A book cover with a blue and white background

Description automatically generated

Contents

[1.0 Project Overview 3](#_Toc164941345)

[1.1.0 MongoDB 3](#_Toc164941346)

[1.1.1 Importance of Embedded Documents and Arrays in MongoDB 4](#_Toc164941347)

[1.2 NoSQL vs SQL 4](#_Toc164941348)

[2.0 Stage I 6](#_Toc164941349)

[2.1 Populate the Database 6](#_Toc164941350)

[2.2 Document Count 7](#_Toc164941351)

[2.3 Find the First Document 7](#_Toc164941352)

[3.0 Stage 2 8](#_Toc164941353)

[3.1 Mongo Shell - Simple Queries 8](#_Toc164941354)

[Question 1 8](#_Toc164941355)

[Question 2 10](#_Toc164941356)

[Question 3 11](#_Toc164941357)

[Question 4 13](#_Toc164941358)

[Question 5 14](#_Toc164941359)

[Question 6 16](#_Toc164941360)

[Question 7 18](#_Toc164941361)

[Question 8 20](#_Toc164941362)

[Question 9 23](#_Toc164941363)

[Question 10 25](#_Toc164941364)

[Question 11 28](#_Toc164941365)

[Question 12 30](#_Toc164941366)

[Question 13 32](#_Toc164941367)

[Question 14 34](#_Toc164941368)

[Question 15 38](#_Toc164941369)

[Question 16 40](#_Toc164941370)

[Question 17 43](#_Toc164941371)

[Question 18 46](#_Toc164941372)

[Question 19 48](#_Toc164941373)

[Question 20 51](#_Toc164941374)

[3.2 Mongo Shell – Aggregation Framework 54](#_Toc164941375)

[Question 1 54](#_Toc164941376)

[Question 2 57](#_Toc164941377)

[Question 3 59](#_Toc164941378)

[Question 4 63](#_Toc164941379)

[Question 5 66](#_Toc164941380)

[Question 6 [Screencast\_1] 70](#_Toc164941381)

[Question 7 73](#_Toc164941382)

[Question 8 75](#_Toc164941383)

[4.0 Stage 4 77](#_Toc164941384)

[4.1 Compass– Overview 77](#_Toc164941385)

[4.2 Compass Simple Queries 78](#_Toc164941386)

[Question 1 78](#_Toc164941387)

[Question 2 79](#_Toc164941388)

[Question 3 80](#_Toc164941389)

[4.3 Compass– Aggregation Framework 82](#_Toc164941390)

[Question 1 82](#_Toc164941391)

[Question 2 85](#_Toc164941392)

[Question 3 87](#_Toc164941393)

[4.4 Atlas Overview 90](#_Toc164941394)

[4.5 Atlas Queries 92](#_Toc164941395)

[Question 1 92](#_Toc164941396)

[Question 2 93](#_Toc164941397)

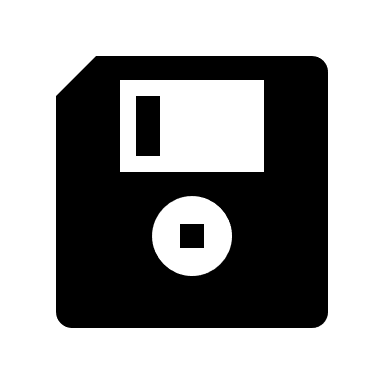
[Question 3 [Screencast 2] 95](#_Toc164941398)

[4.4. Reflection 97](#_Toc164941399)

[Bibliography 98](#_Toc164941400)

[Appendix 100](#_Toc164941401)

# 1.0 Project Overview

[](https://tusmm-my.sharepoint.com/personal/a00275664_student_tus_ie/Documents/Data_Analytics_%20Sem_2/Advanced_Databases/Project_March/Edited/Msc2024_CVL_E_Kennedy.json)The project showcases practical skills with MongoDB, a NoSQL database, by building a substantial collection of Irish concert venue data. The “ConcertVenueLog” collection, featuring 1,000 documents, documents key information about Irish artists and their gross sales in Irish venues. The data will be stored in a MongoDB database named “ConcertVenueLog\_Db”

.

Hyperlink to CVL.json file

## 1.1.0 MongoDB

MongoDB is a NoSQL database that uses JSON and BSON-like documents with dynamic schemas for data storage and retrieval. It's widely used in various industries for its flexibility and scalability. MongoDB allows for efficient handling of unstructured and semi-structured data, making it a popular choice for modern applications requiring flexible data models [1]. MongoDB has many features that allow it to run efficiently and with better scalability. Zhao et al performed a comparative study between MongoDB and MySQL highlighting the differences and key features between SQL-based and NoSQL document-oriented databases [2]. Some of these include utilising indices, aggregation (similar to the SQL Group By clause) and load-balancing, where MongoDB distributes a database system across a cluster of machines. MongoDB can also operate as a file storage system, utilising the function GridFS [3]. Queries are arguably easier to perform in MongoDB because it is a document-oriented database rather than an RDMS.

### Importance of Embedded Documents and Arrays in MongoDB

In the created collection ConcertVenueLog, there are two arrays and two embedded documents.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Feature | Field Values | Description |
| Goodie\_Bag | Array | Refill Pad, USB Key, Highlighter, Torch, Peaked Cap | Promotional items or giveaways given to attendees without cost, associated with the artist(s), or venue. |
| Venue | Embedded Document | Name, Type, Capacity | Information regarding the venue’s name, type of venue and estimated capacity. |
| Merchandise | Array | CD, Tee Shirt, Hoodies, Photo, Hat | Promotional items available to attendees for purchase. |
| Sponsor | Embedded Document | Name, Sector, Headquarters | Information regarding the sponsor of the event’s name, established sector and headquarters location. |

Table 1; CVL file Arrays and Embedded Documents

Arrays help a user maintain order in their database. Arrays provide structured storage capabilities as well as multiple ways to filter data efficiently. Effectively querying array elements is pivotal for developers to extract meaningful insights from the database. This is completed using commands such as $slice, $all and $in [4].

Embedded documents are documents that are nested within another document in MongoDB. Their primary purpose is to allow information to become quick and easy to find with fewer queries becoming necessary to be performed. Data is located within relevant documents and easily uncovered [5].

## 1.2 NoSQL vs SQL

In the case of Oracle, an RDMS, employing object-oriented features, such as user-defined data types, this restricts the query process and causes the process to become more rigid and inflexible [6]. Oracle is a relational database management system (RDBMS) that uses SQL (Structured Query Language) to interact with and manipulate data. It's widely used in various industries for storing, managing, and retrieving information. The relationships between tables in Oracle allow for efficient data organisation and retrieval based on the connections between different entities. Similar to MongoDB, features such as database recovery, disaster recovery, backup, and replication are integrated into the database engine, ensuring enterprise-level reliability and availability [7].

A screenshot of a computer

Description automatically generated

Table 2; Table Detailing Shared Concepts Between Oracle and MongoDB. Source: [8]

Oracle and MongoDB are both powerful databases, but they have different strengths and are used in different ways. Oracle is a relational database management system (RDBMS), which means it organizes data into tables with rows and columns. This allows for strong consistency, integrity, and support for complex queries. MongoDB, on the other hand, is a NoSQL database that does not have a fixed schema. This makes it more flexible and easier to scale

According to a study conducted by Boicea et al, MongoDB offers consistency, durability, and conditional atomicity, whilst Oracle Database offers integrity features that MongoDB does not. However, both databases have their place in the modern data landscape, and the choice between them depends on the specific needs of the application [8].

# 2.0 Stage I

## 2.1 Populate the Database

After accessing the MongoDB folder through the Command Prompt, the *mongoimport* feature is utilised to populate the collection with the ConcertVenueLog data (1000 documents).

Cd C:\Program Files\MongoDb\tools\100\bin\

A black and white text

Description automatically generated

Figure 1; Importation of CVL file

The below code address is used to import the CVL JSON file, creating the collection ConcertVenueLog and the database ConcertVenueLog\_Db.

mongoimport c:\datafiles\Msc2024\_CVL\_E\_Kennedy.json -d ConcertVenueLog\_Db -c ConcertVenueLog --jsonArray

A screenshot of a computer

Description automatically generated

Figure 2; Confirmation Snip of Successful Import

### 2.2 Document Count

Upon successful import, the documents are immediately counted using the Mongo *count()* seen below.

db.ConcertVenueLog.count()

db.ConcertVenueLog.countDocuments()A computer screen shot of a computer

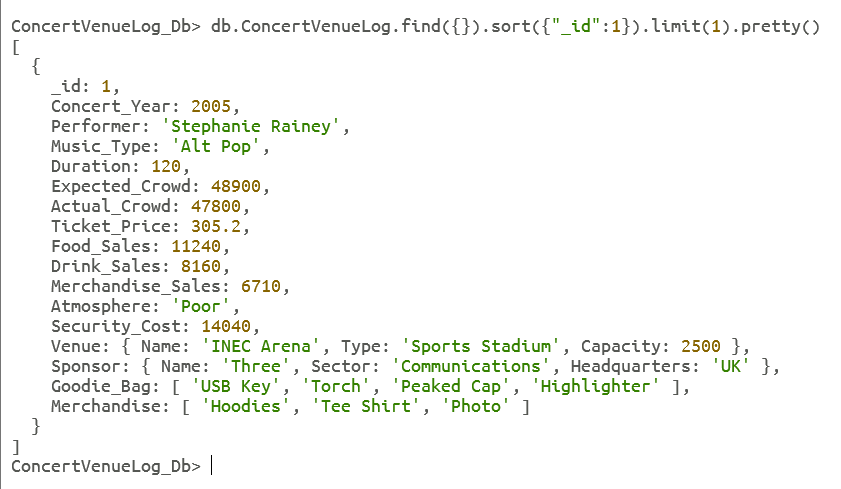
Description automatically generated

A screenshot of a phone

Description automatically generated

Figure 3; Two MongoDB Count Methods

### 2.3 Find the First Document

The first document is located, shown below.

db.ConcertVenueLog.find({}).sort({"\_id ":1}).limit(1).pretty()

# 3.0 Stage 2

## 3.1 Mongo Shell - Simple Queries

### Question 1

List the concert details for the performer Stephanie Rainey for the year 2005 where Alt Pop was performed [9].

//Q1

db.ConcertVenueLog.find ({

$and: [

{“Concert\_Year”: 2005 } ,

{“Performer” : “Stephanie Rainey”} ,

{“Music\_Type” : “Alt Pop”}

]})//.count()

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

### Question 2

Query the database ConcertVenueLog according to the following specifications:

concerts that took place between 2020 and 2022 (ii) the type of music performed was either alternative pop music or Irish folk music (iii) the expected attendance was at least 1000 and (iv) the ticket price was at maximum €200.

