type: [ 'Tee Shirts', 'Umbrellas ' ]
  },
  {
    Expected_Crowd: 80000,
    Actual_Crowd: 66500,
    Drink_Sales: 117300,
    Atmosphere: 'Excellent',
    Sponsor: { Services: [ 'Ticket Sales', 'Radio Interviews' ] },
    Stadium: { Type: 'Hockey' },
    Merchandise_Type: [ 'CD's', 'Tee Shirts', 'Umbrellas ' ]
  }
]
Concert_Db>
```

In this query I have again used **$expr**, which lets us compare two fields in the same document.
So this line: $expr: { $lt: ["$Actual_Crowd", "$Expected_Crowd"] } means: the actual number of people who came was less than expected.

I also used two other things:$all and $nin on the Merchandise_Type array.

- $all: ["Tee Shirts"] means it must include “Tee Shirts”.
- $nin: ["Umbrellas"] means it must not include “Umbrellas”.

This query was looking for concerts that had good expectations but lower turnout, sold t-shirts but no umbrellas, had media promotion, strong drink sales, and weren’t rated badly. I was trying to spot events that had good planning but maybe didn’t fully go as expected.

4 Mongo Shell – Aggregation Framework

In this section I have written aggregation queries tested using the MongoDB Shell. Again each query begins with a question for the query, followed by the actual command used to retrieve the data. Where appropriate, $count is included to verify the number of matches, and $project is used to display only relevant fields. Outputs for the query were manually verified using the shell.

Query 1

Question:

List all concerts after 2015 with a duration of more than 150 minutes, a dearest ticket price above 160, and an expected crowd greater than 70,000.
Only show the concert year, duration, dearest ticket price, expected crowd, performer name, and stadium name.

```
db.Concert.aggregate ([
  {
    $match : { Concert_Year: { $gt: 2015 }, Duration: { $gt: 150 }, Dearest_Ticket_Price: { $gt: 160 }, Expected_Crowd: { $gt: 70000 } }
  },
 {
    $count : "Count_Of_Documents"
  }
])
```

```
db.Concert.aggregate ([
  {
    $match : { Concert_Year: { $gt: 2015 }, Duration: { $gt: 150 }, Dearest_Ticket_Price: { $gt: 160 }, Expected_Crowd: { $gt: 70000 } }
  },
  {
    $project : { _id: 0, Concert_Year: 1, Duration: 1, Expected_Crowd: 1, Dearest_Ticket_Price: 1, "Performer.Name": 1, "Stadium.Name": 1 }
  }
])
```

```
Concert_Db> db.Concert.aggregate ([
...   {
...     $match : { Concert_Year: { $gt: 2015 }, Duration: { $gt: 150 }, Dearest_Ticket_Price: { $gt: 160 }, Expected_Crowd: { $gt: 70000 } }
...   },
...  {
...     $count : "Count_Of_Documents"
...   }
... ])
...
[ { Count_Of_Documents: 4 } ]
Concert_Db> db.Concert.aggregate ([
...   {
...     $match : { Concert_Year: { $gt: 2015 }, Duration: { $gt: 150 }, Dearest_Ticket_Price: { $gt: 160 }, Expected_Crowd: { $gt: 70000 } }
...   },
...   {
...     $project : { _id: 0, Concert_Year: 1, Duration: 1, Expected_Crowd: 1, Dearest_Ticket_Price: 1, "Performer.Name": 1, "Stadium.Name": 1 }
...   }
... ])
...
[
  {
    Concert_Year: 2019,
    Duration: 180,
    Expected_Crowd: 76800,
    Dearest_Ticket_Price: 215,
    Performer: { Name: 'Eoin O'Farrell' },
    Stadium: { Name: 'Olympia Theatre' }
  },
  {
    Concert_Year: 2022,
    Duration: 160,
    Expected_Crowd: 71900,
    Dearest_Ticket_Price: 220,
    Performer: { Name: 'Eoghan Brennan' },
    Stadium: { Name: 'Aviva Stadium' }
  },
  {
    Concert_Year: 2024,
    Duration: 155,
    Expected_Crowd: 76200,
    Dearest_Ticket_Price: 180,
    Performer: { Name: 'Donnchadh Mullan' },
    Stadium: { Name: 'National Concert Hall' }
  },
  {
    Concert_Year: 2017,
    Duration: 170,
    Expected_Crowd: 70200,
    Dearest_Ticket_Price: 180,
    Performer: { Name: 'Shannon Mullan' },
    Stadium: { Name: 'Pairc Ui Chaoimh' }
  }
]
Concert_Db>
```

In this aggregation, I used new stages for aggregation:

- **$match**: This stage filters the documents before doing anything else, it same like a regular find() filter. It only lets matching documents pass through. This stage finds the matching records and as pipeline it passes to another stage for further filtering
- **$project**: This stage controls which fields we want to show or don't want to show. It's like projection in a find(), but used inside the aggregation pipeline.

In this query I showed big concerts that happened after 2015. I was looking for long events with expensive tickets and a huge expected crowd. The goal was to spot large-scale shows that might have been headline events. I only picked a few important fields to show in the final result like who performed, where it happened, and how many people were expected.

Query 2

Question:

How many people were expected to attend concerts each year (after 2010), and what was the average concert duration for those years? Only include years that had at least 10 concerts and where the total expected crowd was more than 1,000,000. Do not sort the results.

```
db.Concert.aggregate ([

  { $match : { Concert_Year: { $gt: 2010 } } },

  { $group : { _id: "$Concert_Year", Total_Crowd: { $sum: "$Expected_Crowd" }, Avg_Duration: { $avg: "$Duration" }, Concert_Count: { $sum: 1 } } },

  { $match : { Concert_Count: { $gte: 10 }, Total_Crowd: { $gt: 2850000 } } },

  { $count : "Count_Of_Documents" }

])
```

```
db.Concert.aggregate ([

  { $match : { Concert_Year: { $gt: 2010 } } },

  { $group : { _id: "$Concert_Year", Total_Crowd: { $sum: "$Expected_Crowd" }, Avg_Duration: { $avg: "$Duration" }, Concert_Count: { $sum: 1 } } },

  { $match : { Concert_Count: { $gte: 10 }, Total_Crowd: { $gt: 2850000 } } },

  { $project : { _id: 0, Concert_Year: "$_id", Total_Crowd: 1, Avg_Duration: { $round: ["$Avg_Duration", 2] }, Concert_Count: 1 } }

])
```

```
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2010 } } },
...   { $group : { _id: "$Concert_Year", Total_Crowd: { $sum: "$Expected_Crowd" }, Avg_Duration: { $avg: "$Duration" }, Concert_Count: { $sum: 1 } } },
...   { $match : { Concert_Count: { $gte: 10 }, Total_Crowd: { $gt: 2850000 } } },
...   { $count : "Count_Of_Documents" }
... ])
...
[ { Count_Of_Documents: 5 } ]
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2010 } } },
...   { $group : { _id: "$Concert_Year", Total_Crowd: { $sum: "$Expected_Crowd" }, Avg_Duration: { $avg: "$Duration" }, Concert_Count: { $sum: 1 } } },
...   { $match : { Concert_Count: { $gte: 10 }, Total_Crowd: { $gt: 2850000 } } },
...   { $project : { _id: 0, Concert_Year: "$_id", Total_Crowd: 1, Avg_Duration: { $round: ["$Avg_Duration", 2] }, Concert_Count: 1 } }
... ])
...
[
  {
    Total_Crowd: 3387100,
    Concert_Count: 80,
    Concert_Year: 2020,
    Avg_Duration: 118.38
  },
  {
    Total_Crowd: 3261800,
    Concert_Count: 81,
    Concert_Year: 2013,
    Avg_Duration: 125.56
  },
  {
    Total_Crowd: 2989100,
    Concert_Count: 69,
    Concert_Year: 2015,
    Avg_Duration: 117.75
  },
  {
    Total_Crowd: 2940000,
    Concert_Count: 79,
    Concert_Year: 2016,
    Avg_Duration: 120.13
  },
  {
    Total_Crowd: 2853800,
    Concert_Count: 70,
    Concert_Year: 2011,
    Avg_Duration: 127.57
  }
]
Concert_Db>
```

- **$group**: This groups all documents by a certain field here it's by Concert_Year. It lets us do calculations for each group.
- **$sum**: Adds up numbers for the group (used here for total crowd and number of concerts).
- **$avg**: Calculates the average (used here for average concert duration).
- **$round**: Rounds off a number (I used it to clean up the average duration).
- **$count**: Just counts how many documents pass through at that point in the pipeline.

In this query I was looking for concerts that happened after 2010. For each year, it adds up the expected crowds and figures out the average show length. But I only kept the years where there were at least 10 concerts and the total expected crowd was huge (over 1 million). I didn't sort the result I just wanted to see which years passed the filter.

Query 3

Question:

For concerts held after 2010 that lasted more than 80 minutes and had a dearest ticket price above 90, calculate the average dearest ticket price and average concert duration for each performer. Also show how many concerts each performer has held. Only include performers who have done at least 2 concerts. Sort the performers in order of average ticket price (highest first).

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2010 }, Duration: { $gt: 80 }, Dearest_Ticket_Price: { $gt: 90 } }
},
  { $group : { _id: "$Performer.Name", Average_Ticket_Price: { $avg: "$Dearest_Ticket_Price" },
Average_Duration: { $avg: "$Duration" }, Num_Of_Concerts: { $sum: 1 } } },
  { $match : { Num_Of_Concerts: { $gte: 2 } } },
  { $count : "Count_Of_Documents" }
])
```

