1. Relational Data because I think, It will easier to manage a data that relate to each other and need to update several times.

2. MongoDB because for this case you have many Data

3. MongoDB uses NoSQL because that was designed to store databases with no specified schema, making significant changes in real time easier.

4. MongoDB for Gaming

   MongoDB's flexible schema means that you can change your data layer just as quickly as you change your game. Need to add new items with new kinds of stats, fix a level, or start storing new player data. So MongoDB stores data as JSON-like documents – you're probably already using JSON in game memory, and storing objects in the same structure makes your life easier.

Name Soravit Tangkulpanich 6310545396

5) Create MongoDB database with following information

Create database and insert data

```
> use Test
< 'switched to db Test'
> db.createCollection("Class")
< { ok: 1 }
> db.Class.insertMany([
      {"name":"Ramesh",  "subject":"maths","marks":87},
      {"name":"Ramesh","subject":"english","marks":59},
      {"name":"Ramesh","subject":"science","marks":77},
      {"name":"Rav","subject":"maths","marks":62},
      {"name":"Rav","subject":"english","marks":83},
      {"name":"Rav","subject":"science","marks":71},
      {"name":"Alison","subject":"maths","marks":84},
      {"name":"Alison","subject":"english","marks":82},
      {"name":"Alison","subject":"science","marks":86},
      {"name":"Steve","subject":"math","marks":81},
      {"name":"Steve","subject":"english","marks":89},
      {"name":"Steve","subject":"science","marks":77},
      {"name":"Jan","subject":"english","marks":0,"reason":"absent"}
      ]
    )
< { acknowledged: true,
    insertedIds:
     { '0': ObjectId("62389790bda6f2ec1c81f6a1"),
       '1': ObjectId("62389790bda6f2ec1c81f6a2"),
       '2': ObjectId("62389790bda6f2ec1c81f6a3"),
       '3': ObjectId("62389790bda6f2ec1c81f6a4"),
       '4': ObjectId("62389790bda6f2ec1c81f6a5"),
       '5': ObjectId("62389790bda6f2ec1c81f6a6"),
       '6': ObjectId("62389790bda6f2ec1c81f6a7"),
       '7': ObjectId("62389790bda6f2ec1c81f6a8"),
       '8': ObjectId("62389790bda6f2ec1c81f6a9"),
       '9': ObjectId("62389790bda6f2ec1c81f6aa"),
       '10': ObjectId("62389790bda6f2ec1c81f6ab"),
       '11': ObjectId("62389790bda6f2ec1c81f6ac"),
       '12': ObjectId("62389790bda6f2ec1c81f6ad") } }
```

5.1 Find the total marks for each student across all subjects.

```
> db.Class.aggregate([{$group:{_id:"$name",totalMark:{$sum:"$marks"}}}])
< { _id: 'Alison', totalMark: 252 }
  { _id: 'Steve', totalMark: 247 }
  { _id: 'Jan', totalMark: 0 }
  { _id: 'Ramesh', totalMark: 223 }
  { _id: 'Rav', totalMark: 216 }
```

5.2 Find the maximum marks scored in each subject.

```
db.Class.aggregate([{$group:{_id:"$subject",maxMark:{$max:"$marks"}}}])
{ _id: 'maths', maxMark: 87 }
{ _id: 'english', maxMark: 89 }
{ _id: 'math', maxMark: 81 }
{ _id: 'science', maxMark: 86 }
```

5.3 Find the minimum marks scored by each student.

```
db.Class.aggregate([{$group:{_id:"$name",minMark:{$max:"$marks"}}}])
{ _id: 'Jan', minMark: 0 }
{ _id: 'Rav', minMark: 83 }
{ _id: 'Steve', minMark: 89 }
{ _id: 'Alison', minMark: 86 }
{ _id: 'Ramesh', minMark: 87 }
```

5.4 Find the top two subjects based on average marks.

```
> db.Class.aggregate([{$group:{_id:"$subject",averageMark:{$avg:"$marks"}}}, {$sort:{"averageMark":-1}},{$limit:2}])
< { _id: 'math', averageMark: 81 }
  { _id: 'science', averageMark: 77.75 }
```