//Q2

db.ConcertVenueLog.find({

$and: [

{ Concert\_Year: { $gte: 2020, $lte: 2022 } },

{ Music\_Type: { $in: [“Alt Pop” , “Irish Folk”]}},

{ Expected\_Crowd: { $gte: 1000 } },

{ Ticket\_Price: { $lte: 200 } }

]})//.count()

A screenshot of a computer program

Description automatically generated

### Question 3

Group the atmosphere ‘Superb’ by the performer and music type performed [10] [11].

//Q3

db.ConcertVenueLog.aggregate([

{$match: { Atmosphere: “Superb” }},

{$group: {\_id: { Performer : “$Performer”, Music\_Type: “$Music\_Type”, Atmosphere: “$Atmosphere”}}}

])

//Q3

db.ConcertVenueLog.aggregate([

{$match: { Atmosphere: “Superb” }},

{$group: {\_id: { Performer : “$Performer”, Music\_Type: “$Music\_Type”, Atmosphere: “$Atmosphere”}}},

{ $count: “count”} ])

**A screenshot of a computer

Description automatically generatedA computer screen shot of a computer code

Description automatically generated**

### Question 4

Query how many performers are listed in the database [12].

//Qs4

db.ConcertVenueLog.aggregate ([

{$group: {\_id: "$Performer"}},

{$count: "Performer"}

])

//Q4

db.ConcertVenueLog.aggregate ([

{$group: {\_id: "$Performer"}},

{$count: "Performer"},

{$count: "count"}

])

A screen shot of a computer program

Description automatically generated

### Question 5

Query the database to find documents of concerts that took place where the venue name contains the word “arena”. The sponsor of the concert must have been Aer Lingus and the concert took place before 2015. Order by concert date ascending [13] [14].

//Qs5

db.ConcertVenueLog.aggregate([

{$match:

{$and: [

{ "Venue.Name": { $regex: "arena", $options: "i" } },

{ Concert\_Year: { $lt: 2015 } },

{ "Sponsor.Name": { $in: ["Aer Lingus"] } }

]}},

{ $sort: { "Concert\_Year": 1 } }

])

//Q5

db.ConcertVenueLog.aggregate([

{$match:

{$and: [

{ "Venue.Name": { $regex: "arena", $options: "i" } },

{ Concert\_Year: { $lt: 2015 } },

{ "Sponsor.Name": { $in: ["Aer Lingus"] } }

]}},

{ $sort: { "Concert\_Year": 1 } },

{$count: "count"}])

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

### Question 6

Find the average drink, merchandise and food sales for each venue type that contains the string “th”. Find the total revenue for each venue type (excluding ticket sales) [15] [16].

//Q6

db.ConcertVenueLog.aggregate([

{$match: { "Venue.Name": /th/ }},

{$group: {

\_id: "$Venue.Type",

averageDrinkSales: { $avg: "$Drink\_Sales" },

averageMerchSales: { $avg: "$Merchandise\_Sales" },

averageFoodSales: { $avg: "$Food\_Sales" },

totalsales: {

$sum:{

$add: [ "$Food\_Sales" , "$Merchandise\_Sales" , "$Drink\_Sales"]}

}}},

{$project: {

\_id: 1,

averageDrinkSales: { $round: ["$averageDrinkSales", 2] },

averageMerchSales: { $round: ["$averageMerchSales",2] },

averageFoodSales: { $round:[ "$averageFoodSales", 2] },

totalsales: { $round:[ "$totalsales", 2 ]}

}}])

//Q6

db.ConcertVenueLog.aggregate([

{$match: { "Venue.Name": /th/ }},

{$group: {

\_id: "$Venue.Type",

averageDrinkSales: { $avg: "$Drink\_Sales" },

averageMerchSales: { $avg: "$Merchandise\_Sales" },

averageFoodSales: { $avg: "$Food\_Sales" },

totalsales: {

$sum:{

$add: [ "$Food\_Sales" , "$Merchandise\_Sales" , "$Drink\_Sales"]}

}}},

{$project: {

\_id: 1,

averageDrinkSales: { $round: ["$averageDrinkSales", 2] },

averageMerchSales: { $round: ["$averageMerchSales",2] },

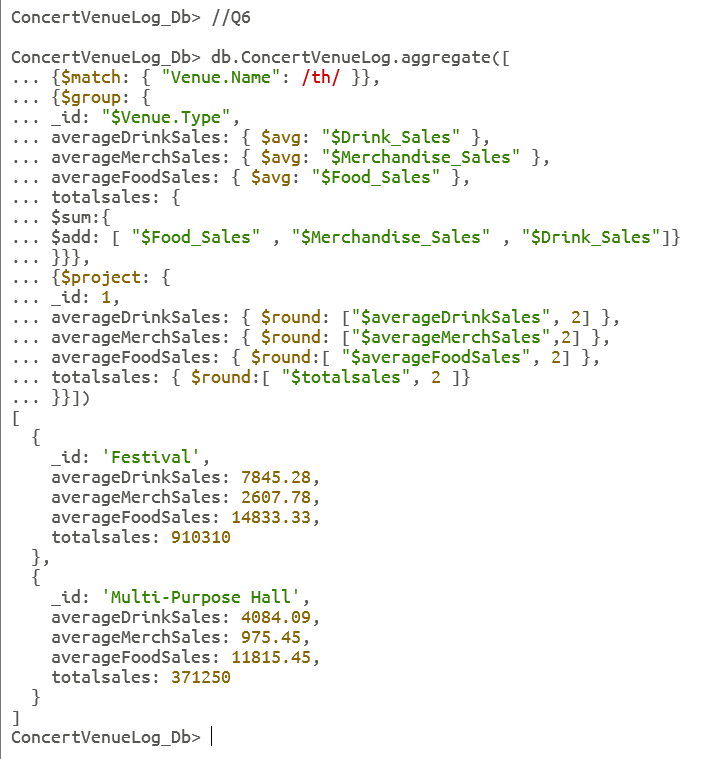
averageFoodSales: { $round:[ "$averageFoodSales", 2] },

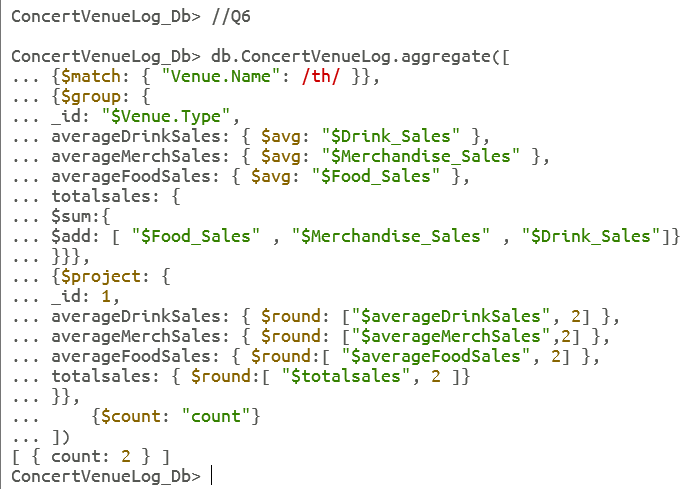
totalsales: { $round:[ "$totalsales", 2 ]}

}},

{$count: "count"}

])





### Question 7

Group the concerts by concert year using ‘$bucket’. Calculate the count of concerts, the average attendance and the total revenue (excluding ticket sales). Order by concert year, and round the resulting figures [15] [16] [17].

//Q7

db.ConcertVenueLog.aggregate([

{$bucket: {

groupBy: "$Concert\_Year",

boundaries: [2000, 2005, 2010, 2015, 2020, 2024],

default: "Other",

output: {

count: { $sum: 1 },

averageAttendance: { $avg: "$Actual\_Crowd" },

totalSales: { $sum: { $add: ["$Drink\_Sales", "$Merchandise\_Sales", "$Food\_Sales"] } }

}}},

{$sort: { \_id: 1 } *// 1 is representative of the bucket field, not Concert\_Year*

},

{$project: {

\_id: 1,

count: 1,

averageAttendance: { $round: ["$averageAttendance", 2] },

totalSales: { $round: ["$totalSales", 2] }

}}])

//Q7

db.ConcertVenueLog.aggregate([

{$bucket: {

groupBy: "$Concert\_Year",

boundaries: [2000, 2005, 2010, 2015, 2020, 2024],

default: "Other",

output: {

count: { $sum: 1 },

averageAttendance: { $avg: "$Actual\_Crowd" },

totalSales: { $sum: { $add: ["$Drink\_Sales", "$Merchandise\_Sales", "$Food\_Sales"] } }

}}},

{$sort: { \_id: 1 } *// 1 is representative of the bucket field, not Concert\_Year*

},

{$project: {

\_id: 1,

count: 1,

averageAttendance: { $round: ["$averageAttendance", 2] },

totalSales: { $round: ["$totalSales", 2] }

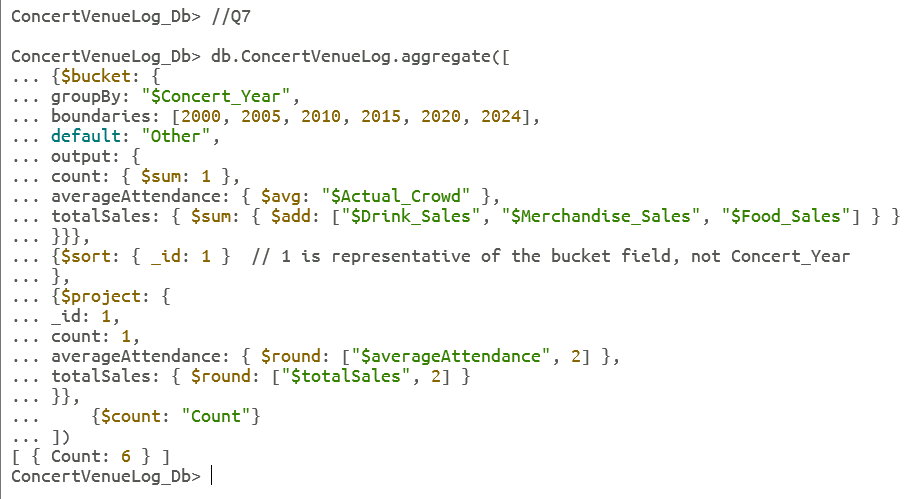
}},

{$count: "Count"}

])

A screenshot of a computer code

Description automatically generated



### Question 8

Query which concerts have (i) goodie bags containing either USB Keys or Highlighters, and do not have a headquarters in the UK, (ii) have a sponsor sector belonging to Energy or Travel (iii) have a Poor atmosphere (iv) have an actual attendance greater than an expected attendance and (v) have a security cost less than 5000. Sort by actual crowd in descending order [18].