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2010 }, Duration: { $gt: 80 }, Dearest_Ticket_Price: { $gt: 90 } }
},
  { $group : { _id: "$Performer.Name", Avg_Ticket_Price: { $avg: "$Dearest_Ticket_Price" },
Avg_Duration: { $avg: "$Duration" }, Num_Of_Concerts: { $sum: 1 } } },
  { $match : { Num_Of_Concerts: { $gte: 2 } } },
  { $project : { _id: 0, Performer: "$_id", Avg_Ticket_Price: { $round: ["$Avg_Ticket_Price", 0] },
Avg_Duration: { $round: ["$Avg_Duration", 2] }, Concert_Count: 1 } },
  { $sort : { Avg_Ticket_Price: -1 } }
])
```

```
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2010 }, Duration: { $gt: 80 }, Dearest_Ticket_Price: { $gt: 90 } } },
...   { $group : { _id: "$Performer.Name", Average_Ticket_Price: { $avg: "$Dearest_Ticket_Price" }, Average_Duration: { $avg: "$Duration" }, Num_Of_Concerts: { $sum: 1 } } },
...   { $match : { Num_Of_Concerts: { $gte: 2 } } },
...   { $count : "Count_Of_Documents" }
... ])
...
[ { Count_Of_Documents: 9 } ]
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2010 }, Duration: { $gt: 80 }, Dearest_Ticket_Price: { $gt: 90 } } },
...   { $group : { _id: "$Performer.Name", Avg_Ticket_Price: { $avg: "$Dearest_Ticket_Price" }, Avg_Duration: { $avg: "$Duration" }, Num_Of_Concerts: { $sum: 1 } } },
...   { $match : { Num_Of_Concerts: { $gte: 2 } } },
...   { $project : { _id: 0, Performer: "$_id", Avg_Ticket_Price: { $round: ["$Avg_Ticket_Price", 0] }, Avg_Duration: { $round: ["$Avg_Duration", 2] }, Concert_Count: 1 } },
...   { $sort : { Avg_Ticket_Price: -1 } }
... ])
...
[
  {
    Performer: 'Cillin Doyle',
    Avg_Ticket_Price: 198,
    Avg_Duration: 150
  },
  {
    Performer: 'Torin O'Farrell',
    Avg_Ticket_Price: 175,
    Avg_Duration: 175
  },
  {
    Performer: 'Daire Moran',
    Avg_Ticket_Price: 172,
    Avg_Duration: 110
  },
  {
    Performer: 'Senan MacDonald',
    Avg_Ticket_Price: 142,
    Avg_Duration: 142.5
  },
  {
    Performer: 'Sorcha Clarke',
    Avg_Ticket_Price: 138,
    Avg_Duration: 122.5
  },
  {
    Performer: 'Blathnaid Power',
    Avg_Ticket_Price: 135,
    Avg_Duration: 145
  },
  {
    Performer: 'Cillian Daly',
    Avg_Ticket_Price: 130,
    Avg_Duration: 137.5
  },
  {
    Performer: 'Cathal Sweeney',
    Avg_Ticket_Price: 102,
    Avg_Duration: 132.5
  },
  {
    Performer: 'Padraig Collins',
    Avg_Ticket_Price: 102,
    Avg_Duration: 115
  }
]
Concert_Db>
```

This query uses all the functions from Query 2, plus one new one: **$sort**.

- **$sort** changes the order of the results. In ascending(1) or descending(-1)
  In this case:{ Avg_Ticket_Price: -1 }

means the results will show performers with the highest average ticket prices first (-1 is descending order).

In this query I was looking for more premium concerts longer shows with higher ticket prices. I grouped the data by performer to see which artists had the most expensive tickets, how long their concerts were, and how many concerts they performed. I only kept the performers who had done at least two such concerts so that the averages are more meaningful.

Query 4

Question:

What types of audiences were most commonly targeted concerts held after 2015? Count how many concerts targeted each audience type. Only include audience types that were targeted in at least 5 concerts. Sort the results by the number of concerts in descending order.

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2015 } } },
  { $unwind : "$Audience_Type" },
  { $group : { _id: "$Audience_Type", Concerts_Targeting_Audience: { $sum: 1 } } },
  { $match : { Concerts_Targeting_Audience: { $gte: 5 } } },
  { $count : "MatchingAudienceTypes" }
])
```

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2015 } } },
  { $unwind : "$Audience_Type" },
  { $group : { _id: "$Audience_Type", Concerts_Targeting_Audience: { $sum: 1 } } },
  { $match : { Concerts_Targeting_Audience: { $gte: 5 } } },
  { $project : { _id: 0, Audience_Type: "$_id", Concerts_Targeting_Audience: 1 } },
  { $sort : { Concerts_Targeting_Audience: -1 } }
])
```

```
Concert_Db> db.Concert.aggregate ([
...    { $match : { Concert_Year: { $gt: 2015 } } },
...    { $unwind : "$Audience_Type" },
...    { $group : { _id: "$Audience_Type", Concerts_Targeting_Audience: { $sum: 1 } } },
...    { $match : { Concerts_Targeting_Audience: { $gte: 5 } } },
...    { $count : "MatchingAudienceTypes" }
... ])
...
[ { MatchingAudienceTypes: 5 } ]
Concert_Db> db.Concert.aggregate ([
...    { $match : { Concert_Year: { $gt: 2015 } } },
...    { $unwind : "$Audience_Type" },
...    { $group : { _id: "$Audience_Type", Concerts_Targeting_Audience: { $sum: 1 } } },
...    { $match : { Concerts_Targeting_Audience: { $gte: 5 } } },
...    { $project : { _id: 0, Audience_Type: "$_id", Concerts_Targeting_Audience: 1 } },
...    { $sort : { Concerts_Targeting_Audience: -1 } }
... ])
...
[
  { Concerts_Targeting_Audience: 338, Audience_Type: 'Middle Aged' },
  { Concerts_Targeting_Audience: 328, Audience_Type: 'Retired ' },
  {
    Concerts_Targeting_Audience: 310,
    Audience_Type: 'Young Working Adults'
  },
  {
    Concerts_Targeting_Audience: 306,
    Audience_Type: 'Third Level Students'
  },
  { Concerts_Targeting_Audience: 302, Audience_Type: 'Teenagers' }
]
Concert_Db>
```

This query introduces a new function called **$unwind**.

- $unwind is used to break apart arrays. For example, if a concert has:

"Audience_Type": ["Teenagers", "Middle Aged"]

$unwind will split it into two separate records, one for each audience type.
This makes it easier to count how often each type appears.

In this query I was finding which audience groups were targeted the most after 2015 like teenagers, students, middle-aged people, etc. Since Audience_Type is an array, I used $unwind to break it up. Then I counted how many concerts each type appeared in. Finally, I filtered out any that appeared less than 5 times, and sorted the result to show the most popular Concert_Targeting Audience.

Query 5

Question:

For each concert year after 2015, calculate the average expected crowd, average actual crowd, and the difference between them. Only include years that had at least 3 concerts. Store the results in a new collection called Crowd_Accuracy_Stats. Confirm that the new collection exists by querying it afterwards.

db.Concert.find().pretty().limit(3)

db.Concert.aggregate ([

```
  { $match : { Concert_Year: { $gt: 2015 } } },

  { $group : { _id: "$Concert_Year", Avg_Expected: { $avg: "$Expected_Crowd" }, Avg_Actual: { $avg: "$Actual_Crowd" }, Num_Of_Concerts: { $sum: 1 } } },

