Question 1

1. db.createCollection("employees", {validator:{$jsonSchema:{bsonType:"object", required:["employeeId", "employeeName"], properties:{employeeId:{bsonType:"int", description:"must be an integer and is required"}, employeeName:{bsonType:"string", description:"must be a string and is required"}, managerName:{bsonType:"string", description:"must be a string if the field exists"}}}}})

2. db.employees.insertMany([{employeeId:Int32(91234), employeeName:"John Chen", managerName:"Wayne Smith"}, {employeeId:Int32(91235), employeeName:"Jane Doe", managerName:"Wayne Smith"}, {employeeId:Int32(91236), employeeName:"John Smith"}, {employeeId:Int32(91237), employeeName:"Patty Ross", managerName:"Tyrone Williams"}, {employeeId:Int32(91238), employeeName:"Alicia Doe", managerName:"Tyrone Williams"}, {employeeId:Int32(91239), employeeName:"Angel Mora", managerName:"Tyrone Williams"}])

3.

A. The data will not be inserted successfully

B. The value for employeeId is a string when it should be an integer. employeeName is not given as a key:value pair even though it is required.

C. MongoBulkWriteError: Document failed validation

```
Result: BulkWriteResult {
  insertedCount: 0,
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {},
  insertedIds: {}
}
Write Errors: [
  WriteError {
    err: {
      index: 0,
      code: 121,
      errmsg: 'Document failed validation',
      errInfo: {
        failingDocumentId: ObjectId('660386f5e2626bd9cea3c44b'),
        details: {
          operatorName: '$jsonSchema',
```

```
schemaRulesNotSatisfied: [
 {
  operatorName: 'properties',
  propertiesNotSatisfied: [
   {
    propertyName: 'employeeId',
    description: 'must be an integer and is required',
    details: [
     {
      operatorName: 'bsonType',
      specifiedAs: { bsonType: 'int' },
      reason: 'type did not match',
      consideredValue: '9A1240',
      consideredType: 'string'
     }
    ]
   }
  ]
 }
```

D. db.employees.insertOne({employeeId:Int32(91240), employeeName:"Random Name", managerName:"Ellie Yoren"})

4.

A. db.employees.find({managerName:null}, {employeeName:1, _id:0})

B. db.employees.find({employeeId:91239})

C. db.employees.find({$and:[{employeeName:{$regex:"Doe"}}, {managerName:{$regex:"Smith"}}]}, {employeeName:1, employeeId:1, _id:0})

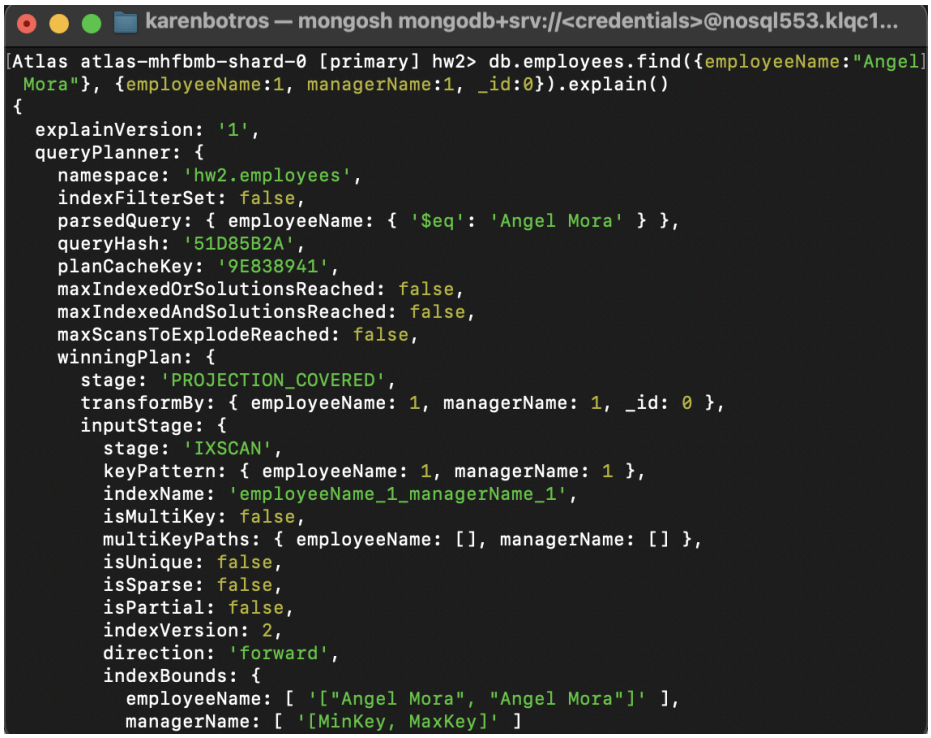
D. db.employees.find({$and:[{employeeName:/Doe$/}, {managerName:{$not:/Smith/}}]}, {employeeName:1, employeeId:1, _id:1})

E. db.employees.updateOne({employeeName:"John Smith"}, {$set:{managerName:"Wayne Smith"}})

5.

A. db.employees.createIndex({employeeName:1, managerName:1})

B. When running the query I ran the query db.employees.find({employeeName:"Angel Mora"}, {employeeName:1, managerName:1, _id:0}). Then I used the same query with .explain() to check if this was a covered query. It says "PROJECTION COVERED" which means it is a covered query and the information given is the most optimized/efficient way to run the query.



Question 2

1. The name of the collection in which IndexStats is being run on is **orders**.

2. There are two indexes. The first is **productName_1_status_**1 and the second is the default index of **_id**. The columns on which the first index is created are **productName** and **status**. The column on which the second index is created is the **_id** column.

3. The first index has been used 4 times. The second index, _id, has been used once, but it never increases.