//8

db.ConcertVenueLog.find({

$and: [

{ "Goodie\_Bag": { $elemMatch: { $in: ["USB Key", "Highlighter"]} } },

{ "Sponsor.Headquarters": { $ne: "UK" } },

{ $or: [

{ "Sponsor.Sector": "Energy" },

{ "Sponsor.Sector": "Travel" }

]},

{ "Atmosphere": { $eq: "Poor" } },

{ $expr: { $gt: [ "$Actual\_Crowd" , "$Expected\_Crowd" ] }},

{ "Security\_Cost": { $lt: 5000 } }

]}).sort({ "Actual\_Crowd": -1 })//.count()

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Question 9

Calculate the moving average of ticket prices for Hozier concerts between 2000 and 2010. Group by Performer and Concert Year [19] [20] [21] [22].

//Q9

db.ConcertVenueLog.aggregate([

{ $match:

{ Performer: "Hozier", Concert\_Year: { $gte: 2000, $lte: 2010 } } },

{ $group:

{ \_id: "$Performer", data: { $push: { Concert\_Year: "$Concert\_Year", Ticket\_Price: "$Ticket\_Price" } } } },

{ $unwind: "$data" },

{ $group:

{ \_id: { Performer: "$\_id", Concert\_Year: "$data.Concert\_Year" }, data: { $push: "$data" }, movingAverage: { $avg: "$data.Ticket\_Price" } } },

{ $project:

{ \_id: 0, Performer: "$\_id.Performer", Concert\_Year: "$\_id.Concert\_Year", data: 1, movingAverage: 1 } }

])

//Q9

db.ConcertVenueLog.aggregate([

{ $match:

{ Performer: "Hozier", Concert\_Year: { $gte: 2000, $lte: 2010 } } },

{ $group:

{ \_id: "$Performer", data: { $push: { Concert\_Year: "$Concert\_Year", Ticket\_Price: "$Ticket\_Price" } } } },

{ $unwind: "$data" },

{ $group:

{ \_id: { Performer: "$\_id", Concert\_Year: "$data.Concert\_Year" }, data: { $push: "$data" }, movingAverage: { $avg: "$data.Ticket\_Price" } } },

{ $project:

{ \_id: 0, Performer: "$\_id.Performer", Concert\_Year: "$\_id.Concert\_Year", data: 1, movingAverage: 1 } },

{$count: "count"}

])

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

### Question 10

Find and display the data types of venue type, venue capacity and music type from venues that had either (i) Irish rap played, and has ‘Temple’ in the venue name or (ii) a security cost greater than 15000 and has ‘Temple’ in the venue name [23] [21] [15]

//Q10

db.ConcertVenueLog.aggregate([

{$match:{

$or:[

{$and:[

{"Music\_Type":"IrishRap"},

{"Venue.Name":{$regex:"Temple"}}

]},

{$and:

[{ "Security\_Cost": { $gt: 15000 } },

{ "Venue.Name": { $regex: "Temple" } }

]}]}},

{$project:

{"Venue.Name": 1,

Music\_Type: 1,

Security\_Cost: 1,

VenueType: { $type: "$Venue.Type" },

VenueCapacity: { $type: "$Venue.Capacity" }

}}])

//Q10

db.ConcertVenueLog.aggregate([

{$match:{

$or:[

{$and:[

{"Music\_Type":"IrishRap"},

{"Venue.Name":{$regex:"Temple"}}

]},

{$and:

[{ "Security\_Cost": { $gt: 15000 } },

{ "Venue.Name": { $regex: "Temple" } }

]}]}},

{$project:

{"Venue.Name": 1,

Music\_Type: 1,

Security\_Cost: 1,

VenueType: { $type: "$Venue.Type" },

VenueCapacity: { $type: "$Venue.Capacity" }

}},

{$count: "count"}

])

A screenshot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

### Question 11

Find and display concerts that took place in sports stadiums between 2020 and 2023 by Bono or the Kneecaps that had a crowd attendance of at least 20000 and merchandise sales greater than 5000. Order by concert year [19] [24]

//Q11

db.ConcertVenueLog.find({

"Concert\_Year": { $gte: 2020, $lte: 2023 },

"Venue.Type": "Sports Stadium",

"Performer": { $in: ["Bono", "The Kneecaps"] },

"Actual\_Crowd": { $gte: 20000 },

"Merchandise\_Sales": { $gt: 5000 }

}).sort({ "Concert\_Year": 1 })

//Q11

db.ConcertVenueLog.find({

"Concert\_Year": { $gte: 2020, $lte: 2023 },

"Venue.Type": "Sports Stadium",

"Performer": { $in: ["Bono", "The Kneecaps"] },

"Actual\_Crowd": { $gte: 20000 },

"Merchandise\_Sales": { $gt: 5000 }

}).sort({ "Concert\_Year": 1 }).count()

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

### Question 12

Display a list of concerts where (i) the venue type performed was at a bar, the actual crowd was less than 5000, the goodie bag is not null and the merchandise is not null, the Concert Year was either less than 2003, greater than 2020 or the Performer was Bono or Imelda May. Only display the Venue Name, Performer, maximum of two Goodie Bags, and maximum of three Merchandise items. Order by Actual Crowd in descending order [15], [25], [26].

//Q12

db.ConcertVenueLog.aggregate([

{$match: {

"Venue.Type": "Bar",

"Actual\_Crowd": { $lt: 5000 },

"Goodie\_Bag": { $exists: true, $not: { $size: 0 } },

"Merchandise": { $exists: true, $not: { $size: 0 } },

$or: [

{ "Concert\_Year": { $lt: 2003 } },

{ "Concert\_Year": { $gt: 2020 } } ],

"Performer": { $in: ["Bono", "Imelda May"] }

}},

{$project:

{\_id: 0,

VenueName: "$Venue.Name",

Performer: 1,

GoodieBagItems: { $slice: ["$Goodie\_Bag", 2] },

MerchandiseItems: { $slice: ["$Merchandise", 3] }

}},

{$sort: { "Actual\_Crowd": -1 }

}])

//Q12

db.ConcertVenueLog.aggregate([

{$match: {

"Venue.Type": "Bar",

"Actual\_Crowd": { $lt: 5000 },

"Goodie\_Bag": { $exists: true, $not: { $size: 0 } },

"Merchandise": { $exists: true, $not: { $size: 0 } },

$or: [

{ "Concert\_Year": { $lt: 2003 } },

{ "Concert\_Year": { $gt: 2020 } }],

"Performer": { $in: ["Bono", "Imelda May"] }

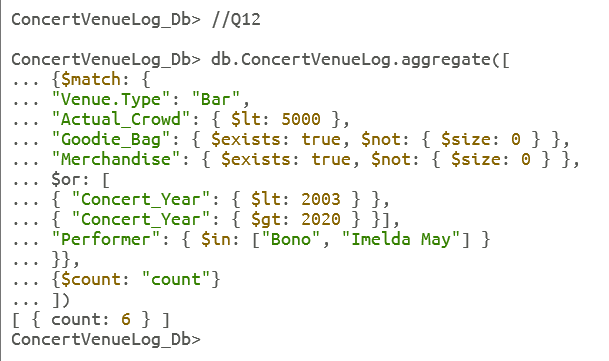
}},

{$count: "count"}

])

A screenshot of a computer program

Description automatically generated



### Question 13

Find and display concerts between the years 2007 and 2010 for the artists Hozier, the Kneecaps and Andrea Corr. Only display the performances held in sports stadiums and theatres with an audience of at least 20000 and a merchandise range of at least three items. Order in ascending order by concert year [9], [13], [23].

//Q13

db.ConcertVenueLog.find({

$and: [

{"Concert\_Year": { $gte: 2007, $lte: 2010 } },

{$or: [

{ "Venue.Type": "Sports Stadium" },

{ "Venue.Type": "Theatre" } ]},

{$or: [

{ "Performer": "Hozier" },

{ "Performer": "The Kneecaps" },

{ "Performer": "Andrea Corr" }

]},

{"Actual\_Crowd": { $gte: 20000 }},

{$expr: { $gte: [ { $size: "$Merchandise" }, 3 ] }

}]

}).sort({ "Concert\_Year": 1 })

//Q13

db.ConcertVenueLog.find({

$and: [

{"Concert\_Year": { $gte: 2007, $lte: 2010 }},

{$or: [

{ "Venue.Type": "Sports Stadium" },

{ "Venue.Type": "Theatre" }

]},

{$or: [

{ "Performer": "Hozier" },

{ "Performer": "The Kneecaps" },

{ "Performer": "Andrea Corr" }

]},

{"Actual\_Crowd": { $gte: 20000 }},

{$expr: { $gte: [ { $size: "$Merchandise" }, 3 ] } }

]

}).sort({ "Concert\_Year": 1 }).count()

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

### Question 14

Filters for concerts between 2000 and 2011, sponsored by a business in Media, in a venue with a capacity greater than 2500 where a USB key was among the Goodie Bag gifts. Calculate the revenue made from ticket sales, food sales, drink sales[27], [28].