  { $match : { Num_Of_Concerts: { $gte: 3 } } },

  { $addFields : { Avg_Expected: { $round: ["$Avg_Expected", 0] },
                   Avg_Actual: { $round: ["$Avg_Actual", 0] },
                   Crowd_Difference: { $round: [{ $subtract: ["$Avg_Expected", "$Avg_Actual"] }, 0]}
                   } },

  { $project : { _id: 0, Concert_Year: "$_id", Avg_Expected: 1, Avg_Actual: 1, Crowd_Difference: 1 } },

  {$sort: { Concert_Year: -1}},

  { $out : "Crowd_Accuracy_Stats" }

])
```

Showing the result in new collection that is crowd accuracy stats created

db.Crowd_Accuracy_Stats.find()

```
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2015 } } },
...   { $group : { _id: "$Concert_Year", Avg_Expected: { $avg: "$Expected_Crowd" }, Avg_Actual: { $avg: "$Actual_Crowd" }, Num_Of_Concerts: { $sum: 1 } } },
...   { $match : { Num_Of_Concerts: { $gte: 3 } } },
...   { $addFields : { Avg_Expected: { $round: ["$Avg_Expected", 0] },
...  Avg_Actual: { $round: ["$Avg_Actual", 0] },
... Crowd_Difference: { $round: [{ $subtract: ["$Avg_Expected", "$Avg_Actual"] }, 0]} } },
...   { $project : { _id: 0, Concert_Year: "$_id", Avg_Expected: 1, Avg_Actual: 1, Crowd_Difference: 1 } },
...   {$sort: { Concert_Year: -1}},
...   { $out : "Crowd_Accuracy_Stats" }
... ])
...

Concert_Db> db.Crowd_Accuracy_Stats.find()
[
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1d6'),
    Avg_Expected: 37652,
    Avg_Actual: 35792,
    Crowd_Difference: 1860,
    Concert_Year: 2024
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1d7'),
    Avg_Expected: 43506,
    Avg_Actual: 40877,
    Crowd_Difference: 2630,
    Concert_Year: 2023
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1d8'),
    Avg_Expected: 38025,
    Avg_Actual: 36865,
    Crowd_Difference: 1160,
    Concert_Year: 2022
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1d9'),
    Avg_Expected: 38019,
    Avg_Actual: 35849,
    Crowd_Difference: 2169,
    Concert_Year: 2021
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1da'),
    Avg_Expected: 42339,
    Avg_Actual: 39266,
    Crowd_Difference: 3072,
    Concert_Year: 2020
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1db'),
    Avg_Expected: 37509,
    Avg_Actual: 36246,
    Crowd_Difference: 1263,
    Concert_Year: 2019
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1dc'),
    Avg_Expected: 41281,
    Avg_Actual: 40166,
    Crowd_Difference: 1115,
    Concert_Year: 2018
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1dd'),
    Avg_Expected: 34354,
    Avg_Actual: 32380,
    Crowd_Difference: 1974,
    Concert_Year: 2017
  },
  {
    _id: ObjectId('67fd0145d1f090d8efe5d1de'),
    Avg_Expected: 37215,
    Avg_Actual: 36234,
    Crowd_Difference: 981,
    Concert_Year: 2016
  }
]
Concert_Db> |
```

This query uses a few new functions:

- **$addFields**: Adds new fields to the result, even after grouping.
- **$subtract**: Subtracts one number from another.
- **$out**: Writes the output into a new collection in the database.

All Together, these query allow us to do some extra calculations and then save the results for future use.

In this query I compared how accurate the crowd estimates were for each year. It checks if concerts were attracting as many people as expected. I grouped the data by year, then calculated both averages (expected and actual), and the difference. I have kept years only with at least 3 concerts to make the averages reliable. Finally, I saved the results in a new collection called Crowd_Accuracy_Stats and showed the result using Crowd_Accuracy_Stats collection.

Query 6

Question:

For concerts held after 2010, calculate the average expected crowd for each concert year. Based on the average, assign a crowd category as follows: ‘Huge’ if the average is greater than 40,000, ‘Medium’ if it is between 35,000 and 40,000, and ‘Small’. Sort the results by concert year in ascending order. if below 35,000

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2010 } } },
  { $group : { _id: "$Concert_Year", Avg_Crowd: { $avg: "$Expected_Crowd" } } },
  { $addFields : {
     Crowd_Category: {
      $cond: {
       if: { $gt: ["$Avg_Crowd", 40000] },
       then: "Huge",
       else: {
        $cond: {
         if: { $gte: ["$Avg_Crowd", 35000] },
         then: "Medium",
         else: "Small"
        }
       }
      }
  }}},
  { $project : { _id: 0, Concert_Year: "$_id", Avg_Crowd: { $round: ["$Avg_Crowd", 0] },
Crowd_Category: 1 } },
{$sort: { Concert_Year : 1}}
])
```

```
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2010 } } },
...   { $group : { _id: "$Concert_Year", Avg_Crowd: { $avg: "$Expected_Crowd" } } },
...   { $addFields : {
...       Crowd_Category: {
...         $cond: {
...           if: { $gt: ["$Avg_Crowd", 40000] },
...           then: "Huge",
...           else: {
...             $cond: {
...               if: { $gte: ["$Avg_Crowd", 35000] },
...               then: "Medium",
...               else: "Small"
...             }
...           }
...         }
...   }}},
...   { $project : { _id: 0, Concert_Year: "$_id", Avg_Crowd: { $round: ["$Avg_Crowd", 0] }, Crowd_Category: 1 } },
... {$sort: { Concert_Year : 1}}
... ])
...
[
  { Crowd_Category: 'Huge', Concert_Year: 2011, Avg_Crowd: 40769 },
  { Crowd_Category: 'Medium', Concert_Year: 2012, Avg_Crowd: 37372 },
  { Crowd_Category: 'Huge', Concert_Year: 2013, Avg_Crowd: 40269 },
  { Crowd_Category: 'Huge', Concert_Year: 2014, Avg_Crowd: 42507 },
  { Crowd_Category: 'Huge', Concert_Year: 2015, Avg_Crowd: 43320 },
  { Crowd_Category: 'Medium', Concert_Year: 2016, Avg_Crowd: 37215 },
  { Crowd_Category: 'Small', Concert_Year: 2017, Avg_Crowd: 34354 },
  { Crowd_Category: 'Huge', Concert_Year: 2018, Avg_Crowd: 41281 },
  { Crowd_Category: 'Medium', Concert_Year: 2019, Avg_Crowd: 37509 },
  { Crowd_Category: 'Huge', Concert_Year: 2020, Avg_Crowd: 42339 },
  { Crowd_Category: 'Medium', Concert_Year: 2021, Avg_Crowd: 38019 },
  { Crowd_Category: 'Medium', Concert_Year: 2022, Avg_Crowd: 38025 },
  { Crowd_Category: 'Huge', Concert_Year: 2023, Avg_Crowd: 43506 },
  { Crowd_Category: 'Medium', Concert_Year: 2024, Avg_Crowd: 37652 }
]
Concert_Db>
```

This query introduces a new function called **$cond**, which is like an if-else condition.

It works like this:

$cond: {

if: <some condition>,

then: <value if true>,

else: <value if false>

}

You can also nest $cond inside another $cond to check multiple ranges (like we did here to decide between "Huge", "Medium", or "Small").

In this query I am grouping concerts by year and calculating the average expected crowd size. Then it gives each year a label either "Huge", "Medium", or "Small" depending on how big that average was. This makes it easy to understand crowd sizes over time without looking at raw numericals.

Query 7

Question:

For concerts held after 2010, group the concerts into ticket price categories using the dearest ticket price. Use the following buckets: ‘Budget’ (up to 50), ‘Affordable’ (51–100), ‘Standard’ (101–200), and ‘Premium’ (over 200). Show how many concerts fall into each ticket price range. Only show results where at least one concert is in the category.

