**CU1)** This query allows customer (e.g. email = ‘ornare@aol.edu’) to check how many movies were watched by a particular profile within their account (e.g. profile.name = ‘Ira Clark’) and how much of the movie they watched (watched until).

db.user.aggregate([ {

'$lookup': {

'from': 'MOVIE',

'localField': 'profiles.history.watched\_movie\_no', 'foreignField': '\_id',

'as': 'Movie'

}

}, {'$unwind': '$profiles'

}, {

'$match': {'$and':[

{'email': 'ornare@aol.edu'}, {'profiles.name': 'Ira Clark'}

]

} }, {

'$project': { 'profiles.history.watched\_movie\_no': 1, 'profiles.history.watched\_movie\_till': 1

} }

]).pretty()

**CU2)** This query allows customer to fetch movie(s) based on a given actor (e.g. ‘Chris Pratt’). This query searches within the movies collection for a match to the actor’s name in the actor field and displays only the fields which are specified in the query: title, actors, and year.

db.movies.find({ "actors": "Chris Pratt"}, {"actors":1, "title":1, 'year':1, \_id:0})

**CU3)** This query allows customer to fetch movie(s)

based on a given genre (e.g. ‘Horror’). This query searches within the movies collection for a match to the given genre in the genre field and displays only the fields which are specified in the query: title, genres, and year.

db.movies.find({ "genres": "Horror"}, {"title":1, "genres":1, 'year':1, \_id:0})

**CU4)** This query allows customer to fetch details of profiles linked to their account. This query searched in the user collection for a match to the given email in the email field, then displays only the fields specified in the query: account email and corresponding profile names.

db.user.find({ "email": "at.nisi@google.ca"}, {"email":1, "profiles.name":1,\_id:0})

**MA1)** This query allows manager to fetch customers whose bill is due on the present day.

db.user.aggregate([ {

'$match': {"billing\_info.due\_date":"2022-05-07T00:00:00"} },

{

'$project': {

'name': 1, 'email': 1

} }

])

**MA2)** This aggregation allows manager to fetch details of monthly subscription numbers, which helps them to get the count of new SMD customers in a given month (e.g. February, 2020). This aggregation looks for a match to the specified date range within the created\_at field, then counts the number of resulting values.

db.user.aggregate([{ $project: { \_id:0,

'created\_at': 1 }

}, {

$match: {

'created\_at': { $gte: '01-02-20', $lte: '28-02-20'

} }

}, {

$count: 'Number of new Customers in February' }])

**MA3)** This aggregation allows manager to fetch details of customer service issue status, which helps them track how many customer service problems are in active/resolved status. This aggregation first projects the desired fields from the *user* collection: customer service status and customer service created date. The aggregation then sorts the results in descending order of created date and ascending alphabetical order of status.

db.user.aggregate([{$project: {

'customer\_service.status': 1,'customer\_service.created\_at': 1, \_id: 0

}}, {$sort: {

'customer\_service.created\_at': -1, 'customer\_service.status': 1} }])

**MA4)** This aggregation allows manager to fetch total number of problems faced by users in a month (e.g. July, 2021). This aggregation first projects the desired fields from the user collection, then looks for a match for the specified date range within the customer service *created\_at* field, then prints total count of results in the new field.

db.user.aggregate([{$project: { 'customer\_service.problem\_desc': 1, 'customer\_service.created\_at': 1, \_id: 0

}}, {$match: { 'customer\_service.created\_at': {

$gte: '2021-07-01',

$lt: '2021-07-31' }

}}, {$count: 'Total Number of Customer Service Issues in July 2021'}])

**MA5)** This aggregation allows manager to fetch list of which employees resolved which customer service issue. This aggregation first projects the desired fields from the user collection, then unwinds the array that has been projected, then looks for a match to the specified status, then sorts the results by ascending order of created date, employee last name, and name of customer service issue.

db.user.aggregate([{$project: { customer\_service: 1,

\_id: 0

}}, {$unwind: {

path: '$customer\_service'

}}, {$match: { 'customer\_service.status': 'Resolved'

}}, {$sort: { 'customer\_service.created\_at': 1, 'customer\_service.handled\_by.lname': 1, 'customer\_service.problem\_desc': 1

}}])

**MA6)** This aggregation allows manager to fetch customer service issue counts by category for a given month (e.g. July, 2021). The aggregation begins by projecting the desired fields from user collection and unwinding the resulting array, then looking for a match to the specified date range within the customer service

created date field, then grouping the results by customer service issue, counting the values in each group, then sorting the results by descending order of counts.

db.user.aggregate([{$project: { customer\_service: 1,

\_id: 0

}}, {$unwind: {

path: '$customer\_service'

}}, {$match: { 'customer\_service.created\_at': {

$gte: '2021-07-01',

$lt: '2021-07-31' }

}}, {$group: {

\_id: '$customer\_service.problem\_desc', NumberComplaintsJuly2021: {

$count: {} }

}}, {$sort: { NumberComplaintsJuly2021: -1

}}])

**MA7)** This aggregation allows manager to fetch which employee handled the most customer service issues in a particular month (for example, July, 2021). The aggregation begins by projecting the desired fields from user collection and unwinding the resulting array, then looking for a match to the specified date range

db.user.aggregate([{$project: { customer\_service: 1

}}, {$unwind: {

path: '$customer\_service'

}}, {$match: { 'customer\_service.created\_at': {

$gte: '2021-07-01',

$lt: '2021-07-31' }

}}, {$group: {

\_id: '$customer\_service.handled\_by.fname', TotalComplaintsHandled: {

$count: {} }

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

}}])

**MA8)** This aggregation allows manager to fetch number of impacted customers in a given month (e.g. July, 2021). The aggregation begins by projecting the desired fields from user collection and unwinding the resulting array, then looking for a match to the specified date range within the customer service created date field, then grouping the results by account email, then counting the total results and printing the results to a new field.

db.user.aggregate([{ $project: {

email: 1,

'customer\_service.created\_at': 1 }

* }, {  
  $match: {  
  'customer\_service.created\_at': { $gte: '2021-01-01',  
  $lte: '2021-07-31'  
  } }
* }, {  
  $group: {  
  \_id: '$email' }
* }, {  
  $count: 'Number Impacted Customers in July'  
  }])

**MA9)** This aggregation allows manager to rank all archived customer service issues by number, sorted highest to lowest. This helps manager identify the most frequent customer service issues to-date. The aggregation begins by projecting the desired fields from user collection and unwinding the resulting array, then grouping the results by customer service issue, counting the values in each group, then sorting the results by descending order of counts.

db.user.aggregate([{ $project: {

customer\_service: 1, \_id: 0

}

* }, {  
  $unwind: {  
  path: '$customer\_service'  
  }
* }, {  
  $group: {  
  \_id: '$customer\_service.problem\_desc', NumberComplaints: {  
  $count: {}
* }, {  
  $sort: {  
  NumberComplaints: -1 }  
  }])

**MA10)** This aggregation allows manager to identify the top 5 watched genres in SMD. The aggregation begins by joining both user and movies collections using $lookup, then unwinds both the movie and genres arrays. It then groups by genres, counts the values in each group, and sorts the results by descending order of counts.

db.user.aggregate([{$lookup: {

from: 'movies',

localField: 'profiles.history.watched\_movie\_no', foreignField: '\_id',

as: 'Movie'

}}, {$unwind: { path: '$Movie' }}, {$unwind: {

path: '$Movie.genres' }}, {$group: {

\_id: '$Movie.genres', count: {

$count: {} }

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

}}, {$limit: 5}]).pretty()