//Q14

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gt: 2000, $lt: 2011 },

"Sponsor.Sector": "Media",

"Venue.Capacity": { $gt: 2500 },

"Goodie\_Bag": { $in: ["USB Key"] }

}},

{$project: {

\_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Actual\_Revenue: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] },

Food\_Sales\_capita: { $divide: ["$Food\_Sales", "$Actual\_Crowd"] },

Drink\_Sales\_capita: { $divide: ["$Drink\_Sales", "$Actual\_Crowd"] },

Merch\_Sales\_capita: { $divide: ["$Merchandise\_Sales", "$Actual\_Crowd"] }

}},

{$project: {

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales\_capita: 1,

Drink\_Sales\_capita: 1,

Merch\_Sales\_capita: 1,

Revenue\_PerCapita: {

$add: ["$Food\_Sales\_capita", "$Drink\_Sales\_capita", "$Merch\_Sales\_capita"]

}}},

{$sort: { Revenue\_PerCapita: -1 }

}])

//Q14

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gt: 2000, $lt: 2011 },

"Sponsor.Sector": "Media",

"Venue.Capacity": { $gt: 2500 },

"Goodie\_Bag": { $in: ["USB Key"] }

}},

{$project: {

\_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Actual\_Revenue: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] },

Food\_Sales\_capita: { $divide: ["$Food\_Sales", "$Actual\_Crowd"] },

Drink\_Sales\_capita: { $divide: ["$Drink\_Sales", "$Actual\_Crowd"] },

Merch\_Sales\_capita: { $divide: ["$Merchandise\_Sales", "$Actual\_Crowd"] }

}},

{$project: {

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales\_capita: 1,

Drink\_Sales\_capita: 1,

Merch\_Sales\_capita: 1,

Revenue\_PerCapita: {

$add: ["$Food\_Sales\_capita", "$Drink\_Sales\_capita", "$Merch\_Sales\_capita"]

}}},

{$sort: { Revenue\_PerCapita: -1 }},

{$count: "count"}])

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

### Question 15

Find and display concerts that took place in bars after 2015. The music type must contain ‘folk’, the ticket price at most 250 and the attendance should be at most 200. Hoodies and tee shirts must have been provided as merchandise also [23]

//Q15

db.ConcertVenueLog.find({

"Concert\_Year": { $gte: 2000 },

"Venue.Type": "Bar",

"Music\_Type": { $regex: /^folk/i },

"Actual\_Crowd": { $lte: 200 },

$or: [

{ "Ticket\_Price": { $lte: 250 } },

{ "Merchandise": { $all: ["Hoodies", "Tee Shirt"] } } ]

}).sort({ "Actual\_Crowd": -1 })

//Q15

db.ConcertVenueLog.find({

"Concert\_Year": { $gte: 2000 },

"Venue.Type": "Bar",

"Music\_Type": { $regex: /^folk/i },

"Actual\_Crowd": { $lte: 200 },

$or: [

{ "Ticket\_Price": { $lte: 250 } },

{ "Merchandise": { $all: ["Hoodies", "Tee Shirt"] } } ]

}).sort({ "Actual\_Crowd": -1 }).count()

A screen shot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

### Question 16

Calculate the average spending per person at a concert. Group by performer and average spending per person. Only display the average spending and clearly state if the performer’s fans are profitable to the vendor and sponsors of the concert. They are considered profitable if they spend over €25 at a random performer’s concert. Order by average spending per capita in descending order [27].

//Q16

db.ConcertVenueLog.aggregate([

{$project: { \_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Total\_Spending\_Per\_Person: {

$divide: [{ $sum: ["$Food\_Sales",

"$Merchandise\_Sales",

"$Drink\_Sales"] },

"$Actual\_Crowd" ]}}},

{$group: {\_id: "$Performer",

Average\_Spending\_Per\_Person: { $avg: "$Total\_Spending\_Per\_Person" }

}},

{$project: {

\_id: 1,

Average\_Spending\_Per\_Person: 1,

Profitable: {$cond: {

if: { $gte: ["$Average\_Spending\_Per\_Person", 25] },

then: "Yes",

else: "No"}} }},

{$sort: {Average\_Spending\_Per\_Person:-1}}

])

//Q16

db.ConcertVenueLog.aggregate([

{$project: { \_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Total\_Spending\_Per\_Person: {

$divide: [{ $sum: ["$Food\_Sales",

"$Merchandise\_Sales",

"$Drink\_Sales"] },

"$Actual\_Crowd" ]}}},

{$group: {\_id: "$Performer",

Average\_Spending\_Per\_Person: { $avg: "$Total\_Spending\_Per\_Person" }

}},

{$project: {

\_id: 1,

Average\_Spending\_Per\_Person: 1,

Profitable: {$cond: {

if: { $gte: ["$Average\_Spending\_Per\_Person", 25] },

then: "Yes",

else: "No"}} }},

{$sort: {Average\_Spending\_Per\_Person:-1}},

{$count : "count"} ])

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated

### Question 17

Find and display details for Niall Horen concerts between 2010 and 2022 with a ticket price greater than €300. The crowd attendance must have been at least 5000 with a security cost of between 6000 and 12000. Only display the performer's name, atmosphere and goodie bag size. State clearly if the concert was likely considered a success or not. Online fan comments suggest if a goodie bag contains at least 3 items, it satisfies a majority of fans regardless of the atmosphere [26], [29].

//Q17

db.ConcertVenueLog.aggregate([

{$match: {Performer: "Niall Horan",

Concert\_Year: { $gte: 2010, $lte: 2022 },

Actual\_Crowd: { $gte: 5000 },

Ticket\_Price: { $not: { $lt: 300 } },

Security\_Cost: { $gte: 6000, $lte: 12000 }

}},

{$project: {

\_id: 0,

Performer: 1,

Atmosphere: 1,

Goodie\_Bag\_Size: { $size: "$Goodie\_Bag" },

Concert\_Rating: {

$cond: {

if: { $eq: ["$Atmosphere", "Superb"] },

then: {$cond: {

if: { $gte: ["$Goodie\_Bag\_Size", 3] },

then: "Brilliant with Lots of Free Merch",

else: "Brilliant but Not Much Free Merch"

}},

else: {$cond: {

if: { $gte: ["$Goodie\_Bag\_Size", 3] },

then: "Not Very Good but Lots of Free Merch",

else: "Not Very Good and Not Much Free Merch"

}} }} }}

])

//Q17

db.ConcertVenueLog.aggregate([

{$match: {Performer: "Niall Horan",

Concert\_Year: { $gte: 2010, $lte: 2022 },

Actual\_Crowd: { $gte: 5000 },

Ticket\_Price: { $not: { $lt: 300 } },

Security\_Cost: { $gte: 6000, $lte: 12000 }

}},

{$project: {

\_id: 0,

Performer: 1,

Atmosphere: 1,

Goodie\_Bag\_Size: { $size: "$Goodie\_Bag" },

Concert\_Rating: {

$cond: {

if: { $eq: ["$Atmosphere", "Superb"] },

then: {$cond: {

if: { $gte: ["$Goodie\_Bag\_Size", 3] },

then: "Brilliant with Lots of Free Merch",

else: "Brilliant but Not Much Free Merch"

}},

else: {$cond: {

if: { $gte: ["$Goodie\_Bag\_Size", 3] },

then: "Not Very Good but Lots of Free Merch",

else: "Not Very Good and Not Much Free Merch"

}} }} }},

{$count : "count" }

])

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

### Question 18

Find and display Stephanie Rainey's performances taking place in a venue with a capacity of at least 500, with a security cost of less than 10000 with a poor atmosphere. Order by performer and skip the first 5 results [24].

//18

db.ConcertVenueLog.find({

"Venue.Capacity": { $gte: 500 },

"Performer": "Stephanie Rainey",

"Security\_Cost": { $lt: 10000 },

"Atmosphere": "Poor" },

{"Performer": 1,

"Venue.Capacity": 1,

"Atmosphere": 1,

"\_id": 0 }

).sort({ "Performer": 1 }) .skip(5)

//count with skip

db.ConcertVenueLog.aggregate([

{$match:

{"Venue.Capacity": { $gte: 500 },

"Performer": "Stephanie Rainey",

"Security\_Cost": { $lt: 10000 },

"Atmosphere": "Poor"}},

{$sort: { "Performer": 1 }},

{$skip: 5},

{$count: "count"}

])

//count no skip

db.ConcertVenueLog.aggregate([

{$match:

{"Venue.Capacity": { $gte: 500 },

"Performer": "Stephanie Rainey",

"Security\_Cost": { $lt: 10000 },

"Atmosphere": "Poor"}},

{$sort: { "Performer": 1 }},

{$count: "count"}])

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

### Question 19

Create a guide that displays performers by their average crowd attendance. Group by performer and average crowd. State clearly if they are a popular artist. They are considered popular if they have an average crowd of at least 10000. Order by average crowd in descending order [18], [29].

//Q19

db.ConcertVenueLog.aggregate([

{$group: {

\_id: "$Performer", // Group by performer

averageActualCrowd: { $avg: "$Actual\_Crowd" }

}},

{$addFields: {

popularity: {

$cond: {

if: { $gte: ["$averageActualCrowd", 10000] },

then: "Popular",

else: "Not Popular"

}}}},

{$project: {

\_id: 0,

Performer: "$\_id",

averageActualCrowd: 1,

popularity: 1}},

{$sort: { averageActualCrowd: -1 }}

])

//Q19

db.ConcertVenueLog.aggregate([

{$group: {

\_id: "$Performer", // Group by performer

averageActualCrowd: { $avg: "$Actual\_Crowd" }

}},

{$addFields: {

popularity: {

$cond: {

if: { $gte: ["$averageActualCrowd", 10000] },

then: "Popular",

else: "Not Popular"

}}}},

{$project: {

\_id: 0,

Performer: "$\_id",

averageActualCrowd: 1,

popularity: 1}},

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

{$count: "count" }

])

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

Question 20

Find the performances where food sales were at least 5000, drink sales were at least 6000, merchandise sales less than 4000 and the performance took place between 2003 and 2007. Ensure the sponsors were either Guinness, Electric Ireland, Supermacs or Allied Irish Bank. Rank the results by Sponsor in order by concert year in ascending order [30]

//Q20

db.ConcertVenueLog.aggregate([

{$match: {

Food\_Sales: { $gte: 5000 },

Drink\_Sales: { $gte: 6000 },

Merchandise\_Sales: { $lt: 4000 },

Concert\_Year: { $gte: 2003, $lte: 2007 },

"Sponsor.Name": { $in: ['Guinness', 'Electric Ireland', 'Supermacs', 'Allied Irish Bank'] }

}},

{ $setWindowFields: {

partitionBy: "$Sponsor.Name",

sortBy: { Concert\_Year: 1 },

output: {

rankYearBySponsor: {

$rank: {}

}}}},

{$project: {

"Sponsor.Name": 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Concert\_Year: 1,

rankYearBySponsor: 1

}}

] )

//Q20

db.ConcertVenueLog.aggregate([

{$match: {

Food\_Sales: { $gte: 5000 },

Drink\_Sales: { $gte: 6000 },

Merchandise\_Sales: { $lt: 4000 },

Concert\_Year: { $gte: 2003, $lte: 2007 },

"Sponsor.Name": { $in: ['Guinness', 'Electric Ireland', 'Supermacs', 'Allied Irish Bank'] }

}},

{ $setWindowFields: {

partitionBy: "$Sponsor.Name",

sortBy: { Concert\_Year: 1 },

output: {

rankYearBySponsor: {

$rank: {}

}}}},

{$project: {

"Sponsor.Name": 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Concert\_Year: 1,

rankYearBySponsor: 1

}}

] ).toArray().length

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer code

Description automatically generated

3.2 Mongo Shell – Aggregation Framework

### Question 1

Find and display concerts where the sponsors were either Electric Ireland, Supermacs or Aer Lingus. For each sponsor, calculate the average actual crowd attendance, the average ticket revenue, the average expected crowd, and the average difference in crowd attendance. Only display the sponsor name, the average crowd attendance, the average ticket revenue, the average expected crowd and the average difference in crowd numbers. Order by average crowd attendance in ascending order [27].