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2010 } } },
  { $bucket : {
     groupBy: "$Dearest_Ticket_Price",
     boundaries: [ 0, 50, 100, 200, 1000 ],
     default: "Other",
     output: {
       Num_Of_Concerts: { $sum: 1 }
     }
  }}
])
```

```
Concert_Db> db.Concert.aggregate ([
...    { $match : { Concert_Year: { $gt: 2010 } } },
...    { $bucket : {
...        groupBy: "$Dearest_Ticket_Price",
...        boundaries: [ 0, 50, 100, 200, 1000 ],
...        default: "Other",
...        output: {
...          Num_Of_Concerts: { $sum: 1 }
...        }
...    }}
... ])
[
  { _id: 0, Num_Of_Concerts: 140 },
  { _id: 50, Num_Of_Concerts: 285 },
  { _id: 100, Num_Of_Concerts: 403 },
  { _id: 200, Num_Of_Concerts: 121 }
]
Concert_Db>
```

This query introduces a new function called **$bucket**.

- $bucket is used to group documents in form of ranges of a number field.
- You give it a list of boundaries, and MongoDB puts values into the correct range (or "bucket").
- Each bucket will count how many documents fell into that range.

In this case, I used it to group concerts by their dearest ticket price in range [0, 50, 100, 200, 1000]

In this query I am grouping all concerts into different price categories. It helps to show how many concerts had cheap tickets vs mid-range or high-end ticket prices. This is useful to see pricing trends and which type of ticketing was more common.

Query 8

Question:

For conclerts held after 2015, show total food sales grouped by concert year, and total drink sales grouped by performer (only include performers with drink sales over 40,000). Use a single aggregation query to produce both results in parallel.

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gt: 2015 } } },
  { $facet : {
     Food_Sales_By_Year: [
       { $group : { _id: "$Concert_Year", Total_Food_Sales: { $sum: "$Food_Sales" } } },
       { $sort : { Total_Food_Sales: -1 } }
     ],
     Drink_Sales_By_Performer: [
       { $group : { _id: "$Performer.Name", Total_Drink_Sales: { $sum: "$Drink_Sales" } } },
       { $match: { Total_Drink_Sales: { $gt: 120000 } } },
       { $sort : { Total_Drink_Sales: -1 } }
     ]
  }}
])
```

```
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gt: 2015 } } },
...   { $facet : {
...       Food_Sales_By_Year: [
...         { $group : { _id: "$Concert_Year", Total_Food_Sales: { $sum: "$Food_Sales" } } },
...         { $sort : { Total_Food_Sales: -1 } }
...       ],
...       Drink_Sales_By_Performer: [
...         { $group : { _id: "$Performer.Name", Total_Drink_Sales: { $sum: "$Drink_Sales" } } },
...         { $match: { Total_Drink_Sales: { $gt: 120000 } } },
...         { $sort : { Total_Drink_Sales: -1 } }
...       ]
...   }}
... ])
...
[
  {
    Food_Sales_By_Year: [
      { _id: 2020, Total_Food_Sales: 4296500 },
      { _id: 2016, Total_Food_Sales: 4116400 },
      { _id: 2017, Total_Food_Sales: 3632600 },
      { _id: 2019, Total_Food_Sales: 3554800 },
      { _id: 2021, Total_Food_Sales: 3392100 },
      { _id: 2023, Total_Food_Sales: 3330500 },
      { _id: 2018, Total_Food_Sales: 3204800 },
      { _id: 2022, Total_Food_Sales: 3081600 },
      { _id: 2024, Total_Food_Sales: 2787800 }
    ],
    Drink_Sales_By_Performer: [
      { _id: 'Cathal Sweeney', Total_Drink_Sales: 237600 },
      { _id: 'Comhghall O'Leary', Total_Drink_Sales: 182100 },
      { _id: 'Daire Moran', Total_Drink_Sales: 165100 },
      { _id: 'Grainne Johnston', Total_Drink_Sales: 154300 },
      { _id: 'Uilliam O'Rourke', Total_Drink_Sales: 151900 },
      { _id: 'Ruairi Nolan', Total_Drink_Sales: 138000 },
      { _id: 'Fiona Kennedy', Total_Drink_Sales: 137700 },
      { _id: 'Darragh Burns', Total_Drink_Sales: 131600 },
      { _id: 'Shannon O'Neill', Total_Drink_Sales: 130600 },
      { _id: 'Eachann O'Shea', Total_Drink_Sales: 129300 },
      { _id: 'Aisling Campbell', Total_Drink_Sales: 128700 }
    ]
  }
]
Concert_Db>
```

This query introduces a new and very useful stage called **$facet**.

- $facet lets you run multiple aggregations in parallel.
- Each section in facet works like its own pipeline, and the results are grouped together at the end.
- It's perfect when you want to get different views of the same data in one go.

This query is like running two separate reports at the same time.

- One part adds up food sales by year this helps understand overall food revenue trends.
- The other part shows which performers had high drink sales so we can see which artists helped boost sales at concerts.

I used $group, $sum, $match, and $sort inside each part, and then combined everything using $facet.

Query 9

Question:

For concerts held during 2020 and 2021 with an expected crowd greater than 60,000, find concerts where the stadium name contains the word ‘Park’. Show the concert year, stadium name, performer name, and expected crowd. Also assign a capacity label using a switch expression: label as ‘Large Stadium’ if the expected crowd is greater than or equal to 70,000, otherwise label as ‘Medium Stadium’. Sort the results by expected crowd in descending order.

```
db.Concert.aggregate ([
  { $match : { Concert_Year: { $gte: 2020, $lte: 2021 }, Expected_Crowd: { $gt: 60000 }, "Stadium.Name": { $regex: "Park", $options: "i" } } },
  { $addFields : {
     Capacity_Label: {
       $switch: {
         branches: [
           { case: { $gte: ["$Expected_Crowd", 70000] }, then: "Large Stadium" }
         ],
         default: "Medium Stadium"
       }
     }
  }},
  { $project : {
     _id: 0,
     Concert_Year: 1,
     "Stadium.Name": 1,
     "Performer.Name": 1,
     Expected_Crowd: 1,
     Capacity_Label: 1
  }},
  { $sort : { Expected_Crowd: -1 } }
])
```

```
Concert_Db> db.Concert.aggregate ([
...   { $match : { Concert_Year: { $gte: 2020, $lte: 2021 }, Expected_Crowd: { $gt: 60000 }, "Stadium.Name": { $regex: "Park", $options: "i" } } },
...   { $addFields : {
...       Capacity_Label: {
...         $switch: {
...           branches: [
...             { case: { $gte: ["$Expected_Crowd", 70000] }, then: "Large Stadium" }
...           ],
...           default: "Medium Stadium"
...         }
...       }
...   }},
...   { $project : {
...       _id: 0,
...       Concert_Year: 1,
...       "Stadium.Name": 1,
...       "Performer.Name": 1,
...       Expected_Crowd: 1,
...       Capacity_Label: 1
...   }},
...   { $sort : { Expected_Crowd: -1 } }
... ])
...
[
  {
    Concert_Year: 2021,
    Expected_Crowd: 76900,
    Performer: { Name: 'Eachann Flanagan' },
    Stadium: { Name: 'Nowlan Park' },
    Capacity_Label: 'Large Stadium'
  },
  {
    Concert_Year: 2020,
    Expected_Crowd: 76900,
    Performer: { Name: 'Fiadh Murphy' },
    Stadium: { Name: 'Nowlan Park' },
    Capacity_Label: 'Large Stadium'
  },
  {
    Concert_Year: 2021,
    Expected_Crowd: 69800,
    Performer: { Name: 'Donnchadh McDonnell' },
    Stadium: { Name: 'Nowlan Park' },
    Capacity_Label: 'Medium Stadium'
  },
  {
    Concert_Year: 2020,
    Expected_Crowd: 68200,
    Performer: { Name: 'Odhran McCarthy' },
    Stadium: { Name: 'OMoore Park' },
    Capacity_Label: 'Medium Stadium'
  },
  {
    Concert_Year: 2020,
    Expected_Crowd: 66900,
    Performer: { Name: 'Donnchadh MacDonald' },
    Stadium: { Name: 'Nowlan Park' },
    Capacity_Label: 'Medium Stadium'
  },
  {
    Concert_Year: 2020,
    Expected_Crowd: 66300,
    Performer: { Name: 'Aine Johnston' },
    Stadium: { Name: 'OMoore Park' },
    Capacity_Label: 'Medium Stadium'
  },
  {
    Concert_Year: 2020,
    Expected_Crowd: 66000,
    Performer: { Name: 'Torin O'Leary' },
    Stadium: { Name: 'Thomond Park' },
    Capacity_Label: 'Medium Stadium'
  }
]
Concert_Db>
```

This query introduces a new function called **$switch**, which is very similar to creating conditional labels using **$cond** its like a cleaner version of multiple $cond statements.

- $switch checks different cases, one after another, and picks the first one that is true.
- If none of the cases match, it uses a default value.

In this query, I used $switch to label each concert based on how large the expected crowd was.