//Q1

db.ConcertVenueLog.aggregate([

{$match:

{"Sponsor.Name": { $in: ["Electric Ireland", "Supermacs", "Aer Lingus"] }

}}, // match Sponsor name to Electric Ireland, Supermacs or Aer Lingus

{$group: {\_id: "$Sponsor.Name", //group by Sponsor Name

avgCrowdAttendance: { $avg: "$Actual\_Crowd" }, //find average actual crowd

avgIncomePerConcert: { $avg: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } }, // find the average ticket revenue

avgExpectedCrowd: { $avg: "$Expected\_Crowd" }, //find average expected crowd

avgDifferenceCrowd: { $avg: { $subtract: ["$Actual\_Crowd", "$Expected\_Crowd"] } } // find the average difference in actual crowd and expected crowd

}},

{$project: {\_id: 0, // exclude id number

Sponsor: "$\_id", // include Sponsor name

avgCrowdAttendance: 1, //include average crowd attendance

avgIncomePerConcert: 1, //include average income

avgExpectedCrowd: 1, //include average expected crowd

avgDifferenceCrowd: 1 //include average differnence

}},

{$sort: { avgCrowdAttendance: 1 } } ]) //sort by average crowd attendance ascending order

//Q1

db.ConcertVenueLog.aggregate([

{$match:

{"Sponsor.Name": { $in: ["Electric Ireland", "Supermacs", "Aer Lingus"] }

}},

{$group: {\_id: "$Sponsor.Name",

avgCrowdAttendance: { $avg: "$Actual\_Crowd" },

avgIncomePerConcert: { $avg: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

avgExpectedCrowd: { $avg: "$Expected\_Crowd" },

avgDifferenceCrowd: { $avg: { $subtract: ["$Actual\_Crowd", "$Expected\_Crowd"] } }

}},

{$project: {\_id: 0,

Sponsor: "$\_id",

avgCrowdAttendance: 1,

avgIncomePerConcert: 1,

avgExpectedCrowd: 1,

avgDifferenceCrowd: 1 }},

{$sort: { avgCrowdAttendance: 1 }},

{$count: "count"}])



A screenshot of a computer program

Description automatically generated

### Question 2

Find and display 8 sample documents from the collection that (i) have the venue type ‘Sports Stadium’, (ii) have either a ‘Good’ or ‘Superb’ atmosphere and (iii) are sponsored by either ‘Supermacs’, ‘Dunnes’, ‘Aviva’ or ‘Penney’s’. Display only relevant information, as well as performer name, ticket price and the actual crowd attendance figures [19], [31].

//Q2

db.ConcertVenueLog.aggregate([

{$match:

{"Venue.Type": "Sports Stadium", // filter by sports stadiums

Atmosphere: { $in: ["Good", "Superb"] }, //filter atmosphere is either good or superb

"Sponsor.Name": { $in: ["Supermacs", "Dunnes", "Aviva", "Penney's"] } } //sponsor name is either Supermacs, Dunnes, Aviva or Penneys

},

{$sample: { size: 8 }}, //sample size of 8

{$project:

{\_id: 0, //exclude id number

Performer: 1, //include performer

Ticket\_Price: 1, //include ticket price

Actual\_Crowd: "$Actual\_Crowd", //include actual crowd

Atmosphere: 1, //include atmosphere

Sponsor\_Name: "$Sponsor.Name", // include sponsor name

Venue\_Type: "$Venue.Type" }}, // include venue type

{$sort: { Actual\_Crowd: -1 } //order by actual crowd in descending order

}])

//Q2

db.ConcertVenueLog.aggregate([

{$match:

{"Venue.Type": "Sports Stadium",

Atmosphere: { $in: ["Good", "Superb"] },

"Sponsor.Name": { $in: ["Supermacs", "Dunnes", "Aviva", "Penney's"] } }

},

{$sample: { size: 8 }},

{$project:

{\_id: 0,

Performer: 1,

Ticket\_Price: 1,

Actual\_Crowd: "$Actual\_Crowd",

Atmosphere: 1,

Sponsor\_Name: "$Sponsor.Name",

Venue\_Type: "$Venue.Type" }},

{$sort: { Actual\_Crowd: -1 }

},

{$count: "count"}

])

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Question 3

Filters for concerts between 2000 and 2011, sponsored by a business in Media, in a venue with a capacity greater than 2500 where a USB key was among the Goodie Bag gifts. Calculate the revenue made from ticket sales, food sales, drink sales [27].

//Q3

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gt: 2000, $lt: 2011 }, //greater than 2000, less than 2011

"Sponsor.Sector": "Media", //Sponsor sector is Media

"Venue.Capacity": { $gt: 2500 }, //Venue Capacity greater than 2500

"Goodie\_Bag": { $in: ["USB Key"] } //USB Key included in array

}},

{$project: {

\_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Actual\_Revenue: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] }, //multiply actual crowd by ticket price

Food\_Sales\_capita: { $divide: ["$Food\_Sales", "$Actual\_Crowd"] }, divide food sales by actual crowd

Drink\_Sales\_capita: { $divide: ["$Drink\_Sales", "$Actual\_Crowd"] }, //divide drink sales by actual crowd

Merch\_Sales\_capita: { $divide: ["$Merchandise\_Sales", "$Actual\_Crowd"] } //divide merch sales by actual crowd

}},

{$project: {

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales\_capita: 1,

Drink\_Sales\_capita: 1,

Merch\_Sales\_capita: 1,

Revenue\_PerCapita: {

$add: ["$Food\_Sales\_capita", "$Drink\_Sales\_capita", "$Merch\_Sales\_capita"]

}}}, // add up figures to create Revenue Per Capita

{$sort: { Revenue\_PerCapita: -1 } // order in descending order

}])

//Q3

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gt: 2000, $lt: 2011 },

"Sponsor.Sector": "Media",

"Venue.Capacity": { $gt: 2500 },

"Goodie\_Bag": { $in: ["USB Key"] }

}},

{$project: {

\_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Actual\_Revenue: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] },

Food\_Sales\_capita: { $divide: ["$Food\_Sales", "$Actual\_Crowd"] },

Drink\_Sales\_capita: { $divide: ["$Drink\_Sales", "$Actual\_Crowd"] },

Merch\_Sales\_capita: { $divide: ["$Merchandise\_Sales", "$Actual\_Crowd"] }

}},

{$project: {

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales\_capita: 1,

Drink\_Sales\_capita: 1,

Merch\_Sales\_capita: 1,

Revenue\_PerCapita: {

$add: ["$Food\_Sales\_capita", "$Drink\_Sales\_capita", "$Merch\_Sales\_capita"]

}}},

{$sort: { Revenue\_PerCapita: -1 }},

{$count: "count"}])

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

### Question 4

Create a new collection from the ConcertVenueLog, comprised of performances that have taken place in a sports stadium before the year 2010 or after 2015, with an attendance of more than 20000. Goodie bags and the range of merchandise sold are not limited to two and only performances featuring Enya and Allie Sherlock must be considered. Only display the collection by venue name, performer, and the first two items in goodie bags and merchandise range. Order in descending order by total concerts performed in each venue [32].

//Q4

db.ConcertVenueLog.aggregate([

{$match: {

"Venue.Type": "Sports Stadium",

"Actual\_Crowd": { $gt: 20000 },

"Goodie\_Bag": { $exists: true, $not: { $size: 2 } }, // filter for existing goodie\_bag arrays, where size is not 2

"Merchandise": { $exists: true, $not: { $size: 2 } }, // filter for existing merchandise arrays, where size is not two

$or: [

{ "Concert\_Year": { $lt: 2010 } },

{ "Concert\_Year": { $gt: 2015 } }

],

"Performer": { $in: ["Enya", "Allie Sherlock"] }

}},

{

$project: {

VenueName: "$Venue.Name",

Performer: 1,

GoodieBagItems: { $slice: ["$Goodie\_Bag", 2] }, // only show the first two items in goodie bag

MerchandiseItems: { $slice: ["$Merchandise", 2] } // only show first two items in merchandise array

}},

{$group: {

\_id: {

VenueName: "$VenueName",

Performer: "$Performer"},

GoodieBagItems: { $first: "$GoodieBagItems" }, //retains the first goodie bag items for each group

MerchandiseItems: { $first: "$MerchandiseItems" }, // retains the first merchandise items for each group

TotalConcerts: { $sum: 1 }, // sum of perfromances

}},

{$sort: { "TotalConcerts": -1 }},

{$out: "ConcertSumDetails"

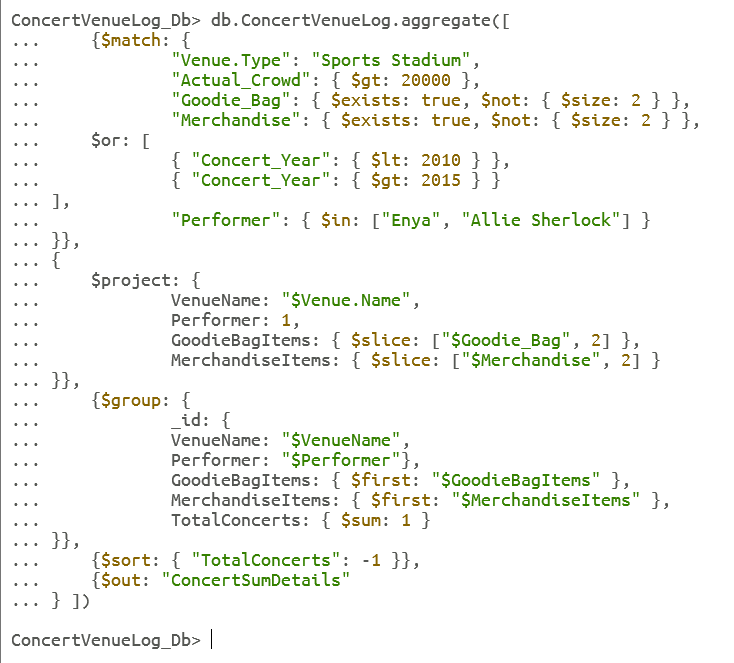
} ])

show collections

db.ConcertSumDetails.find()

db.ConcertSumDetails.count()

db.ConcertSumDetails.drop()



A screenshot of a computer program

Description automatically generated

A close-up of a music concert

Description automatically generated

### Question 5

Provide sponsor details for the top three most expensive sponsors regarding security costs for performances from the year 2015. Calculate the total security cost, total ticket revenue, total merchandise sales, maximum ticket prices, minimum ticket prices and average crowd attendance for the three sponsors. Also display how many different venue types, and years the sponsor has been active. Order by security cost in descending order [33], [34].

//Q5

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gte: 2015 }

}},

{$group: {

\_id: "$Sponsor.Name",

totalSecurityCost: { $sum: "$Security\_Cost" },

totalRevenue: { $sum: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

totalMerchandiseSales: { $sum: "$Merchandise\_Sales" },

maxTicketPrice: { $max: "$Ticket\_Price" }, //maximum ticket price

minTicketPrice: { $min: "$Ticket\_Price" }, //minimum ticket price

averageCrowd: { $avg: "$Actual\_Crowd" }, //average crowd attendance

venueTypes: { $addToSet: "$Venue.Type" }, // create new array to find unique Type values

concertYears: { $addToSet: "$Concert\_Year" } // create new array to find unique Year values

}},

{$project: {

\_id: 1,

totalSecurityCost: 1,

totalRevenue: 1,

totalMerchandiseSales: 1,

maxTicketPrice: 1,

minTicketPrice: 1,

averageCrowd: 1,

venueTypesCount: { $size: "$venueTypes" }, // display size of array

concertYearsCount: { $size: "$concertYears" } //display size of array

}},

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

{$limit: 3} //limit to top three

])

//Q5

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gte: 2015 }

}},

{$group: {

\_id: "$Sponsor.Name",

totalSecurityCost: { $sum: "$Security\_Cost" },

totalRevenue: { $sum: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

totalMerchandiseSales: { $sum: "$Merchandise\_Sales" },

maxTicketPrice: { $max: "$Ticket\_Price" },

minTicketPrice: { $min: "$Ticket\_Price" },

averageCrowd: { $avg: "$Actual\_Crowd" },

venueTypes: { $addToSet: "$Venue.Type" },

concertYears: { $addToSet: "$Concert\_Year" }

}},

{$project: {

\_id: 1,

totalSecurityCost: 1,

totalRevenue: 1,

totalMerchandiseSales: 1,

maxTicketPrice: 1,

minTicketPrice: 1,

averageCrowd: 1,

venueTypesCount: { $size: "$venueTypes" },

concertYearsCount: { $size: "$concertYears" }

}},

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

{$limit: 3}

]).toArray().length //adjusted count to account for arrays

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

### Question 6 [Screencast\_1]

How much ticket revenue was generated from Irish Rap concerts held in venues with a capacity between 10,000 and 20,000, during the years 2015 to 2023?

If the concert took place between 2020 and 2023, adjust the ticket revenue by 20% to account for the circumstance that COVID-19 restrictions had not taken place and more attendees would have been permitted.

State clearly if restrictions were in place or not.

//Q6

db.ConcertVenueLog.aggregate([

{$match: {

"Music\_Type": "Irish Rap",

"Venue.Capacity": { $gte: 10000, $lte: 20000 },

"Concert\_Year": { $gte: 2015, $lte: 2023 }

}},

{$addFields: { // creates new field

Ticket\_Revenue: { $multiply: ["$Ticket\_Price", "$Actual\_Crowd"] }, //new field

Covid\_Restrictions: { // new field

$cond: { // condition

if: { $gte: ["$Concert\_Year", 2020] }, // if Concert Year greater than or equal to 2020

then: "Yes", // then display yes

else: "No" // else display no

}}}},

{$addFields: // creates new field

{Adjusted\_Ticket\_Revenue: {

$cond: { // condition

if: { $eq: ["$Covid\_Restrictions", "Yes"] }, if field is equal to yes

then: { $multiply: ["$Ticket\_Revenue", 1.2] }, //multiply by 1.2

else: "$Ticket\_Revenue" // else display Ticket Revenue field

} } }},

{$project: {

\_id: 0,

Concert\_Year: 1,

Performer: 1,

Ticket\_Price: 1,

Actual\_Crowd: 1,

Ticket\_Revenue: 1,

Adjusted\_Ticket\_Revenue: 1,

Covid\_Restrictions: 1

}},

{$sort: { Adjusted\_Ticket\_Revenue: -1 }

} ])

//Q6

db.ConcertVenueLog.aggregate([

{$match: {

"Music\_Type": "Irish Rap",

"Venue.Capacity": { $gte: 10000, $lte: 20000 },

"Concert\_Year": { $gte: 2015, $lte: 2023 }

}},

{$addFields: {

Ticket\_Revenue: { $multiply: ["$Ticket\_Price", "$Actual\_Crowd"] },

Covid\_Restrictions: {

$cond: {

if: { $gte: ["$Concert\_Year", 2020] },

then: "Yes",

else: "No"

}}}},

{$addFields:

{Adjusted\_Ticket\_Revenue: {

$cond: {if: { $eq: ["$Covid\_Restrictions", "Yes"] },

then: { $multiply: ["$Ticket\_Revenue", 1.2] },

else: "$Ticket\_Revenue"

} } }},

{$project: {

\_id: 0,

Concert\_Year: 1,

Performer: 1,

Ticket\_Price: 1,

Actual\_Crowd: 1,

Ticket\_Revenue: 1,

Adjusted\_Ticket\_Revenue: 1,

Covid\_Restrictions: 1

}},

{$sort: { Adjusted\_Ticket\_Revenue: -1 }

},

{$count: "count"}

])

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

### Question 7

Divide concerts performed between 2020 and 2023, into five distinct groups based on actual\_crowd size and allow MongoDB to determine the boundaries. For each bucket, calculate the sum of concerts, average ticket price, ticket revenue (actual crowd multiplied by ticket price), minimum crowd attendance and maximum crowd attendance [35].

//Q7

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gte: 2020, $lte: 2023 }

}},

{$bucketAuto: { // automatically groups documents into buckets

groupBy: "$Actual\_Crowd", // buckets grouped by Actual Crowd

buckets: 5, // amount of buckets specified

output: {

"Concerts\_In\_Range": { $sum: 1 }, //amount of concerts between 2020 and 2023

"Average\_Ticket\_Price": { $avg: "$Ticket\_Price" },

"Total\_Revenue": { $sum: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

"Min\_Actual\_Crowd": { $min: "$Actual\_Crowd" },

"Max\_Actual\_Crowd": { $max: "$Actual\_Crowd" }

}}} ])

//Q7

db.ConcertVenueLog.aggregate([

{$match: {

"Concert\_Year": { $gte: 2020, $lte: 2023 }

}},

{$bucketAuto: {

groupBy: "$Actual\_Crowd",

buckets: 5,

output: {

"Concerts\_In\_Range": { $sum: 1 },

"Average\_Ticket\_Price": { $avg: "$Ticket\_Price" },

"Total\_Revenue": { $sum: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

"Min\_Actual\_Crowd": { $min: "$Actual\_Crowd" },

"Max\_Actual\_Crowd": { $max: "$Actual\_Crowd" }

}}},

{$count: "count"}

])

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Question 8

Calculate the ratio of crowd attendance to ticket price for Andrea Corr’s performances for the year 2020.

//Q8

db.ConcertVenueLog.aggregate([

{$match: {

"Performer": "Andrea Corr",

"Concert\_Year": 2020

}},

{$project: {

\_id: 0,

ratio: { $divide: ["$Actual\_Crowd", "$Ticket\_Price"] } // calculated ratio for ticket price to actual crowd

}} ])

//Q8

db.ConcertVenueLog.aggregate([

{$match: {

"Performer": "Andrea Corr",

"Concert\_Year": 2020

}},

{$project: {

\_id: 0,

ratio: { $divide: ["$Actual\_Crowd", "$Ticket\_Price"] }

}},

{$count: "count"}

])

A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated

# 4.0 Stage 4

## 4.1 Compass– Overview

MongoDB Compass is downloaded together with MongoDB Shell. Once you open Compass, it will request a connection string. You can use a local host 27017 and simply press "connect" to be directed to the main Compass page.

A database is created by selecting the ‘+’ icon beside Databases and a collection is created by selecting the ‘+’ icon beside the newly created database or ‘Create Collection’. A screenshot of a computer

Description automatically generated



Figure 1; Database Overview

Inside the new collection, data can be imported from the library using ‘Add Data’ and read into Compass [36].

A screenshot of a computer

Description automatically generated

Figure 2; Importing Data

## 4.2 Compass Simple Queries

### Question 1

[Adapted from MongoDB Simple Queries Q1]

List the concert details for the performer Stephanie Rainey for the year 2005 where Alt Pop was performed [9].

//Q1 Compass Version

{"$and": [

{"Concert\_Year": 2005},

{"Performer": "Stephanie Rainey"},

{"Music\_Type": "Alt Pop"}

]}

A screenshot of a computer

Description automatically generated

### Question 2

[Adapted from MongoDB Simple Queries Q8]

Query which concerts have (i) goodie bags containing either USB Keys or Highlighters, and do not have a headquarters in the UK, (ii) have a sponsor sector belonging to Energy or Travel (iii) have a Poor atmosphere (iv) have an actual attendance greater than an expected attendance and (v) have a security cost less than 5000. Sort by actual crowd in descending order [18].

//Q2 Compass Version

{"$and": [

{"Goodie\_Bag": {

"$elemMatch": {

"$in": ["USB Key", "Highlighter"]

}}},

{"Sponsor.Headquarters": {

"$ne": "UK"

}},

{"$or": [

{"Sponsor.Sector": "Energy"},

{"Sponsor.Sector": "Travel"}

]},

{"Atmosphere": {

"$eq": "Poor"

}},

{"$expr": {

"$gt": ["$Actual\_Crowd", "$Expected\_Crowd"]

}},

{"Security\_Cost": {

"$lt": 5000

}} ]}

A screenshot of a computer

Description automatically generated

### Question 3

[Adapted from MongoDB Simple Queries Q16]

Calculate the average spending per person at a concert. Group by performer and average spending per person. Only display the average spending and clearly state if the performer’s fans are profitable to the vendor and sponsors of the concert. They are considered profitable if they spend over €25 at a random performer’s concert. Order by average spending per capita in descending order [27].