In this query I am focusing on big concerts that happened in 2020 or 2021 in stadiums which has "Park" in their name. I used a regex to search for "Park" without worrying about uppercase or lowercase letters using $options. Then, I labeled each concert by crowd size using $switch, and sorted everything by the biggest crowds first.

5 MongoDB Compass – Overview and Simple Queries

5.1 Overview

MongoDB Compass is a visual tool that helps you work with MongoDB using a clean and intercative interface. Instead of typing everything in the terminal, Compass lets you see collections, documents, run queries, and view results all in one window.

In my project, I used Compass to:

- View and explore the documents in my Concert collection.
- Run simple and advanced queries using the filter panel.
- Try out projections and limits.
- Use different views like Stage View, Text View, and Table View.

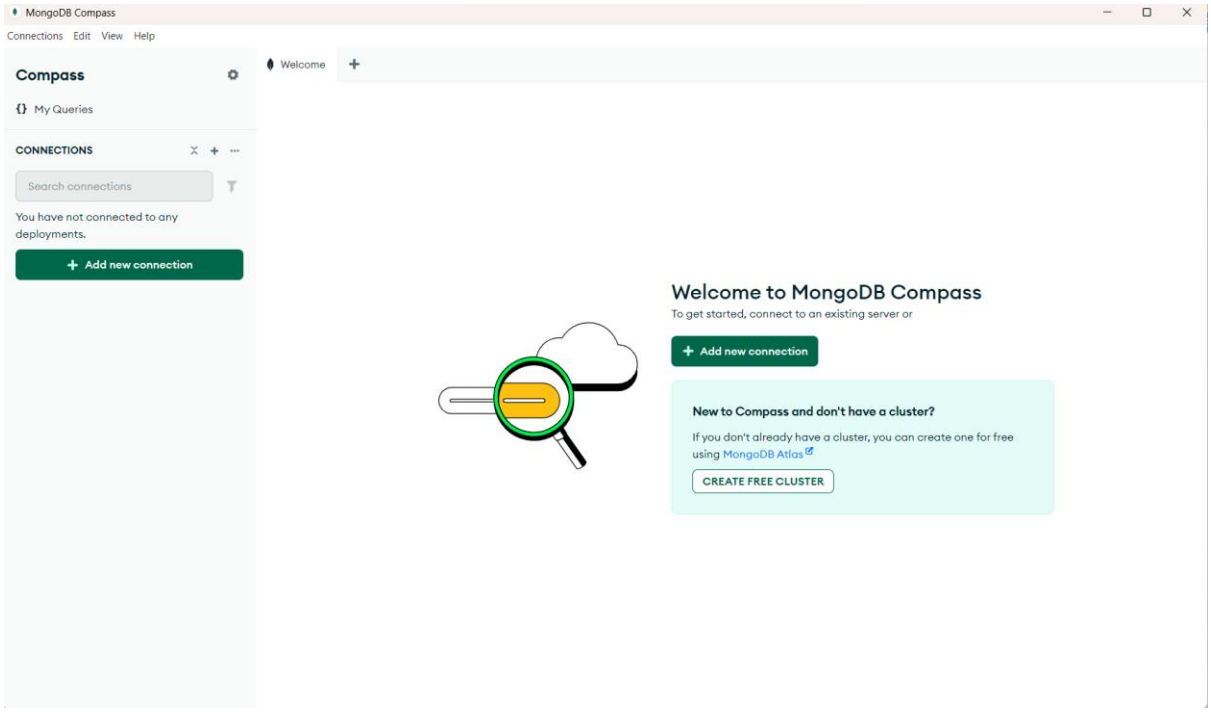

Figure 1: you can see the MongoDB Compass application is installed and is ready to connect to the local MongoDB server. This is the first step I did before you can start interacting with your project interface data. MongoDB Compass is a powerful tool for visualizing, querying, and analyzing MongoDB data with an interactive.

Figure Shows that MongoDB Compass is installed and that the application is ready to connect. This step is crucial because it ensures that you have the right environment to work with your database.

5.2 Connecting to MongoDB

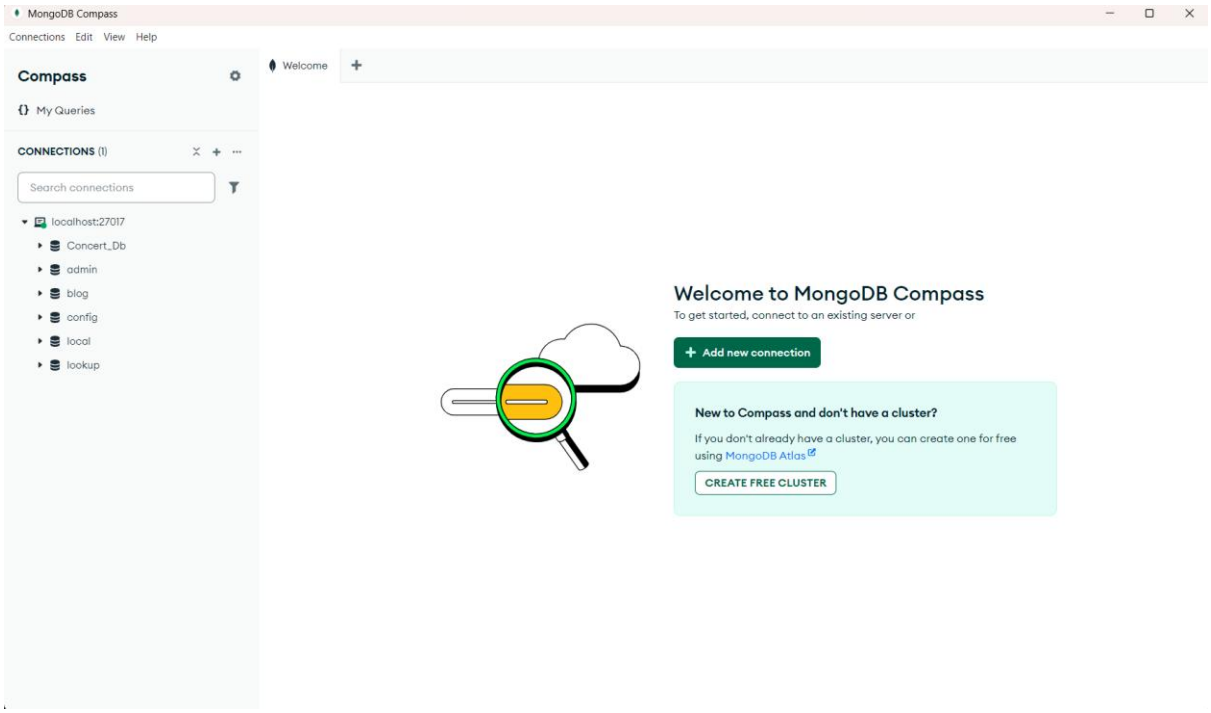

Figure 2: Shows that after clicking the connection button compass automatically connected to the localhost server without the need of connection string. This is because the compass automatically detects the local mongodb server.

The image above confirms that the connection to the local mogodb server has established. Thi step is required to further access your database and collections in the database.

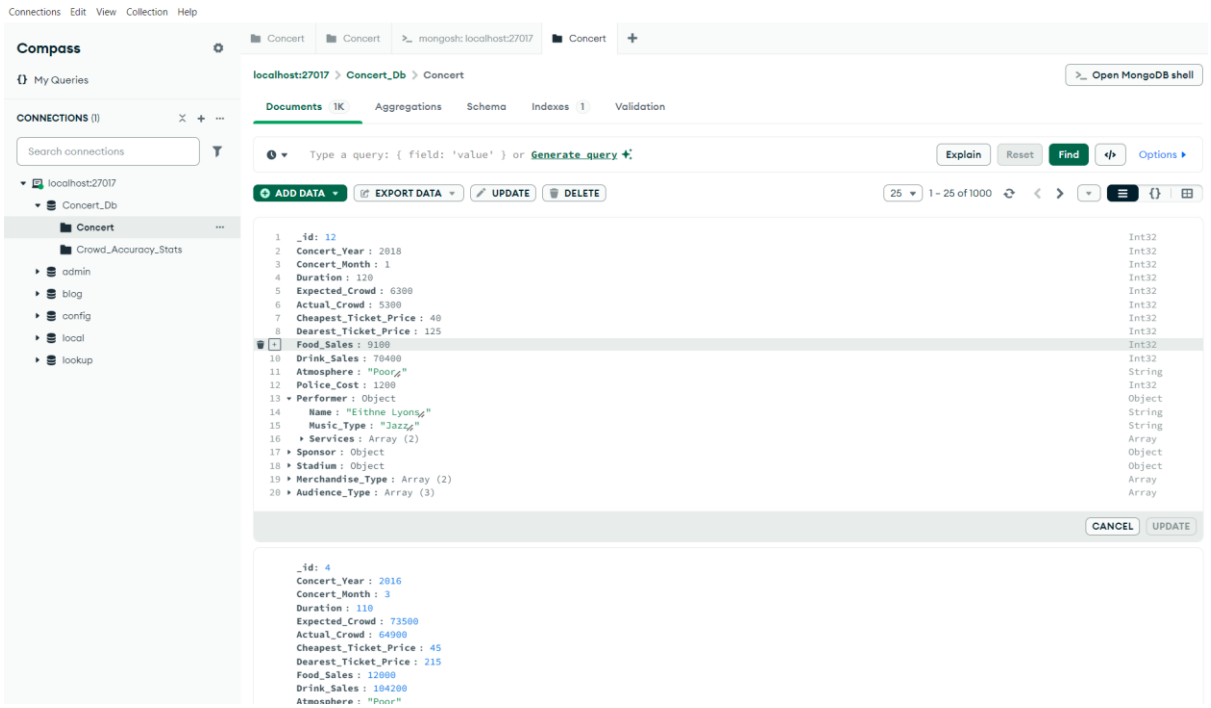

Figure 3: Shows Concert_Db database in the sidebar after the connection when clicking on the database name you can see whole collection is been accessed and all the documents within the collection are displayed.

This step is useful as it allows you to visually inspect the entire data in the collection and understand the data before you run any queries or aggregations. Also you can edit or delete any fields from the document or whole document if required as you can see in figure3. At a time you can see number of records for example in figure I have set to 25 records from 1000 records in the collection so it will show only initial 25 records you can change it according to your requirents.

5.3 Simple Queries in Compass

Query 1:

Find concerts held in 2022, with an audience size over 20,000, where the atmosphere was “Excellent”, and the music type was “Rock”.

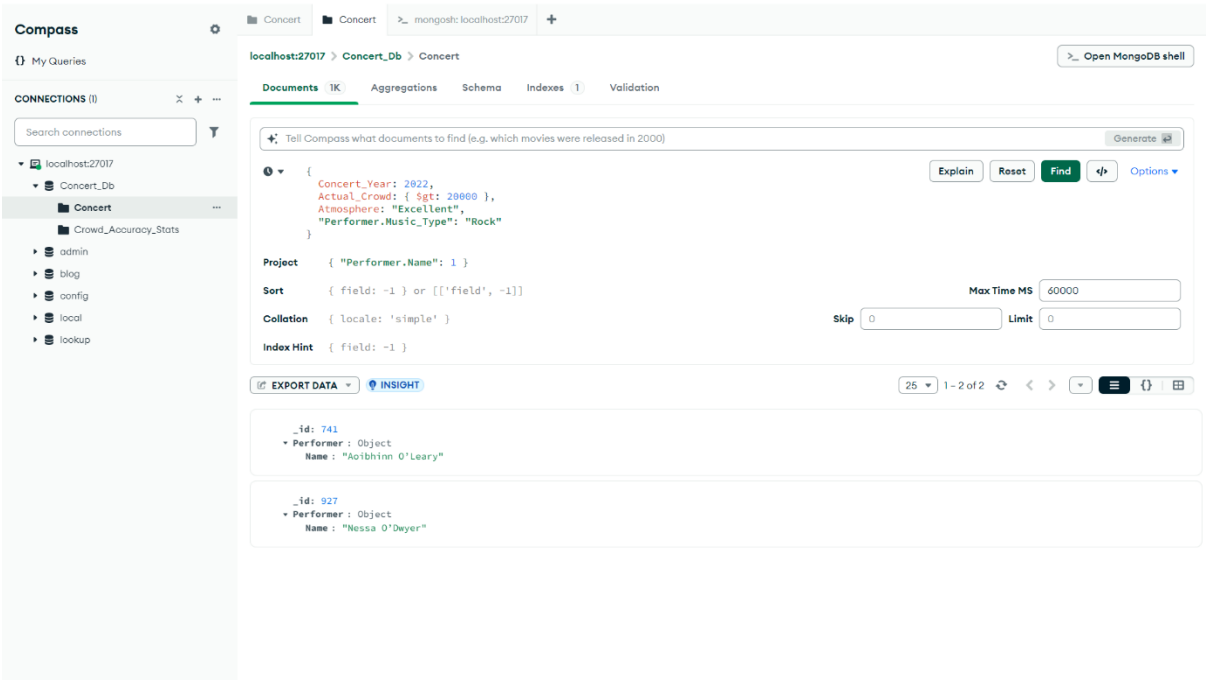

Figure 4: Shows the first query execution. Projection is used to display only the Performer Name, which means that only the names of the performers meeting the criteria will be shown in the results.

Result: The query returned the performers names for the concerts matching the filter. In this case, performers like Aoibhinn O’Leary and Nessa O’Dwyer have been shown in the results.

Query 2:

Find concerts held in Dublin in 2023, where police cost is under €15,000, and sponsor amount is over €120,000.

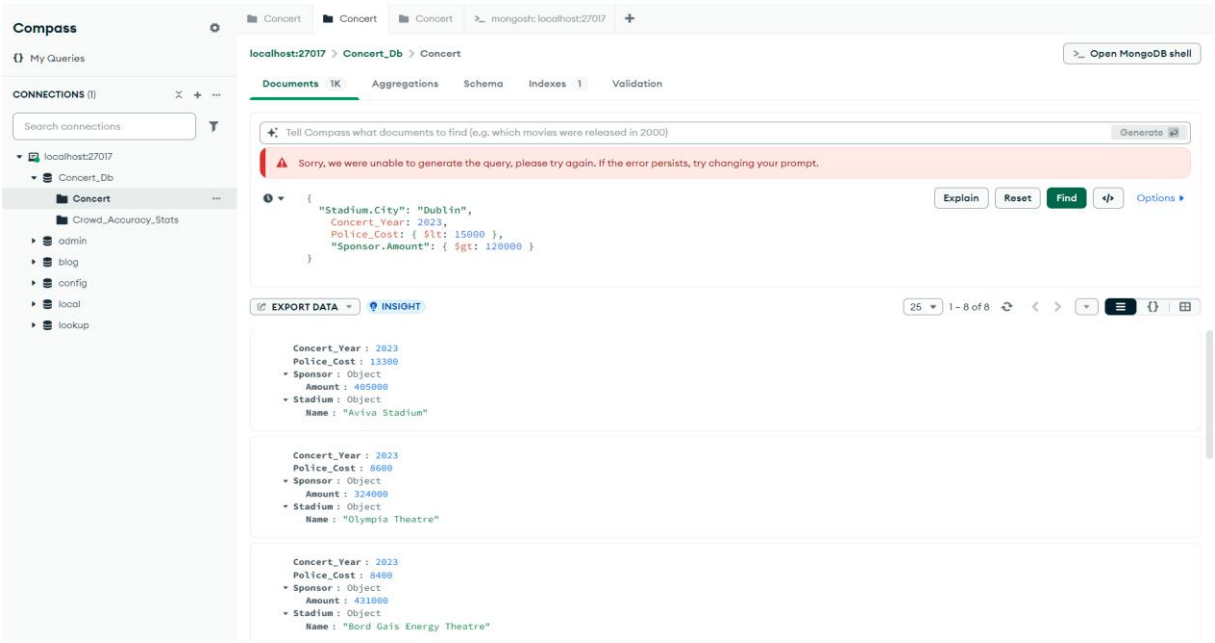

Figure 5: This query is showing concerts held in 2023 year in Dublin with some other criteria's. The main struggle I faced here was taking snip in this query still in the figure 4 all the documents from the output is not visible that is the issue in compass you cant toggle the output screen so whole output in the snip will not be visible in the snip

Result: The query returned the limited fields like concert_year, police_cost based on the projection that I provided.

5.4 Aggregation Queries in Compass

After completing the simple find queries to inspect the data, I moved on to explore some aggregation queries using MongoDB Compass. The Aggregation Tab provides a more advanced way to process the data, especially when you need to apply filters, grouping, and transformations across multiple stages.

Query 1:

List all concerts after 2015 with a duration of more than 150 minutes, a dearest ticket price above 160, and an expected crowd greater than 70,000.
Only show the concert year, duration, dearest ticket price, expected crowd, performer name, and stadium name.