//Q3

*[Stage 1] -* $project

{

Performer: 1,

Actual\_Crowd: 1,

Total\_Spending\_Per\_Person: {

$divide: [

{ $sum: ["$Food\_Sales", "$Merchandise\_Sales", "$Drink\_Sales"] },

"$Actual\_Crowd"

] }

}

*[Stage 2]* - $group

{

\_id: "$Performer",

Average\_Spending\_Per\_Person: { $avg: "$Total\_Spending\_Per\_Person" }

}

*[Stage 3]* - $project

{

\_id: 1,

Average\_Spending\_Per\_Person: 1,

Profitable: {

$cond: {

if: { $gte: ["$Average\_Spending\_Per\_Person", 25] },

then: "Yes",

else: "No"

}}

}

*[Stage 4]* - $sort

{

Average\_Spending\_Per\_Person: -1

}

A screenshot of a computer

Description automatically generated

## 4.3 Compass– Aggregation Framework

### Question 1

[Adapted from MongoDB Aggregation Framework Q1]

Find and display concerts where the sponsors were either Electric Ireland, Supermacs or Aer Lingus. For each sponsor, calculate the average actual crowd attendance, the average ticket revenue, the average expected crowd, and the average difference in crowd attendance. Only display the sponsor name, the average crowd attendance, the average ticket revenue, the average expected crowd and the average difference in crowd numbers. Order by average crowd attendance in ascending order [27].

// Stage Version Q1

[Stage 1] - $match

{

"Sponsor.Name":{$in: ["Electric Ireland", "Supermacs", "Aer Lingus"]}

}

[Stage 2] - $group

{

\_id: "$Sponsor.Name",

avgCrowdAttendance: { $avg: "$Actual\_Crowd" },

avgIncomePerConcert: { $avg: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

avgExpectedCrowd: { $avg: "$Expected\_Crowd" },

avgDifferenceCrowd: { $avg: { $subtract: ["$Actual\_Crowd", "$Expected\_Crowd"] } }

}

[Stage 3] - $project

{

\_id: 0,

Sponsor: "$\_id",

avgCrowdAttendance: 1,

avgIncomePerConcert: 1,

avgExpectedCrowd: 1,

avgDifferenceCrowd: 1

}

[Stage 4] - $sort

{

avgCrowdAttendance: 1

}

// Text Version Q1

[

{$match: {"Sponsor.Name": {$in: ["Electric Ireland","Supermacs","Aer Lingus",],},},},

{

$group: {

\_id: "$Sponsor.Name",

avgCrowdAttendance: {

$avg: "$Actual\_Crowd",},

avgIncomePerConcert: {$avg: {$multiply: ["$Actual\_Crowd", "$Ticket\_Price", ], }, },

avgExpectedCrowd: { $avg: "$Expected\_Crowd", },

avgDifferenceCrowd: {$avg: { $subtract: [ "$Actual\_Crowd", "$Expected\_Crowd", ], }, },

},},

{

$project: {

\_id: 0,

Sponsor: "$\_id",

avgCrowdAttendance: 1,

avgIncomePerConcert: 1,

avgExpectedCrowd: 1,

avgDifferenceCrowd: 1,

},},

{$sort: {avgCrowdAttendance: 1,},},

]

A screenshot of a computer program

Description automatically generated

### Question 2

[Adapted from MongoDB Aggregation Framework Q8]

Divide concerts performed between 2020 and 2023, into five distinct groups based on actual\_crowd size and allow MongoDB to determine the boundaries. For each bucket, calculate the sum of concerts, average ticket price, ticket revenue (actual crowd multiplied by ticket price), minimum crowd attendance and maximum crowd attendance [35].

// Stage Version Q2

*[Stage 1]* - $match

{

"Venue.Type": "Sports Stadium",

"Atmosphere": { "$in": ["Good", "Superb"] },

"Sponsor.Name": { "$in": ["Supermacs", "Dunnes", "Aviva", "Penney's"] }

}

*[Stage 2]* - $sample

{

size: 8

}

*[Stage 3]* - $project

{

\_id: 0,

Performer: 1,

Ticket\_Price: 1,

Actual\_Crowd: 1,

Atmosphere: 1,

Sponsor\_Name: "$Sponsor.Name",

Venue\_Type: "$Venue.Type"

}

*[Stage 4]* - $sort

{Actual\_Crowd: -1}

// Text Version Q2

[

{$match: {"Venue.Type": "Sports Stadium",

Atmosphere: {$in: ["Good", "Superb"],

},

"Sponsor.Name": {$in: ["Supermacs","Dunnes","Aviva","Penney's",

],

},},},

{$sample: {size: 8,},},

{$project: {

\_id: 0,

Performer: 1,

Ticket\_Price: 1,

Actual\_Crowd: 1,

Atmosphere: 1,

Sponsor\_Name: "$Sponsor.Name",

Venue\_Type: "$Venue.Type",

},},

{$sort: {Actual\_Crowd: -1,},}, ]

A screenshot of a computer

Description automatically generated

### Question 3

[Adapted from MongoDB Aggregation Framework Q6]

// Stage Version Q3

*[Stage 1] -* $match

{

"Music\_Type": { "$eq": "Irish Rap" },

"Venue.Capacity": { "$gte": 10000, "$lte": 20000 },

"Concert\_Year": { "$gte": 2015, "$lte": 2023 }

}

*[Stage 2]* - $addFields

{

"Ticket\_Revenue": { "$multiply": ["$Ticket\_Price", "$Actual\_Crowd"] },

"Covid\_Restrictions": {

"$cond": {

"if": { "$gte": ["$Concert\_Year", 2020] },

"then": "Yes",

"else": "No"

}}

}

*[Stage 3]* - $addFields

{

Adjusted\_Ticket\_Revenue: {

$cond: {

if: { $eq: ["$Covid\_Restrictions", "Yes"] },

then: { $multiply: ["$Ticket\_Revenue", 1.2] },

else: "$Ticket\_Revenue"

}}

}

*[Stage 4]* - $project

{

Concert\_Year: 1,

Performer:1,

Ticket\_Price:1,

Actual\_Crowd:1,

Ticket\_Revenue:1,

Adjusted\_Ticket\_Revenue:1,

Covid\_Restrictions:1

}

*[Stage 5]* - $sort

{

Adjusted\_Ticket\_Revenue: -1

}

// Text Version Q3

[

{$match: {Music\_Type: {$eq: "Irish Rap",},

"Venue.Capacity": {$gte: 10000,$lte: 20000,},

Concert\_Year: {$gte: 2015,$lte: 2023,

},},},

{$addFields: {Ticket\_Revenue: {$multiply: ["$Ticket\_Price","$Actual\_Crowd",],},

Covid\_Restrictions: {$cond:

{if: {$gte: ["$Concert\_Year", 2020],},

then: "Yes",

else: "No",

},},},},

{$addFields: {Adjusted\_Ticket\_Revenue: {$cond: {

if: {$eq: ["$Covid\_Restrictions", "Yes"],},

then: {$multiply: ["$Ticket\_Revenue", 1.2],},

else: "$Ticket\_Revenue",}, },}, },

{$project: {

Concert\_Year: 1,

Performer: 1,

Ticket\_Price: 1,

Actual\_Crowd: 1,

Ticket\_Revenue: 1,

Adjusted\_Ticket\_Revenue: 1,

Covid\_Restrictions: 1,

},

},

{$sort:{Adjusted\_Ticket\_Revenue: -1,},},

]

A screenshot of a computer program

Description automatically generated

## 4.4 Atlas Overview

Atlas was set up, using MongoDB compass. An admin user was created and the connection string was copied into Compass’ connection portal.

A screenshot of a computer

Description automatically generated

Figure 4; Guide to Compass / Atlas Connection

Once Compass has connected to Atlas, the CVL database within Compass will appear within Atlas.

A screenshot of a computer

Description automatically generated

Figure 5; CVL Database in Atlas

Atlas has many unique features compared to Compass and MongoDB Shell. Atlas provides automation for many administrative tasks and provides built-in scalability [37]. Additional features such as data visualisation are also available for users. A quick graph, shown below, was produced from the CVL database information. An efficient tool encompanied with this is also the use of natural language processing (NLP) which allows a user to explain what type of graph they would like and what they would like to achieve. Atlas suggests some suitable. This feature is also available in Compass for the creation of queries, however only upon connection to Atlas can it be utilised.

A graph with green line

Description automatically generated

Figure 6; Example of Atlas's visualisation features with a calculated field – Concert\_Year vs Ticket\_Revenue

## 4.5 Atlas Queries

### Question 1

[Adapted from MongoDB Simple Queries Q2]

Query the database ConcertVenueLog according to the following specifications:

concerts that took place between 2020 and 2022 (ii) the type of music performed was either alternative pop music or Irish folk music (iii) the expected attendance was at least 1000 and (iv) the ticket price was at maximum €200.

//Q1

$and: [

{ Concert\_Year: { $gte: 2020, $lte: 2022 } },

{ Music\_Type: { $in: [“Alt Pop” , “Irish Folk”]}},

{ Expected\_Crowd: { $gte: 1000 } },

{ Ticket\_Price: { $lte: 200 } }

]}

A screenshot of a computer

Description automatically generated

### Question 2

[Adapted from MongoDB Aggregation Framework Q3]

//2

[Stage 1] - $match

{

"Concert\_Year": { $gt: 2000, $lt: 2011 },

"Sponsor.Sector": "Media",

"Venue.Capacity": { $gt: 2500 },

"Goodie\_Bag": { $in: ["USB Key"] }

}

[Stage 2] - $project

{

\_id: 0,

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales: 1,

Drink\_Sales: 1,

Merchandise\_Sales: 1,

Actual\_Revenue: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] },

Food\_Sales\_capita: { $divide: ["$Food\_Sales", "$Actual\_Crowd"] },

Drink\_Sales\_capita: { $divide: ["$Drink\_Sales", "$Actual\_Crowd"] },

Merch\_Sales\_capita: { $divide: ["$Merchandise\_Sales", "$Actual\_Crowd"] }

}

[Stage 3] - $project

{

Performer: 1,

Actual\_Crowd: 1,

Ticket\_Price: 1,

Food\_Sales\_capita: 1,

Drink\_Sales\_capita: 1,

Merch\_Sales\_capita: 1,

Revenue\_PerCapita: {

$add:["$Food\_Sales\_capita", "$Drink\_Sales\_capita", "$Merch\_Sales\_capita"]

}}

[Stage 4] - $sort

{ Revenue\_PerCapita: -1 }

A screenshot of a computer

Description automatically generated

### Question 3 [Screencast 2]

[Adapted from MongoDB Aggregation Framework Q5]

Provide sponsor details for the top three most expensive sponsors regarding security costs for performances from the year 2015.

Calculate the total security cost, total ticket revenue, total merchandise sales, maximum ticket prices, minimum ticket prices and average crowd attendance for the three sponsors.

Also display how many different venue types, and years the sponsor has been active. Order by security cost in descending order [33], [34].

//Q3

*[Stage 1]* - $match

{

"Concert\_Year": { $gte: 2015 }

}

*[Stage 2]* - $group

{

id: "$Sponsor.Name",

totalSecurityCost: { $sum: "$Security\_Cost" },

totalRevenue: { $sum: { $multiply: ["$Actual\_Crowd", "$Ticket\_Price"] } },

totalMerchandiseSales: { $sum: "$Merchandise\_Sales" },

maxTicketPrice: { $max: "$Ticket\_Price" },

minTicketPrice: { $min: "$Ticket\_Price" },

averageCrowd: { $avg: "$Actual\_Crowd" },

venueTypes: { $addToSet: "$Venue.Type" },

concertYears: { $addToSet: "$Concert\_Year" }

}

*[Stage 3]* - $project

{\_

id: 1,

totalSecurityCost: 1,

totalRevenue: 1,

totalMerchandiseSales: 1,

maxTicketPrice: 1,

minTicketPrice: 1,

averageCrowd: 1,

venueTypesCount: { $size: "$venueTypes" },

concertYearsCount: { $size: "$concertYears" }

}

*[Stage 4]* - $sort

{ totalSecurityCost: -1 }

*[Stage 5]* - $limit

3

A screenshot of a phone

Description automatically generated

## 4.4. Reflection

During the initial stages of familiarising myself with JSON format and setting up the database, I encountered some challenges. To address this, I opted to download a pre-produced JSON file and conducted comparisons to identify any errors that I might have missed.

Moreover, adapting to the new format proved to be quite overwhelming. Being relatively new to MongoDB and NoSQL databases in general, I found it challenging to navigate the various types of parentheses and differentiate between embedded documents and arrays. However, with practice and consistent exposure to simple queries, I was able to overcome this hurdle.

To enhance my understanding and proficiency in MongoDB, I dedicated my free time to participating in online learning tutorials and watched numerous YouTube videos on the topic. These efforts helped me accumulate knowledge and improve my ability to create queries for the project. Along the way, I encountered minor setbacks such as incorrect syntax, which occasionally required me to restart queries until they worked properly. Despite these challenges, I persisted through the tough process. In my own opinion, I found Compass faster and easier to work with, especially after discovering the ‘Wizard’ assistant which is quite helpful to beginners in MongoDB. Utilising the ‘Wizard’, there was also a smaller chance of making minor errors, such as spelling or syntax errors.

One particularly challenging moment occurred when I accidentally dropped my database after executing an ‘$out’ query. This led to complications during the subsequent import process. Fortunately, I managed to resolve the issue by reinstalling MongoDB.

# Bibliography

[1] K. Banker, D. Garrett, P. Bakkum, and S. Verch, *MongoDB in Action: Covers MongoDB version 3.0*. Simon and Schuster, 2016.

[2] G. Zhao, W. Huang, S. Liang, and Y. Tang, ‘Modeling MongoDB with Relational Model’, in *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, Xi’an, Shaanxi, China: IEEE, Sep. 2013, pp. 115–121. doi: 10.1109/EIDWT.2013.25.

[3] ‘GridFS — MongoDB Manual’. Accessed: Apr. 08, 2024. [Online]. Available: https://www.mongodb.com/docs/manual/core/gridfs/

[4] ‘MongoDB Query an Array’, GeeksforGeeks. Accessed: Apr. 08, 2024. [Online]. Available: https://www.geeksforgeeks.org/mongodb-query-an-array/

[5] *#14 Embedded Documents | Fundamentals of MongoDB | Complete MongoDB Course 2022*, (Sep. 12, 2022). Accessed: Apr. 10, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=EFWMv\_vRz54

[6] ‘Database Concepts’, Oracle Help Center. Accessed: Apr. 08, 2024. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/18/cncpt/introduction-to-oracle-database.html#GUID-166C1E31-CDBC-47D9-867A-3D4C9AAC837D

[7] T. Lahiri *et al.*, ‘Oracle Database In-Memory: A dual format in-memory database’, in *2015 IEEE 31st International Conference on Data Engineering*, Apr. 2015, pp. 1253–1258. doi: 10.1109/ICDE.2015.7113373.

[8] A. Boicea, F. Radulescu, and L. I. Agapin, ‘MongoDB vs Oracle – Database Comparison’, in *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, Sep. 2012, pp. 330–335. doi: 10.1109/EIDWT.2012.32.

[9] *08 MongoDB Queries - $and Operator*, (Jan. 05, 2020). Accessed: Apr. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=3CXvEaOuLq0

[10] *13 Aggregation Example 5 $match and $group - MongoDB Aggregation Tutorial*, (May 16, 2018). Accessed: Apr. 12, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=r1A-C\_wxi0A

[11] *17 Aggregation Example 8 $count - MongoDB Aggregation Tutorial*, (May 16, 2018). Accessed: Apr. 12, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=jH-5VRnfe2o

[12] *19 Aggregation Example 9 $group and $count - MongoDB Aggregation Tutorial*, (May 16, 2018). Accessed: Apr. 12, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=qj-yUH\_zg0U

[13] *30. Evaluation Operator $expr and $regex operators for querying the documents - MongoDB*, (Aug. 07, 2022). Accessed: Apr. 13, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=d3lsBowZQ6c

[14] *🔥Regex In MongoDB | MongoDB Regular Expressions Explained | MongoDB Tutorial Beginners | SimpliCode*, (Dec. 01, 2022). Accessed: Apr. 13, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=5ABbf2uSxrw

[15] *25 Aggregation Example 13 $project with restructuring - MongoDB Aggregation Tutorial*, (May 16, 2018). Accessed: Apr. 13, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=GG6Ax1B1Ag8

[16] ‘ProQuest Ebook Central - Reader’. Accessed: Apr. 13, 2024. [Online]. Available: https://ebookcentral.proquest.com/lib/aitie/reader.action?docID=5532278&query=#

[17] *65. Understand $bucket Aggregation Pipeline to add boundaries for grouping documents - MongoDB*, (Aug. 26, 2022). Accessed: Apr. 13, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=zhvt8r6Y\_u4

[18] 000, ‘Answer to “Mongo field A greater than field B”’, Stack Overflow. Accessed: Apr. 13, 2024. [Online]. Available: https://stackoverflow.com/a/16042568

[19] D. Bierer, *MongoDB 4 Quick Start Guide: Learn the Skills You Need to Work with the World’s Most Popular NoSQL Database*. Birmingham, UNITED KINGDOM: Packt Publishing, Limited, 2018. Accessed: Apr. 10, 2024. [Online]. Available: http://ebookcentral.proquest.com/lib/aitie/detail.action?docID=5532278

[20] *The Difference Between $set & $push Operators in MongoDB - Adding Array Elements to a Non-Array Key*, (Mar. 03, 2023). Accessed: Apr. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=-x69s54vs00

[21] *MongoDB movieDetails Aggregation Framework 07 $match $unwind $group $sort $out Stages Arrays*, (Dec. 16, 2021). Accessed: Apr. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=VhvIc1zc\_f0

[22] X. Guihot, ‘Answer to “Moving averages with MongoDB’s aggregation framework?”’, Stack Overflow. Accessed: Apr. 20, 2024. [Online]. Available: https://stackoverflow.com/a/70135885

[23] ‘Run aggregation with $match and $or and $regexMatch from JavaScript? - Working with Data’, MongoDB Developer Community Forums. Accessed: Apr. 13, 2024. [Online]. Available: https://www.mongodb.com/community/forums/t/run-aggregation-with-match-and-or-and-regexmatch-from-javascript/213446

[24] *MongoDB Simple Query - Sort, Limit, Skip [MongoDB# 04]*, (Jul. 14, 2023). Accessed: Apr. 19, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=cTQCdo9cpUg

[25] *15 MongoDB $exists and $type query operators*, (Jan. 05, 2020). Accessed: Apr. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=5WxsVOuHQQk

[26] *12 MongoDB Queries - Array operators $all, $size*, (Jan. 05, 2020). Accessed: Apr. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=yDm0SdZHwEk

[27] *MongoDB movieDetails Aggregation Framework 08 Calculations $multiply $divide $subtract $add*, (Dec. 16, 2021). Accessed: Apr. 17, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=JNiDYpm2IM4

[28] *MongoDB movieDetails Aggregation Framework 05 $match $project $group $sort Stages*, (Dec. 16, 2021). Accessed: Apr. 17, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=a8n\_TsBpmqQ

[29] N. Lunn, ‘Answer to “Is there an elseif thing in MongoDB to $cond while aggregating”’, Stack Overflow. Accessed: Apr. 17, 2024. [Online]. Available: https://stackoverflow.com/a/27479572

[30] ‘$rank (aggregation) - MongoDB Manual v7.0’. Accessed: Apr. 23, 2024. [Online]. Available: https://www.mongodb.com/docs/manual/reference/operator/aggregation/rank/

[31] dalanmiller, ‘Answer to “Random Sampling from Mongo”’, Stack Overflow. Accessed: Apr. 20, 2024. [Online]. Available: https://stackoverflow.com/a/35298966

[32] *MongoDB movieDetails Aggregation Framework 04 $match $group $sort $out Stages $avg $max $min*, (Dec. 16, 2021). Accessed: Apr. 17, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=k0FS7ypVOpo

[33] *18 Different documents count methods - MongoDB Aggregation Tutorial*, (May 16, 2018). Accessed: Apr. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=GiQsQX5HGjs

[34] ‘MongoDB - $addToSet Operator’, GeeksforGeeks. Accessed: Apr. 20, 2024. [Online]. Available: https://www.geeksforgeeks.org/mongodb-addtoset-operator/

[35] *MongoDB movieDetails Aggregation Framework 10 $bucketAuto*, (Dec. 16, 2021). Accessed: Apr. 21, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=gXdD-ebW2Nk

[36] *Complete MongoDB Tutorial #4 - Using MongoDB Compass*, (Mar. 22, 2022). Accessed: Apr. 21, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=bJSj1a84I20

[37] ‘MongoDB vs MongoDB Atlas | What are the differences?’, StackShare. Accessed: Apr. 22, 2024. [Online]. Available: https://stackshare.io/stackups/mongodb-vs-mongodb-atlas

Appendix

Links to helpful start-up websites not referenced;

<https://www.linkedin.com/learning/learning-mongodb-17360744/modern-database-and-application-design-with-mongodb?u=42358916>

https://www.linkedin.com/learning/mongodb-essential-training/features-of-mongodb?autoSkip=true&resume=false&u=42358916

<https://docs.mongodb.com/manual/tutorial/getting-started/>

<https://docs.mongodb.com/manual/crud/>

<https://docs.mongodb.com/manual/aggregation/>

Links to YouTube Screencasts;

Screencast 1;

<https://youtu.be/O8g4IbTdduk>

Screencast 2;

<https://youtu.be/m097GHKz8U8>