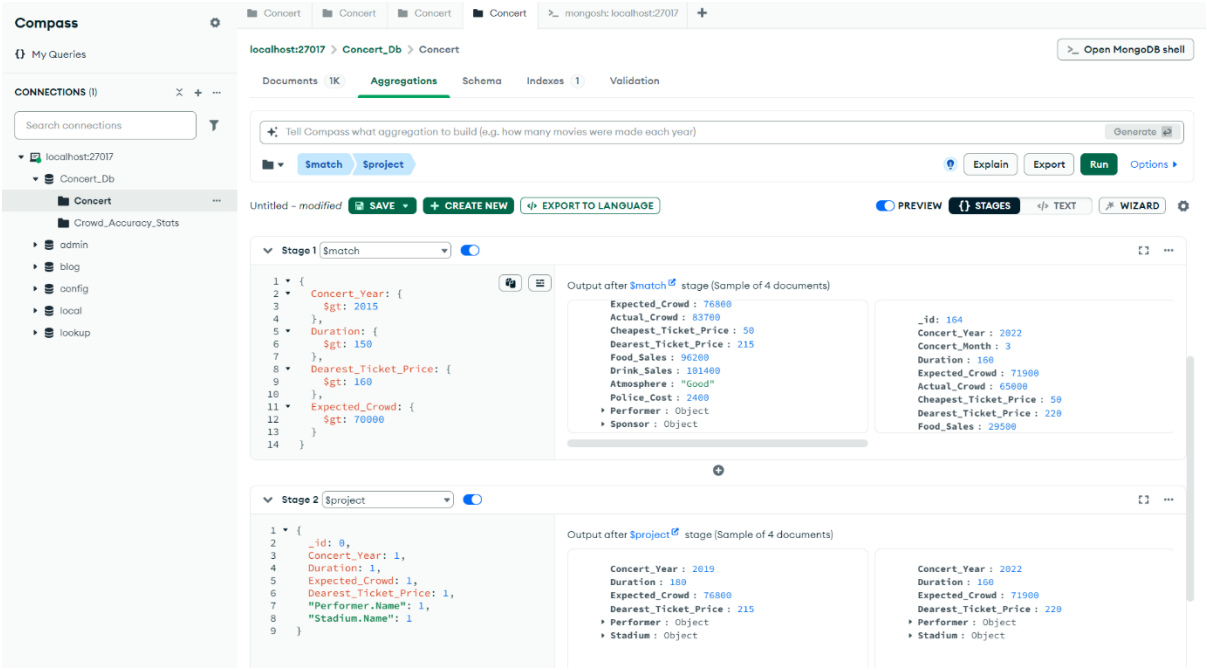

Figure 6: It shows the detail view of how aggreation pipeline is performed in compass here $match is matching concerts after 2015, duration of more than 150 min, dearest ticket price above 160 and expected crowd greated than 7000. In mongodb compass the stages are given different sections which make the interface really good as you can see $project has different section and it is selecting couple of fields like concert_year, duration etc which is visible in the output. Also you can disable or enable the stages using the toggle button given besides the stage name. In this figure I have used the stages view to showcase the output which makes the output unreadable in next query I will use text view and compare with the stage view.

Query 2:

What types of audiences were most commonly targeted after by concerts held 2015? Count how many concerts targeted each audience type. Only include audience types that were targeted in at least 5 concerts. Sort the results by the number of concerts in descending order.

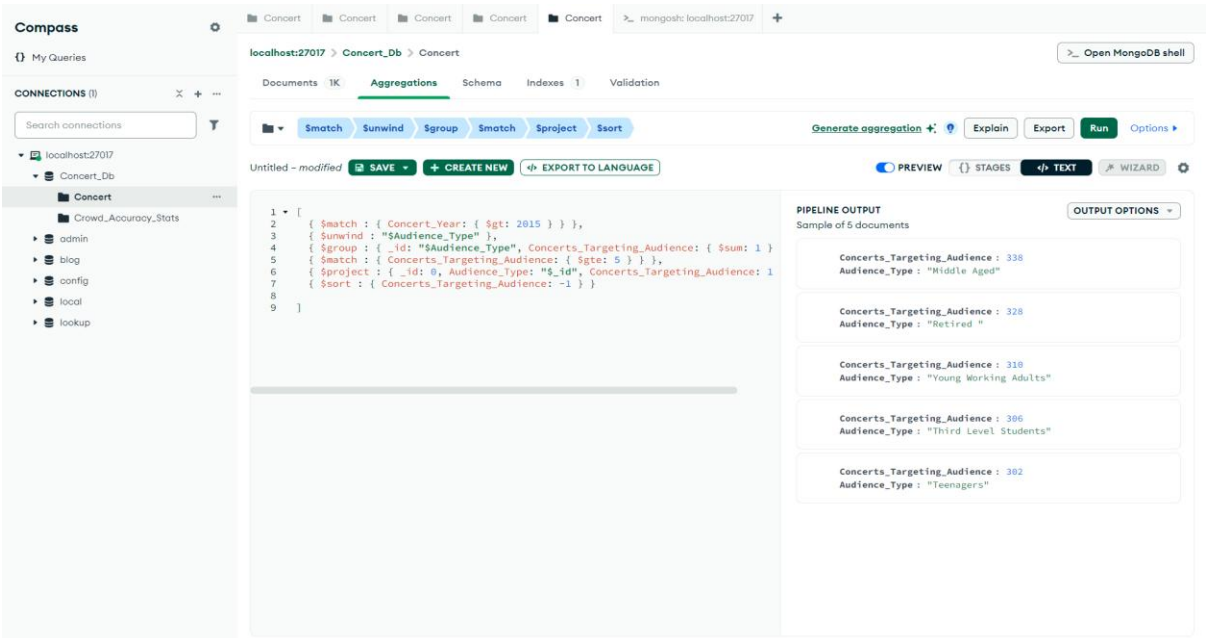

Figure 7: MongoDB Compass displays the Aggregation Pipeline using Text View. This view shows all stages of the aggregation query in one place, increasing its readabilty. The pipeline includes several stages, such as:

1. **$match**: Filters concerts after 2015.
2. **$unwind**: Expands the Audience_Type array so each audience type is handled separately.
3. **$group**: Groups concerts by Audience_Type and counts how many concerts target each audience type, but only includes audience types that have at least 5 concerts.
4. **$sort**: Sorts the results by the number of concerts targeting each audience type.

This view provides the advantage of seeing the entire query in one window. It's helpful for understanding the full structure of the aggregation query, especially when you have multiple stages. However, it lacks the ability to toggle individual stages like the Stages View does.

Comparison between stage and text view:

In Stages View, each stage is visually separated, and you can toggle the visibility of individual stages to inspect their impact. However, Text View is more compact and shows everything at once, which can be beneficial for getting an overview of the query, but it lacks the flexibility of toggling stages individually.

5.5 Other features explored in compass

In MongoDB Compass, apart from the Text View and Stages View, there is also two more that I came across for viewing and querying your data: Code View and Table View.

Code View:

- This is the most compact and developer-friendly format for interacting with your data. It’s very similar to the Mongo Shell, where queries are written in raw code.
- It’s useful for those who prefer writing MongoDB commands manually or need to see the query and output in a more code-centric format.

Table View:

- As shown in Figure 9, Table View gives you a clean, table-like layout, where each row corresponds to a document in your collection.
- This view is perfect for quickly viewing and analyzing your data because its clearly visible in table format, especially when working with aggregations. It allows you to visually scan through the results, making it ideal for analyzing performance data, audience numbers, etc.

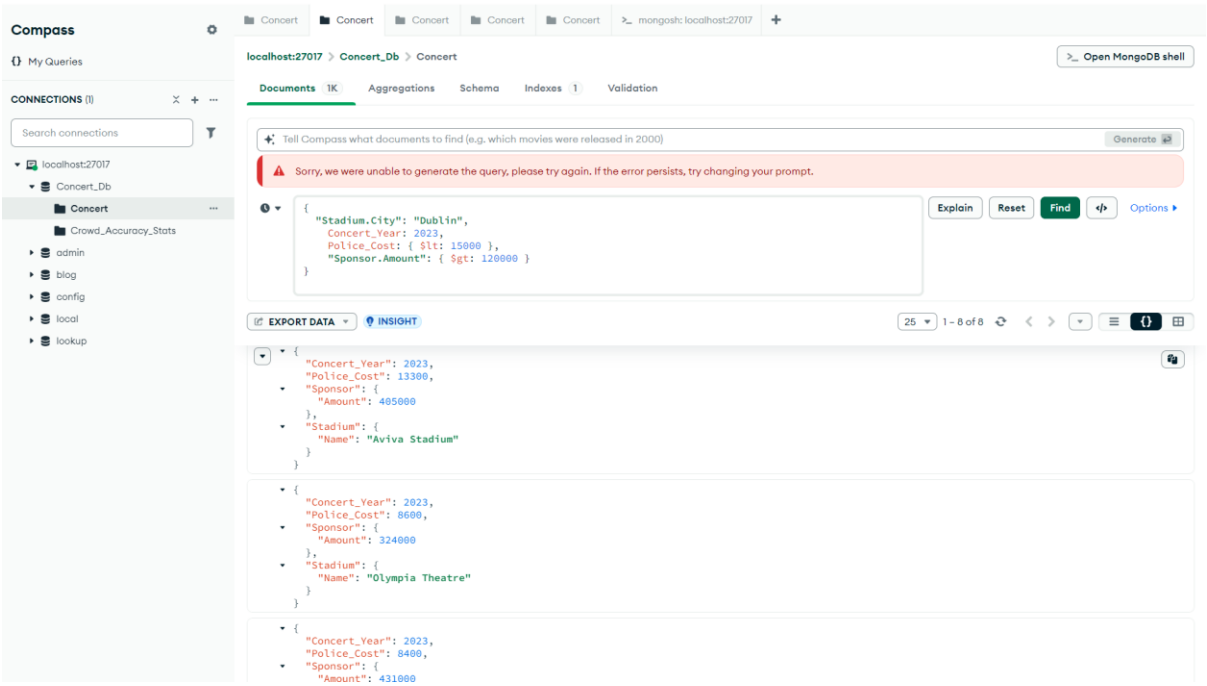

Figure 8: shows the Code View output in MongoDB Compass. This view displays the query results in a raw document model format, similar to how you would view the output in a MongoDB shell.

It's ideal for anyone whoever needs to interact with the data programmatically or export it directly for use in an application.

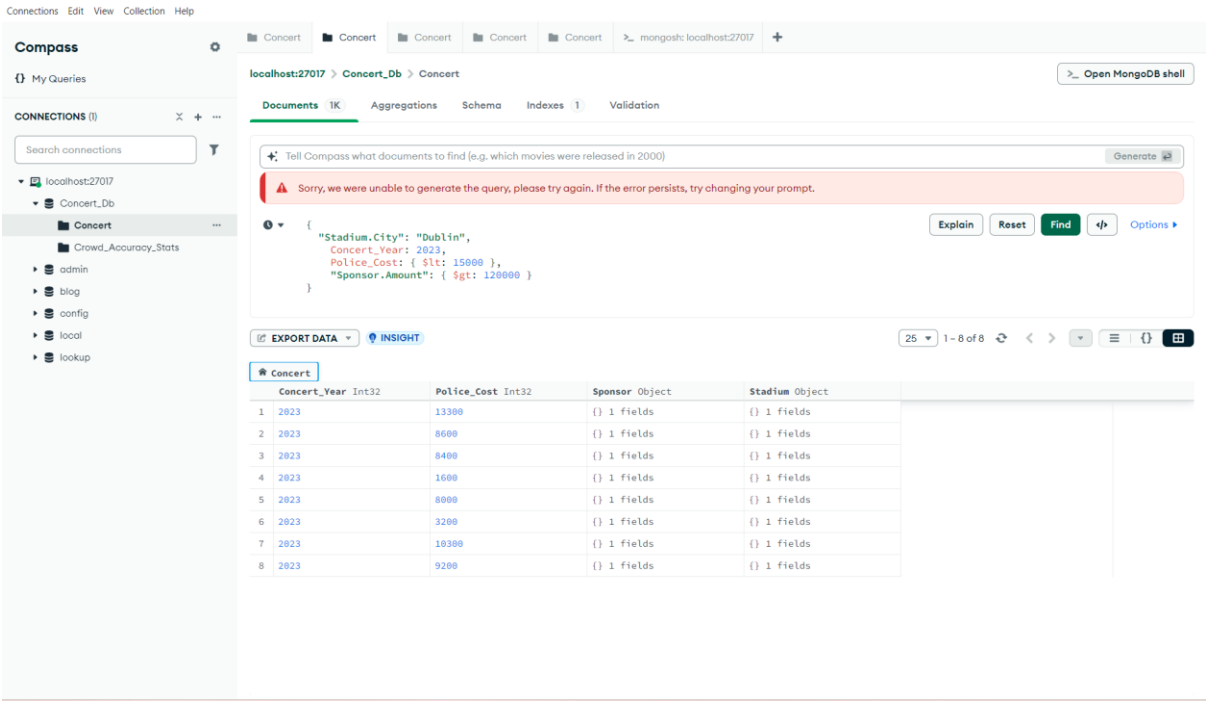

Figure 9: The results are presented in Table View, which transforms the documents into a table-like format. The rows represent individual documents, and columns represent fields such as Concert Year, Performer Name, and other relevant information.

This Table View allows you to visually scan the output, making it much easier to identify trends and patterns in the data. For example, you can quickly compare concert years, crowd sizes, and performers, which makes it a very useful format for business analysis and reporting.

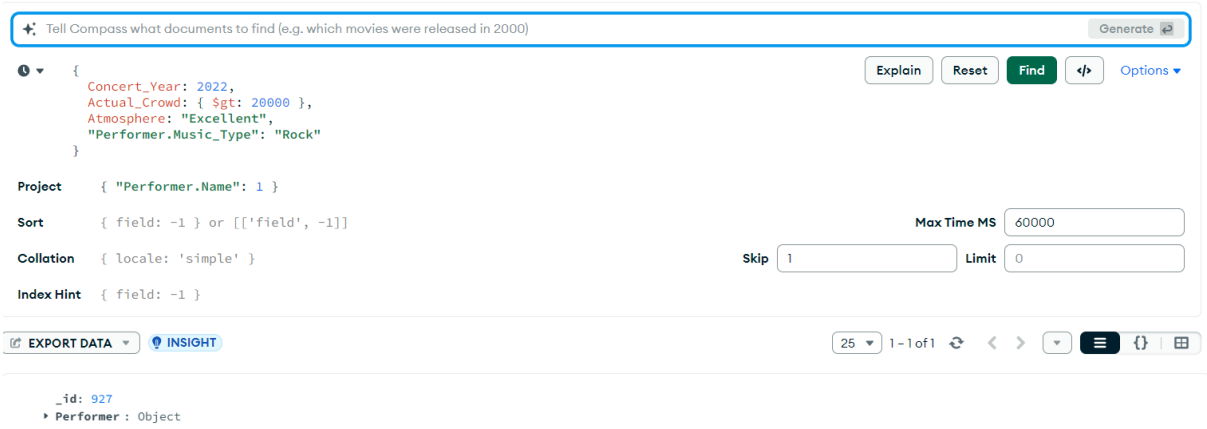

Figure 10: : In this Figure, MongoDB Compass demonstrates how the skip and limit options work when querying documents. The query in this example is filtering concerts in 2022 with an "Excellent" atmosphere and "Rock" music type, and where the actual crowd is greater than 20,000.

Here's what we can see:

1. **Skip**: Skip is used to skip or neglect given number of documents from the output.
    - The skip field is set to 1, which means that MongoDB will skip the first document in the query result and return the second document.
    - This is useful for pagination — for example, when you have a list of results and want to show the next set of results after the first page.
2. **Limit**: However Limit is used to limit the records upto given number.

    for eg, In the above figure if I remove skip: 1 and provide limit: 1 it will limit the output upto 1 document so the result will be only 1 document.

6 Appendix

6.1 Youtube Links

These are the my video recordings and screencasts where I demonstrate how I built and ran my MongoDB queries and aggregation pipelines. The screencasts walk through my project setup, queries, and results using MongoDB Shell, Compass.

- Screencast 1: MongoDB Shell + Aggregation Queries

  Screencast1 demonstrates aggregation query 5 step by step

- Screencast 2: MongoDB Compass

  Screencast2 demonstrates aggregation query 4 step by step using compass

6.2 Reference Links

Below are some of the most helpful learning resources I used while working on this project:

- MongoDB Essential Training – LinkedIn Learning
  This course helped me understand MongoDB basics, indexes, queries, and aggregation.
- MongoDB Compass Tutorial Playlist – YouTube (Net Ninja)
  Easy-to-follow visual guides for using MongoDB Compass effectively.
- MongoDB Official Documentation
  This was my main reference for syntax, operators, and understanding aggregation stages.